

---

# **EvalML Documentation**

***Release 0.30.2***

**Alteryx, Inc.**

**Aug 16, 2021**



# CONTENTS

<b>1</b>	<b>Install</b>	<b>3</b>
<b>2</b>	<b>Start</b>	<b>5</b>
<b>3</b>	<b>Tutorials</b>	<b>13</b>
<b>4</b>	<b>User Guide</b>	<b>45</b>
<b>5</b>	<b>API Reference</b>	<b>145</b>
<b>6</b>	<b>Release Notes</b>	<b>1269</b>
	<b>Python Module Index</b>	<b>1315</b>
	<b>Index</b>	<b>1319</b>



EvalML is an AutoML library that builds, optimizes, and evaluates machine learning pipelines using domain-specific objective functions.

Combined with [Featuretools](#) and [Compose](#), EvalML can be used to create end-to-end supervised machine learning solutions.



## INSTALL

EvalML is available for Python 3.7 and 3.8 with experimental support 3.9. It can be installed with pip or conda.

### 1.1 Pip with all dependencies

To install evalml with pip, run the following command:

```
pip install evalml
```

### 1.2 Pip with core dependencies

EvalML includes several optional dependencies. The `xgboost` and `catboost` packages support pipelines built around those modeling libraries. The `plotly` and `ipywidgets` packages support plotting functionality in automl searches. These dependencies are recommended, and are included with EvalML by default but are not required in order to install and use EvalML.

EvalML's core dependencies are listed in `core-requirements.txt` in the source code, and optional requirements are listed in `requirements.txt`.

To install EvalML with only the core required dependencies, download the EvalML source [from pypi](#) to access the requirements files. Then run the following:

```
pip install evalml --no-dependencies
pip install -r core-requirements.txt
```

#### 1.2.1 Add-ons

**Update checker** Receive automatic notifications of new EvalML releases

```
pip install evalml[update_checker]
```

## 1.3 Conda with all dependencies

To install evalml with conda run the following command:

```
conda install -c conda-forge evalml
```

## 1.4 Conda with core dependencies

To install evalml with only core dependencies run the following command:

```
conda install -c conda-forge evalml-core
```

## 1.5 Windows

Additionally, if you are using pip to install EvalML, it is recommended you first install the following packages using conda: \* numba (needed for shap and prediction explanations). Install with `conda install -c conda-forge numba` \* graphviz if you're using EvalML's plotting utilities. Install with `conda install -c conda-forge python-graphviz`

The [XGBoost](#) library may not be pip-installable in some Windows environments. If you are encountering installation issues, please try installing XGBoost from [Github](#) before installing EvalML or install evalml with conda.

## 1.6 Mac

In order to run on Mac, [LightGBM](#) requires the OpenMP library to be installed, which can be done with [HomeBrew](#) by running

```
brew install libomp
```

Additionally, graphviz can be installed by running

```
brew install graphviz
```

## 1.7 Python 3.9 support

Evalml can still be installed with pip in python 3.9 but note that `sktime`, one of our dependencies, will not be installed because that library does not yet support python 3.9. This means the `PolynomialDetrending` component will not be usable in python 3.9. You can try to install `sktime` [from source](#) in python 3.9 to use the `PolynomialDetrending` component but be warned that we only test it in python 3.7 and 3.8.



**START**

In this guide, we'll show how you can use EvalML to automatically find the best pipeline for predicting whether or not a credit card transaction is fraudulent. Along the way, we'll highlight EvalML's built-in tools and features for understanding and interacting with the search process.

```
[1]: import evalml
      from evalml import AutoMLSearch
      from evalml.utils import infer_feature_types
```

First, we load in the features and outcomes we want to use to train our model.

```
[2]: X, y = evalml.demos.load_fraud(n_rows=250)
```

```

      Number of Features
Boolean                1
Categorical             6
Numeric                5

Number of training examples: 250
Targets
False    88.40%
True     11.60%
Name: fraud, dtype: object
```

First, we will clean the data. Since EvalML accepts a pandas input, it can run type inference on this data directly. Since we'd like to change the types inferred by EvalML, we can use the `infer_feature_types` utility method. Here's what we're going to do with the following dataset:

- Reformat the `expiration_date` column so it reflects a more familiar date format.
- Cast the `lat` and `lng` columns from float to str.
- Use `infer_feature_types` to specify what types certain columns should be. For example, to avoid having the `provider` column be inferred as natural language text, we have specified it as a categorical column instead.

The `infer_feature_types` utility method takes a pandas or numpy input and converts it to a pandas dataframe with a [Woodwork](#) accessor, providing us with flexibility to cast the data as necessary.

```
[3]: X.ww['expiration_date'] = X['expiration_date'].apply(lambda x: '20{}-01-{}'.format(x.
      ↪split("/") [1], x.split("/") [0]))
      X = infer_feature_types(X, feature_types= {'store_id': 'categorical',
      'expiration_date': 'datetime',
      'lat': 'categorical',
      'lng': 'categorical',
      'provider': 'categorical'})

      X.ww
```

```
[3]:
```

	Physical Type	Logical Type	Semantic Tag(s)
Column			
card_id	int64	Integer	['numeric']
store_id	int64	Integer	['numeric']
datetime	datetime64[ns]	Datetime	[]
amount	int64	Integer	['numeric']
currency	category	Categorical	['category']
customer_present	bool	Boolean	[]
expiration_date	category	Categorical	['category']
provider	category	Categorical	['category']
lat	float64	Double	['numeric']
lng	float64	Double	['numeric']
region	category	Categorical	['category']
country	category	Categorical	['category']

In order to validate the results of the pipeline creation and optimization process, we will save some of our data as a holdout set.

```
[4]: X_train, X_holdout, y_train, y_holdout = evalml.preprocessing.split_data(X, y,
    ↪problem_type='binary', test_size=.2)
```

**Note:** To provide data to EvalML, it is recommended that you initialize a woodwork accessor so that you control how EvalML will treat each feature, such as as a numeric feature, a categorical feature, a text feature or other type of feature. Consult the [the Woodwork project](#) for help on how to do this. Here, `split_data()` returns dataframes with woodwork accessors.

EvalML has many options to configure the pipeline search. At the minimum, we need to define an objective function. For simplicity, we will use the F1 score in this example. However, the real power of EvalML is in using domain-specific *objective functions* or *building your own*.

Below EvalML utilizes Bayesian optimization (EvalML's default optimizer) to search and find the best pipeline defined by the given objective.

EvalML provides a number of parameters to control the search process. `max_batches` is one of the parameters which controls the stopping criterion for the AutoML search. It indicates the maximum number of rounds of AutoML to evaluate, where each round may train and score a variable number of pipelines. In this example, `max_batches` is set to 1.

**\*\* Graphing methods, like AutoMLSearch, on Jupyter Notebook and Jupyter Lab require [ipywidgets](#) to be installed.**

**\*\* If graphing on Jupyter Lab, [jupyterlab-plotly](#) required. To download this, make sure you have [npm](#) installed.**

```
[5]: automl = AutoMLSearch(X_train=X_train, y_train=y_train,
    problem_type='binary', objective='f1', max_batches=1)
```

```
Generating pipelines to search over...
8 pipelines ready for search.
```

When we call `search()`, the search for the best pipeline will begin. There is no need to wrangle with missing data or categorical variables as EvalML includes various preprocessing steps (like imputation, one-hot encoding, feature selection) to ensure you're getting the best results. As long as your data is in a single table, EvalML can handle it. If not, you can reduce your data to a single table by utilizing [Featuretools](#) and its Entity Sets.

You can find more information on pipeline components and how to integrate your own custom pipelines into EvalML [here](#).

```
[6]: automl.search()
```

```
*****
* Beginning pipeline search *
*****
```

Optimizing for F1.  
Greater score is better.

Using SequentialEngine to train and score pipelines.  
Searching up to 1 batches for a total of 9 pipelines.  
Allowed model families: lightgbm, extra\_trees, decision\_tree, xgboost, random\_forest,  
↳linear\_model, catboost

```
FigureWidget({
  'data': [{'mode': 'lines+markers',
            'name': 'Best Score',
            'type'...
```

Evaluating Baseline Pipeline: Mode Baseline Binary Classification Pipeline  
Mode Baseline Binary Classification Pipeline:

Starting cross validation  
Finished cross validation - mean F1: 0.000

```
*****
* Evaluating Batch Number 1 *
*****
```

Elastic Net Classifier w/ Imputer + DateTime Featurization Component + One Hot  
↳Encoder + SMOTENC Oversampler + Standard Scaler:  
Starting cross validation  
Finished cross validation - mean F1: 0.263  
High coefficient of variation (cv >= 0.2) within cross validation scores.  
Elastic Net Classifier w/ Imputer + DateTime Featurization Component + One  
↳Hot Encoder + SMOTENC Oversampler + Standard Scaler may not perform as estimated on  
↳unseen data.

Decision Tree Classifier w/ Imputer + DateTime Featurization Component + One Hot  
↳Encoder + SMOTENC Oversampler:  
Starting cross validation  
Finished cross validation - mean F1: 0.448  
High coefficient of variation (cv >= 0.2) within cross validation scores.  
Decision Tree Classifier w/ Imputer + DateTime Featurization Component + One  
↳Hot Encoder + SMOTENC Oversampler may not perform as estimated on unseen data.

Random Forest Classifier w/ Imputer + DateTime Featurization Component + One Hot  
↳Encoder + SMOTENC Oversampler:  
Starting cross validation  
Finished cross validation - mean F1: 0.500

LightGBM Classifier w/ Imputer + DateTime Featurization Component + One Hot Encoder +  
↳SMOTENC Oversampler:  
Starting cross validation  
Finished cross validation - mean F1: 0.417  
High coefficient of variation (cv >= 0.2) within cross validation scores.  
LightGBM Classifier w/ Imputer + DateTime Featurization Component + One Hot  
↳Encoder + SMOTENC Oversampler may not perform as estimated on unseen data.

Logistic Regression Classifier w/ Imputer + DateTime Featurization Component + One  
↳Hot Encoder + SMOTENC Oversampler + Standard Scaler:  
Starting cross validation  
Finished cross validation - mean F1: 0.233

```
/home/docs/checkouts/readthedocs.org/user_builds/feature-labs-inc-evalml/envs/v0.30.2/
↳lib/python3.8/site-packages/xgboost/sklearn.py:1146: UserWarning:
```

The use of label encoder in XGBClassifier is deprecated and will be removed in a  
↳future release. To remove this warning, do the following: 1) Pass option use\_label\_  
↳encoder=False when constructing XGBClassifier object; and 2) Encode your labels (y)  
↳as integers starting with 0, i.e. 0, 1, 2, ..., [num\_class - 1].

```
[20:22:13] WARNING: ../src/learner.cc:1095: Starting in XGBoost 1.3.0, the default
↳evaluation metric used with the objective 'binary:logistic' was changed from 'error
↳' to 'logloss'. Explicitly set eval_metric if you'd like to restore the old
↳behavior.
```

```
/home/docs/checkouts/readthedocs.org/user_builds/feature-labs-inc-evalml/envs/v0.30.2/
↳lib/python3.8/site-packages/xgboost/sklearn.py:1146: UserWarning:
```

The use of label encoder in XGBClassifier is deprecated and will be removed in a  
↳future release. To remove this warning, do the following: 1) Pass option use\_label\_  
↳encoder=False when constructing XGBClassifier object; and 2) Encode your labels (y)  
↳as integers starting with 0, i.e. 0, 1, 2, ..., [num\_class - 1].

```
[20:22:13] WARNING: ../src/learner.cc:1095: Starting in XGBoost 1.3.0, the default
↳evaluation metric used with the objective 'binary:logistic' was changed from 'error
↳' to 'logloss'. Explicitly set eval_metric if you'd like to restore the old
↳behavior.
```

```
/home/docs/checkouts/readthedocs.org/user_builds/feature-labs-inc-evalml/envs/v0.30.2/
↳lib/python3.8/site-packages/xgboost/sklearn.py:1146: UserWarning:
```

The use of label encoder in XGBClassifier is deprecated and will be removed in a  
↳future release. To remove this warning, do the following: 1) Pass option use\_label\_  
↳encoder=False when constructing XGBClassifier object; and 2) Encode your labels (y)  
↳as integers starting with 0, i.e. 0, 1, 2, ..., [num\_class - 1].

```
[20:22:14] WARNING: ../src/learner.cc:1095: Starting in XGBoost 1.3.0, the default
↳evaluation metric used with the objective 'binary:logistic' was changed from 'error
↳' to 'logloss'. Explicitly set eval_metric if you'd like to restore the old
↳behavior.
```

```
XGBoost Classifier w/ Imputer + DateTime Featurization Component + One Hot Encoder +
↳SMOTENC Oversampler:
```

```
Starting cross validation
```

```
Finished cross validation - mean F1: 0.372
```

```
High coefficient of variation (cv >= 0.2) within cross validation scores.
```

```
XGBoost Classifier w/ Imputer + DateTime Featurization Component + One Hot
```

```
↳Encoder + SMOTENC Oversampler may not perform as estimated on unseen data.
```

```
Extra Trees Classifier w/ Imputer + DateTime Featurization Component + One Hot
```

```
↳Encoder + SMOTENC Oversampler:
```

```
Starting cross validation
```

```
Finished cross validation - mean F1: 0.263
```

```
High coefficient of variation (cv >= 0.2) within cross validation scores.
```

```
Extra Trees Classifier w/ Imputer + DateTime Featurization Component + One
```

```
↳Hot Encoder + SMOTENC Oversampler may not perform as estimated on unseen data.
```

```
CatBoost Classifier w/ Imputer + DateTime Featurization Component + SMOTENC
```

```
↳Oversampler:
```

```
Starting cross validation
```

```
Finished cross validation - mean F1: 0.646
```

```
High coefficient of variation (cv >= 0.2) within cross validation scores.
```

(continues on next page)

(continued from previous page)

```

CatBoost Classifier w/ Imputer + DateTime Featurization Component + SMOTENC
↪Oversampler may not perform as estimated on unseen data.

Search finished after 00:22
Best pipeline: CatBoost Classifier w/ Imputer + DateTime Featurization Component +
↪SMOTENC Oversampler
Best pipeline F1: 0.646154

```

We also provide a *standalone* ``search`` method `<./generated/evalml.automl.search.html>` which does all of the above in a single line, and returns the `AutoMLSearch` instance and data check results. If there were data check errors, AutoML will not be run and no `AutoMLSearch` instance will be returned.

After the search is finished we can view all of the pipelines searched, ranked by score. Internally, EvalML performs cross validation to score the pipelines. If it notices a high variance across cross validation folds, it will warn you. EvalML also provides additional *data checks* to analyze your data to assist you in producing the best performing pipeline.

```
[7]: automl.rankings
```

```

[7]:      id      pipeline_name  search_order  \
0      8  CatBoost Classifier w/ Imputer + DateTime Feat...      8
1      3  Random Forest Classifier w/ Imputer + DateTime...      3
2      2  Decision Tree Classifier w/ Imputer + DateTime...      2
3      4  LightGBM Classifier w/ Imputer + DateTime Feat...      4
4      6  XGBoost Classifier w/ Imputer + DateTime Featu...      6
5      7  Extra Trees Classifier w/ Imputer + DateTime F...      7
6      1  Elastic Net Classifier w/ Imputer + DateTime F...      1
7      5  Logistic Regression Classifier w/ Imputer + Da...      5
8      0      Mode Baseline Binary Classification Pipeline      0

      mean_cv_score  standard_deviation_cv_score  validation_score  \
0          0.646154          0.213175          0.769231
1          0.500000          0.100000          0.400000
2          0.447992          0.328221          0.769231
3          0.416667          0.144338          0.500000
4          0.372222          0.256219          0.666667
5          0.263091          0.233224          0.444444
6          0.262500          0.073006          0.266667
7          0.232828          0.044955          0.266667
8          0.000000          0.000000          0.000000

      percent_better_than_baseline  high_variance_cv  \
0                64.615385                True
1                50.000000                False
2                44.799226                True
3                41.666667                True
4                37.222222                True
5                26.309068                True
6                26.250000                True
7                23.282828                False
8                 0.000000                False

                        parameters
0  {'Imputer': {'categorical_impute_strategy': 'm...
1  {'Imputer': {'categorical_impute_strategy': 'm...
2  {'Imputer': {'categorical_impute_strategy': 'm...
3  {'Imputer': {'categorical_impute_strategy': 'm...

```

(continues on next page)

(continued from previous page)

```

4 {'Imputer': {'categorical_impute_strategy': 'm...
5 {'Imputer': {'categorical_impute_strategy': 'm...
6 {'Imputer': {'categorical_impute_strategy': 'm...
7 {'Imputer': {'categorical_impute_strategy': 'm...
8 {'Baseline Classifier': {'strategy': 'mode'}}

```

If we are interested in see more details about the pipeline, we can view a summary description using the `id` from the rankings table:

```
[8]: automl.describe_pipeline(3)
```

```

*****
* Random Forest Classifier w/ Imputer + DateTime Featurization Component + One Hot_
↳Encoder + SMOTENC Oversampler *
*****

Problem Type: binary
Model Family: Random Forest

Pipeline Steps
=====
1. Imputer
    * categorical_impute_strategy : most_frequent
    * numeric_impute_strategy : mean
    * categorical_fill_value : None
    * numeric_fill_value : None
2. DateTime Featurization Component
    * features_to_extract : ['year', 'month', 'day_of_week', 'hour']
    * encode_as_categories : False
    * date_index : None
3. One Hot Encoder
    * top_n : 10
    * features_to_encode : None
    * categories : None
    * drop : if_binary
    * handle_unknown : ignore
    * handle_missing : error
4. SMOTENC Oversampler
    * sampling_ratio : 0.25
    * k_neighbors_default : 5
    * n_jobs : -1
    * sampling_ratio_dict : None
    * categorical_features : [3, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21,
↳22, 23, 24, 25, 26, 27, 28, 29, 30, 31, 32, 33, 34, 35, 36, 37, 38, 39, 40, 41, 42,
↳43, 44, 45, 46, 47, 48, 49, 50, 51, 52, 53, 54, 55, 56, 57, 58, 59]
    * k_neighbors : 5
5. Random Forest Classifier
    * n_estimators : 100
    * max_depth : 6
    * n_jobs : -1

Training
=====
Training for binary problems.
Objective to optimize binary classification pipeline thresholds for: <evalml.
↳objectives.standard_metrics.F1 object at 0x7ff0d0a40460>

```

(continues on next page)

(continued from previous page)

Total training time (including CV): 2.7 seconds

Cross Validation

	F1	MCC Binary	Log Loss Binary	Gini	AUC	Precision	Balanced_
Accuracy Binary	Accuracy Binary	# Training	# Validation				
0	0.400	0.476	0.264	0.648	0.824	1.000	
→ 0.625		0.910	133	67			
1	0.500	0.490	0.333	0.424	0.712	0.750	
→ 0.679		0.910	133	67			
2	0.600	0.634	0.260	0.758	0.879	1.000	
→ 0.714		0.939	134	66			
mean	0.500	0.533	0.286	0.610	0.805	0.917	
→ 0.673		0.920	-	-			
std	0.100	0.087	0.041	0.170	0.085	0.144	
→ 0.045		0.017	-	-			
coef of var	0.200	0.163	0.144	0.279	0.106	0.157	
→ 0.067		0.018	-	-			

We can also view the pipeline parameters directly:

```
[9]: pipeline = automl.get_pipeline(3)
print(pipeline.parameters)

{'Imputer': {'categorical_impute_strategy': 'most_frequent', 'numeric_impute_strategy': 'mean', 'categorical_fill_value': None, 'numeric_fill_value': None}, 'DateTimeFeaturization Component': {'features_to_extract': ['year', 'month', 'day_of_week', 'hour'], 'encode_as_categories': False, 'date_index': None}, 'One Hot Encoder': {'top_n': 10, 'features_to_encode': None, 'categories': None, 'drop': 'if_binary', 'handle_unknown': 'ignore', 'handle_missing': 'error'}, 'SMOTENC Oversampler': {'sampling_ratio': 0.25, 'k_neighbors_default': 5, 'n_jobs': -1, 'sampling_ratio_dict': None, 'categorical_features': [3, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31, 32, 33, 34, 35, 36, 37, 38, 39, 40, 41, 42, 43, 44, 45, 46, 47, 48, 49, 50, 51, 52, 53, 54, 55, 56, 57, 58, 59], 'k_neighbors': 5}, 'Random Forest Classifier': {'n_estimators': 100, 'max_depth': 6, 'n_jobs': -1}}
```

We can now select the best pipeline and score it on our holdout data:

```
[10]: pipeline = automl.best_pipeline
pipeline.score(X_holdout, y_holdout, ["f1"])

[10]: OrderedDict([('F1', 0.5)])
```

We can also visualize the structure of the components contained by the pipeline:

```
[11]: pipeline.graph()

[11]:
```





## TUTORIALS

Below are examples of how to apply EvalML to a variety of problems:

### 3.1 Building a Fraud Prediction Model with EvalML

In this demo, we will build an optimized fraud prediction model using EvalML. To optimize the pipeline, we will set up an objective function to minimize the percentage of total transaction value lost to fraud. At the end of this demo, we also show you how introducing the right objective during the training results in a much better than using a generic machine learning metric like AUC.

```
[1]: import evalml
      from evalml import AutoMLSearch
      from evalml.objectives import FraudCost
```

#### 3.1.1 Configure “Cost of Fraud”

To optimize the pipelines toward the specific business needs of this model, we can set our own assumptions for the cost of fraud. These parameters are

- `retry_percentage` - what percentage of customers will retry a transaction if it is declined?
- `interchange_fee` - how much of each successful transaction do you collect?
- `fraud_payout_percentage` - the percentage of fraud will you be unable to collect
- `amount_col` - the column in the data the represents the transaction amount

Using these parameters, EvalML determines attempt to build a pipeline that will minimize the financial loss due to fraud.

```
[2]: fraud_objective = FraudCost(retry_percentage=.5,
                                interchange_fee=.02,
                                fraud_payout_percentage=.75,
                                amount_col='amount')
```

### 3.1.2 Search for best pipeline

In order to validate the results of the pipeline creation and optimization process, we will save some of our data as the holdout set.

```
[3]: X, y = evalml.demos.load_fraud(n_rows=5000)
```

```

      Number of Features
Boolean                1
Categorical             6
Numeric                5

Number of training examples: 5000
Targets
False      86.20%
True       13.80%
Name: fraud, dtype: object
```

EvalML natively supports one-hot encoding. Here we keep 1 out of the 6 categorical columns to decrease computation time.

```
[4]: cols_to_drop = ['datetime', 'expiration_date', 'country', 'region', 'provider']
    for col in cols_to_drop:
        X.ww.pop(col)
```

```
X_train, X_holdout, y_train, y_holdout = evalml.preprocessing.split_data(X, y,
    problem_type='binary', test_size=0.2, random_seed=0)
```

```
X.ww
```

```
[4]:
```

Column	Physical Type	Logical Type	Semantic Tag(s)
card_id	int64	Integer	['numeric']
store_id	int64	Integer	['numeric']
amount	int64	Integer	['numeric']
currency	category	Categorical	['category']
customer_present	bool	Boolean	[]
lat	float64	Double	['numeric']
lng	float64	Double	['numeric']

Because the fraud labels are binary, we will use `AutoMLSearch(X_train=X_train, y_train=y_train, problem_type='binary')`. When we call `.search()`, the search for the best pipeline will begin.

```
[5]: automl = AutoMLSearch(X_train=X_train, y_train=y_train,
    problem_type='binary',
    objective=fraud_objective,
    additional_objectives=['auc', 'f1', 'precision'],
    allowed_model_families=["random_forest", "linear_model"],
    max_batches=1,
    optimize_thresholds=True)
```

```
automl.search()
```

```
Generating pipelines to search over...
3 pipelines ready for search.
```

```
*****
* Beginning pipeline search *
*****
```

(continues on next page)

(continued from previous page)

Optimizing for Fraud Cost.  
Lower score is better.

Using SequentialEngine to train and score pipelines.  
Searching up to 1 batches for a total of 4 pipelines.  
Allowed model families: random\_forest, linear\_model

```
FigureWidget({
  'data': [{'mode': 'lines+markers',
            'name': 'Best Score',
            'type'...
```

Evaluating Baseline Pipeline: Mode Baseline Binary Classification Pipeline  
Mode Baseline Binary Classification Pipeline:

Starting cross validation  
Finished cross validation - mean Fraud Cost: 0.790

```
*****
* Evaluating Batch Number 1 *
*****
```

Elastic Net Classifier w/ Imputer + One Hot Encoder + SMOTENC Oversampler + Standard  
↪Scaler:

Starting cross validation  
Finished cross validation - mean Fraud Cost: 0.286  
High coefficient of variation (cv >= 0.2) within cross validation scores.

Elastic Net Classifier w/ Imputer + One Hot Encoder + SMOTENC Oversampler +  
↪Standard Scaler may not perform as estimated on unseen data.

Random Forest Classifier w/ Imputer + One Hot Encoder + SMOTENC Oversampler:

Starting cross validation  
Finished cross validation - mean Fraud Cost: 0.289  
High coefficient of variation (cv >= 0.2) within cross validation scores.

Random Forest Classifier w/ Imputer + One Hot Encoder + SMOTENC Oversampler  
↪may not perform as estimated on unseen data.

Logistic Regression Classifier w/ Imputer + One Hot Encoder + SMOTENC Oversampler +  
↪Standard Scaler:

Starting cross validation  
Finished cross validation - mean Fraud Cost: 0.286  
High coefficient of variation (cv >= 0.2) within cross validation scores.

Logistic Regression Classifier w/ Imputer + One Hot Encoder + SMOTENC  
↪Oversampler + Standard Scaler may not perform as estimated on unseen data.

Search finished after 00:14

Best pipeline: Elastic Net Classifier w/ Imputer + One Hot Encoder + SMOTENC  
↪Oversampler + Standard Scaler

Best pipeline Fraud Cost: 0.285505

## View rankings and select pipelines

Once the fitting process is done, we can see all of the pipelines that were searched, ranked by their score on the fraud detection objective we defined.

```
[6]: automl.rankings
```

```
[6]:      id      pipeline_name  search_order  \
0    1  Elastic Net Classifier w/ Imputer + One Hot En...      1
1    3  Logistic Regression Classifier w/ Imputer + On...      3
2    2  Random Forest Classifier w/ Imputer + One Hot ...      2
3    0      Mode Baseline Binary Classification Pipeline      0

      mean_cv_score  standard_deviation_cv_score  validation_score  \
0          0.285505             0.436205          0.040953
1          0.285505             0.436205          0.040953
2          0.289310             0.434436          0.790953
3          0.789648             0.001136          0.790953

      percent_better_than_baseline  high_variance_cv  \
0                50.414342             True
1                50.414342             True
2                50.033835             True
3                 0.000000             False

                        parameters
0  {'Imputer': {'categorical_impute_strategy': 'm...
1  {'Imputer': {'categorical_impute_strategy': 'm...
2  {'Imputer': {'categorical_impute_strategy': 'm...
3    {'Baseline Classifier': {'strategy': 'mode'}}
```

To select the best pipeline we can call `automl.best_pipeline`.

```
[7]: best_pipeline = automl.best_pipeline
```

## Describe pipelines

We can get more details about any pipeline created during the search process, including how it performed on other objective functions, by calling the `describe_pipeline` method and passing the `id` of the pipeline of interest.

```
[8]: automl.describe_pipeline(automl.rankings.iloc[1]["id"])
```

```
*****
* Logistic Regression Classifier w/ Imputer + One Hot Encoder + SMOTENC Oversampler +
↪Standard Scaler *
*****

Problem Type: binary
Model Family: Linear

Pipeline Steps
=====
1. Imputer
    * categorical_impute_strategy : most_frequent
    * numeric_impute_strategy : mean
    * categorical_fill_value : None
```

(continues on next page)

(continued from previous page)

```

    * numeric_fill_value : None
2. One Hot Encoder
    * top_n : 10
    * features_to_encode : None
    * categories : None
    * drop : if_binary
    * handle_unknown : ignore
    * handle_missing : error
3. SMOTENC Oversampler
    * sampling_ratio : 0.25
    * k_neighbors_default : 5
    * n_jobs : -1
    * sampling_ratio_dict : None
    * categorical_features : [3, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15]
    * k_neighbors : 5
4. Standard Scaler
5. Logistic Regression Classifier
    * penalty : l2
    * C : 1.0
    * n_jobs : -1
    * multi_class : auto
    * solver : lbfgs

```

Training

=====

Training for binary problems.

Objective to optimize binary classification pipeline thresholds for: <evalml.

→objectives.fraud\_cost.FraudCost object at 0x7fa88ad44b80>

Total training time (including CV): 3.8 seconds

Cross Validation

-----

	Fraud Cost	AUC	F1	Precision	# Training	# Validation
0	0.041	0.854	0.011	1.000	2,666	1,334
1	0.789	0.799	0.000	0.000	2,667	1,333
2	0.026	0.828	0.355	0.228	2,667	1,333
mean	0.286	0.827	0.122	0.409	-	-
std	0.436	0.027	0.202	0.524	-	-
coef of var	1.528	0.033	1.656	1.280	-	-

### 3.1.3 Evaluate on holdout data

Finally, since the best pipeline is already trained, we evaluate it on the holdout data.

Now, we can score the pipeline on the holdout data using both our fraud cost objective and the AUC (Area under the ROC Curve) objective.

```
[9]: best_pipeline.score(X_holdout, y_holdout, objectives=["auc", fraud_objective])
```

```
[9]: OrderedDict([('AUC', 0.8097111537038906),
                  ('Fraud Cost', 0.0015780080694855455)])
```

### 3.1.4 Why optimize for a problem-specific objective?

To demonstrate the importance of optimizing for the right objective, let's search for another pipeline using AUC, a common machine learning metric. After that, we will score the holdout data using the fraud cost objective to see how the best pipelines compare.

```
[10]: automl_auc = AutoMLSearch(X_train=X_train, y_train=y_train,
                                problem_type='binary',
                                objective='auc',
                                additional_objectives=['f1', 'precision'],
                                max_batches=1,
                                allowed_model_families=["random_forest", "linear_model"],
                                optimize_thresholds=True)

automl_auc.search()
```

Generating pipelines to search over...  
3 pipelines ready for search.

```
*****
* Beginning pipeline search *
*****

Optimizing for AUC.
Greater score is better.

Using SequentialEngine to train and score pipelines.
Searching up to 1 batches for a total of 4 pipelines.
Allowed model families: random_forest, linear_model
```

```
FigureWidget({
  'data': [{'mode': 'lines+markers',
            'name': 'Best Score',
            'type': ...}]
})
```

Evaluating Baseline Pipeline: Mode Baseline Binary Classification Pipeline  
Mode Baseline Binary Classification Pipeline:  
Starting cross validation  
Finished cross validation - mean AUC: 0.500

```
*****
* Evaluating Batch Number 1 *
*****
```

Elastic Net Classifier w/ Imputer + One Hot Encoder + SMOTENC Oversampler + Standard\_  
↪Scaler:  
Starting cross validation  
Finished cross validation - mean AUC: 0.827

Random Forest Classifier w/ Imputer + One Hot Encoder + SMOTENC Oversampler:  
Starting cross validation  
Finished cross validation - mean AUC: 0.853

Logistic Regression Classifier w/ Imputer + One Hot Encoder + SMOTENC Oversampler +\_  
↪Standard Scaler:  
Starting cross validation  
Finished cross validation - mean AUC: 0.827

Search finished after 00:07  
Best pipeline: Random Forest Classifier w/ Imputer + One Hot Encoder + SMOTENC\_  
↪Oversampler

(continues on next page)

(continued from previous page)

```
Best pipeline AUC: 0.853164
```

Like before, we can look at the rankings of all of the pipelines searched and pick the best pipeline.

```
[11]: automl_auc.rankings
```

```
[11]:      id      pipeline_name  search_order  \
0    2  Random Forest Classifier w/ Imputer + One Hot ...      2
1    1  Elastic Net Classifier w/ Imputer + One Hot En...      1
2    3  Logistic Regression Classifier w/ Imputer + On...      3
3    0      Mode Baseline Binary Classification Pipeline      0

      mean_cv_score  standard_deviation_cv_score  validation_score  \
0          0.853164              0.010011          0.863894
1          0.827102              0.027002          0.853667
2          0.826675              0.027444          0.853544
3          0.500000              0.000000          0.500000

      percent_better_than_baseline  high_variance_cv  \
0                35.316397              False
1                32.710221              False
2                32.667497              False
3                 0.000000              False

                                parameters
0  {'Imputer': {'categorical_impute_strategy': 'm...
1  {'Imputer': {'categorical_impute_strategy': 'm...
2  {'Imputer': {'categorical_impute_strategy': 'm...
3      {'Baseline Classifier': {'strategy': 'mode'}}
```

```
[12]: best_pipeline_auc = automl_auc.best_pipeline
```

```
[13]: # get the fraud score on holdout data
best_pipeline_auc.score(X_holdout, y_holdout, objectives=["auc", fraud_objective])
```

```
[13]: OrderedDict([('AUC', 0.8644456773933219),
                  ('Fraud Cost', 0.026054721586011263)])
```

```
[14]: # fraud score on fraud optimized again
best_pipeline.score(X_holdout, y_holdout, objectives=["auc", fraud_objective])
```

```
[14]: OrderedDict([('AUC', 0.8097111537038906),
                  ('Fraud Cost', 0.0015780080694855455)])
```

When we optimize for AUC, we can see that the AUC score from this pipeline performs better compared to the AUC score from the pipeline optimized for fraud cost; however, the losses due to fraud are a much larger percentage of the total transaction amount when optimized for AUC and much smaller when optimized for fraud cost. As a result, we lose a noticeable percentage of the total transaction amount by not optimizing for fraud cost specifically.

Optimizing for AUC does not take into account the user-specified `retry_percentage`, `interchange_fee`, `fraud_payout_percentage` values, which could explain the decrease in fraud performance. Thus, the best pipelines may produce the highest AUC but may not actually reduce the amount loss due to your specific type fraud.

This example highlights how performance in the real world can diverge greatly from machine learning metrics.

## 3.2 Building a Lead Scoring Model with EvalML

In this demo, we will build an optimized lead scoring model using EvalML. To optimize the pipeline, we will set up an objective function to maximize the revenue generated with true positives while taking into account the cost of false positives. At the end of this demo, we also show you how introducing the right objective during the training is significantly better than using a generic machine learning metric like AUC.

```
[1]: import evalml
      from evalml import AutoMLSearch
      from evalml.objectives import LeadScoring
```

### 3.2.1 Configure LeadScoring

To optimize the pipelines toward the specific business needs of this model, you can set your own assumptions for how much value is gained through true positives and the cost associated with false positives. These parameters are

- `true_positive` - dollar amount to be gained with a successful lead
- `false_positive` - dollar amount to be lost with an unsuccessful lead

Using these parameters, EvalML builds a pipeline that will maximize the amount of revenue per lead generated.

```
[2]: lead_scoring_objective = LeadScoring(
      true_positives=100,
      false_positives=-5
    )
```

### 3.2.2 Dataset

We will be utilizing a dataset detailing a customer's job, country, state, zip, online action, the dollar amount of that action and whether they were a successful lead.

```
[3]: from urllib.request import urlopen
      import pandas as pd
      import woodwork as ww
      customers_data = urlopen('https://featurelabs-static.s3.amazonaws.com/lead_scoring_ml_
      ↪apps/customers.csv')
      interactions_data = urlopen('https://featurelabs-static.s3.amazonaws.com/lead_scoring_
      ↪ml_apps/interactions.csv')
      leads_data = urlopen('https://featurelabs-static.s3.amazonaws.com/lead_scoring_ml_
      ↪apps/previous_leads.csv')
      customers = pd.read_csv(customers_data)
      interactions = pd.read_csv(interactions_data)
      leads = pd.read_csv(leads_data)

      X = customers.merge(interactions, on='customer_id').merge(leads, on='customer_id')
      y = X['label']
      X = X.drop(['customer_id', 'date_registered', 'birthday', 'phone', 'email',
                  'owner', 'company', 'id', 'time_x',
                  'session', 'referrer', 'time_y', 'label', 'country'], axis=1)
      display(X.head())
```

	job	state	zip	action	amount
0	Engineer, mining	NY	60091.0	page_view	NaN
1	Psychologist, forensic	CA	NaN	purchase	135.23

(continues on next page)



(continued from previous page)

2	Psychologist, forensic	CA	NaN	page_view	NaN
3	Air cabin crew	NaN	60091.0	download	NaN
4	Air cabin crew	NaN	60091.0	page_view	NaN

We will convert our data into Woodwork data structures. Doing so enables us to have more control over the types passed to and inferred by AutoML.

```
[4]: X.ww.init(semantic_tags={'job': 'category'}, logical_types={'job': 'Categorical'})
y = ww.init_series(y)
X.ww
```

```
[4]:
```

	Physical Type	Logical Type	Semantic Tag(s)
Column			
job	category	Categorical	['category']
state	category	Categorical	['category']
zip	float64	Double	['numeric']
action	category	Categorical	['category']
amount	float64	Double	['numeric']

### 3.2.3 Search for the best pipeline

In order to validate the results of the pipeline creation and optimization process, we will save some of our data as a holdout set.

EvalML natively supports one-hot encoding and imputation so the above NaN and categorical values will be taken care of.

```
[5]: X_train, X_holdout, y_train, y_holdout = evalml.preprocessing.split_data(X, y,
↪problem_type='binary', test_size=0.2, random_seed=0)

X.ww
```

```
[5]:
```

	Physical Type	Logical Type	Semantic Tag(s)
Column			
job	category	Categorical	['category']
state	category	Categorical	['category']
zip	float64	Double	['numeric']
action	category	Categorical	['category']
amount	float64	Double	['numeric']

Because the lead scoring labels are binary, we will use set the problem type to “binary”. When we call `.search()`, the search for the best pipeline will begin.

```
[6]: automl = AutoMLSearch(X_train=X_train, y_train=y_train,
                           problem_type='binary',
                           objective=lead_scoring_objective,
                           additional_objectives=['auc'],
                           allowed_model_families=["catboost", "random_forest", "linear_
↪model"],
                           max_batches=1)

automl.search()

Generating pipelines to search over...
4 pipelines ready for search.
```

(continues on next page)

(continued from previous page)

```
*****
* Beginning pipeline search *
*****
```

```
Optimizing for Lead Scoring.
Greater score is better.
```

```
Using SequentialEngine to train and score pipelines.
Searching up to 1 batches for a total of 5 pipelines.
Allowed model families: linear_model, catboost, random_forest
```

```
FigureWidget({
  'data': [{'mode': 'lines+markers',
            'name': 'Best Score',
            'type'...
```

```
Evaluating Baseline Pipeline: Mode Baseline Binary Classification Pipeline
Mode Baseline Binary Classification Pipeline:
```

```
  Starting cross validation
  Finished cross validation - mean Lead Scoring: 0.000
```

```
*****
* Evaluating Batch Number 1 *
*****
```

```
Elastic Net Classifier w/ Imputer + One Hot Encoder + SMOTENC Oversampler + Standard_
↳Scaler:
```

```
  Starting cross validation
  Finished cross validation - mean Lead Scoring: 0.014
  High coefficient of variation (cv >= 0.2) within cross validation scores.
  Elastic Net Classifier w/ Imputer + One Hot Encoder + SMOTENC Oversampler +
↳Standard Scaler may not perform as estimated on unseen data.
```

```
Random Forest Classifier w/ Imputer + One Hot Encoder + SMOTENC Oversampler:
```

```
  Starting cross validation
  Finished cross validation - mean Lead Scoring: 0.015
  High coefficient of variation (cv >= 0.2) within cross validation scores.
  Random Forest Classifier w/ Imputer + One Hot Encoder + SMOTENC Oversampler_
↳may not perform as estimated on unseen data.
```

```
Logistic Regression Classifier w/ Imputer + One Hot Encoder + SMOTENC Oversampler +
↳Standard Scaler:
```

```
  Starting cross validation
  Finished cross validation - mean Lead Scoring: 0.014
  High coefficient of variation (cv >= 0.2) within cross validation scores.
  Logistic Regression Classifier w/ Imputer + One Hot Encoder + SMOTENC_
↳Oversampler + Standard Scaler may not perform as estimated on unseen data.
```

```
CatBoost Classifier w/ Imputer + SMOTENC Oversampler:
```

```
  Starting cross validation
  Finished cross validation - mean Lead Scoring: 0.358
  High coefficient of variation (cv >= 0.2) within cross validation scores.
  CatBoost Classifier w/ Imputer + SMOTENC Oversampler may not perform as_
↳estimated on unseen data.
```

```
Search finished after 00:16
```

```
Best pipeline: CatBoost Classifier w/ Imputer + SMOTENC Oversampler
```

```
Best pipeline Lead Scoring: 0.358194
```

## View rankings and select pipeline

Once the fitting process is done, we can see all of the pipelines that were searched, ranked by their score on the lead scoring objective we defined.

```
[7]: automl.rankings
```

```
[7]:
```

	id	pipeline_name	search_order	\
0	4	CatBoost Classifier w/ Imputer + SMOTENC Overs...	4	
1	2	Random Forest Classifier w/ Imputer + One Hot ...	2	
2	1	Elastic Net Classifier w/ Imputer + One Hot En...	1	
3	3	Logistic Regression Classifier w/ Imputer + On...	3	
4	0	Mode Baseline Binary Classification Pipeline	0	

  

	mean_cv_score	standard_deviation_cv_score	validation_score	\
0	0.358194	0.328811	-0.016129	
1	0.015067	0.040868	-0.016129	
2	0.013992	0.041501	-0.016129	
3	0.013992	0.041501	-0.016129	
4	0.000000	0.000000	0.000000	

  

	percent_better_than_baseline	high_variance_cv	\
0	inf	True	
1	inf	True	
2	inf	True	
3	inf	True	
4	0.0	False	

  

```

parameters
0 {'Imputer': {'categorical_impute_strategy': 'm...
1 {'Imputer': {'categorical_impute_strategy': 'm...
2 {'Imputer': {'categorical_impute_strategy': 'm...
3 {'Imputer': {'categorical_impute_strategy': 'm...
4 {'Baseline Classifier': {'strategy': 'mode'}}

```

To select the best pipeline we can call `automl.best_pipeline`.

```
[8]: best_pipeline = automl.best_pipeline
```

## Describe pipeline

You can get more details about any pipeline, including how it performed on other objective functions by calling `.describe_pipeline()` and specifying the `id` of the pipeline.

```
[9]: automl.describe_pipeline(automl.rankings.iloc[0]["id"])
```

```

*****
* CatBoost Classifier w/ Imputer + SMOTENC Oversampler *
*****

Problem Type: binary
Model Family: CatBoost

Pipeline Steps
=====
1. Imputer

```

(continues on next page)

(continued from previous page)

```

    * categorical_impute_strategy : most_frequent
    * numeric_impute_strategy : mean
    * categorical_fill_value : None
    * numeric_fill_value : None
2. SMOTENC Oversampler
    * sampling_ratio : 0.25
    * k_neighbors_default : 5
    * n_jobs : -1
    * sampling_ratio_dict : None
    * categorical_features : [0, 1, 3]
    * k_neighbors : 5
3. CatBoost Classifier
    * n_estimators : 10
    * eta : 0.03
    * max_depth : 6
    * bootstrap_type : None
    * silent : True
    * allow_writing_files : False
    * n_jobs : -1

```

Training

=====

Training for binary problems.

Objective to optimize binary classification pipeline thresholds for: <evalml.

↪objectives.lead\_scoring.LeadScoring object at 0x7fdd324eb880>

Total training time (including CV): 1.3 seconds

Cross Validation

-----

	Lead Scoring	AUC	# Training	# Validation
0	-0.016	0.869	3,099	1,550
1	0.490	0.887	3,099	1,550
2	0.600	0.889	3,100	1,549
mean	0.358	0.882	-	-
std	0.329	0.011	-	-
coef of var	0.918	0.012	-	-

### 3.2.4 Evaluate on hold out

Finally, since the best pipeline was trained on all of the training data, we evaluate it on the holdout dataset.

```

[10]: best_pipeline_score = best_pipeline.score(X_holdout, y_holdout, objectives=["auc",
↪lead_scoring_objective])
best_pipeline_score

[10]: OrderedDict([('AUC', 0.8585599879117558),
                  ('Lead Scoring', 0.8469475494411006)])

```

### 3.2.5 Why optimize for a problem-specific objective?

To demonstrate the importance of optimizing for the right objective, let's search for another pipeline using AUC, a common machine learning metric. After that, we will score the holdout data using the lead scoring objective to see how the best pipelines compare.

```
[11]: automl_auc = evalml.AutoMLSearch(X_train=X_train, y_train=y_train,
                                     problem_type='binary',
                                     objective='auc',
                                     additional_objectives=[lead_scoring_objective],
                                     allowed_model_families=["catboost", "random_forest",
                                     ↪ "linear_model"],
                                     max_batches=1)

automl_auc.search()
```

Generating pipelines to search over...  
4 pipelines ready for search.

```
*****
* Beginning pipeline search *
*****

Optimizing for AUC.
Greater score is better.

Using SequentialEngine to train and score pipelines.
Searching up to 1 batches for a total of 5 pipelines.
Allowed model families: linear_model, catboost, random_forest
```

```
FigureWidget({
  'data': [{'mode': 'lines+markers',
            'name': 'Best Score',
            'type'...}]
```

Evaluating Baseline Pipeline: Mode Baseline Binary Classification Pipeline  
Mode Baseline Binary Classification Pipeline:  
Starting cross validation  
Finished cross validation - mean AUC: 0.500

```
*****
* Evaluating Batch Number 1 *
*****
```

Elastic Net Classifier w/ Imputer + One Hot Encoder + SMOTENC Oversampler + Standard\_↪Scaler:  
Starting cross validation  
Finished cross validation - mean AUC: 0.653  
Random Forest Classifier w/ Imputer + One Hot Encoder + SMOTENC Oversampler:  
Starting cross validation  
Finished cross validation - mean AUC: 0.652  
Logistic Regression Classifier w/ Imputer + One Hot Encoder + SMOTENC Oversampler +\_↪Standard Scaler:  
Starting cross validation  
Finished cross validation - mean AUC: 0.653  
CatBoost Classifier w/ Imputer + SMOTENC Oversampler:  
Starting cross validation  
Finished cross validation - mean AUC: 0.882

(continues on next page)

(continued from previous page)

```

Search finished after 00:11
Best pipeline: CatBoost Classifier w/ Imputer + SMOTENC Oversampler
Best pipeline AUC: 0.881709

```

```
[12]: automl_auc.rankings
```

```

[12]:   id                pipeline_name  search_order  \
0    4  CatBoost Classifier w/ Imputer + SMOTENC Overs...      4
1    3  Logistic Regression Classifier w/ Imputer + On...      3
2    1  Elastic Net Classifier w/ Imputer + One Hot En...      1
3    2  Random Forest Classifier w/ Imputer + One Hot ...      2
4    0      Mode Baseline Binary Classification Pipeline      0

   mean_cv_score  standard_deviation_cv_score  validation_score  \
0         0.881709             0.010861             0.869201
1         0.653038             0.035883             0.691832
2         0.652855             0.035875             0.691582
3         0.651591             0.059622             0.717764
4         0.500000             0.000000             0.500000

   percent_better_than_baseline  high_variance_cv  \
0                38.170947             False
1                15.303770             False
2                15.285487             False
3                15.159098             False
4                 0.000000             False

                                parameters
0  {'Imputer': {'categorical_impute_strategy': 'm...
1  {'Imputer': {'categorical_impute_strategy': 'm...
2  {'Imputer': {'categorical_impute_strategy': 'm...
3  {'Imputer': {'categorical_impute_strategy': 'm...
4      {'Baseline Classifier': {'strategy': 'mode'}}

```

Like before, we can look at the rankings and pick the best pipeline.

```
[13]: best_pipeline_auc = automl_auc.best_pipeline
```

```

[14]: # get the auc and lead scoring score on holdout data
best_pipeline_auc_score = best_pipeline_auc.score(X_holdout, y_holdout, objectives=[
    ↪ "auc", lead_scoring_objective])
best_pipeline_auc_score

```

```

[14]: OrderedDict([('AUC', 0.8585599879117558),
                  ('Lead Scoring', 0.08168529664660361)])

```

```

[15]: assert best_pipeline_score['Lead Scoring'] > best_pipeline_auc_score['Lead Scoring']
assert best_pipeline_auc_score['Lead Scoring'] >= 0

```

When we optimize for AUC, we can see that the AUC score from this pipeline is similar to the AUC score from the pipeline optimized for lead scoring. However, the revenue per lead is much smaller per lead when optimized for AUC and was much larger when optimized for lead scoring. As a result, we would have a huge gain on the amount of revenue if we optimized for lead scoring.

This happens because optimizing for AUC does not take into account the user-specified `true_positive` (dollar amount to be gained with a successful lead) and `false_positive` (dollar amount to be lost with an unsuccessful

lead) values. Thus, the best pipelines may produce the highest AUC but may not actually generate the most revenue through lead scoring.

This example highlights how performance in the real world can diverge greatly from machine learning metrics.

## 3.3 Using the Cost-Benefit Matrix Objective

The Cost-Benefit Matrix (`CostBenefitMatrix`) objective is an objective that assigns costs to each of the quadrants of a confusion matrix to quantify the cost of being correct or incorrect.

### 3.3.1 Confusion Matrix

**Confusion matrices** are tables that summarize the number of correct and incorrectly-classified predictions, broken down by each class. They allow us to quickly understand the performance of a classification model and where the model gets “confused” when it is making predictions. For the binary classification problem, there are four possible combinations of prediction and actual target values possible:

- true positives (correct positive assignments)
- true negatives (correct negative assignments)
- false positives (incorrect positive assignments)
- false negatives (incorrect negative assignments)

An example of how to calculate a confusion matrix can be found [here](#).

### 3.3.2 Cost-Benefit Matrix

Although the confusion matrix is an incredibly useful visual for understanding our model, each prediction that is correctly or incorrectly classified is treated equally. For example, for detecting breast cancer, the confusion matrix does not take into consideration that it could be much more costly to incorrectly classify a malignant tumor as benign than it is to incorrectly classify a benign tumor as malignant. This is where the cost-benefit matrix shines: it uses the cost of each of the four possible outcomes to weigh each outcome differently. By scoring using the cost-benefit matrix, we can measure the score of the model by a concrete unit that is more closely related to the goal of the model. In the below example, we will show how the cost-benefit matrix objective can be used, and how it can give us better real-world impact when compared to using other standard machine learning objectives.

### 3.3.3 Customer Churn Example

#### Data

In this example, we will be using a customer churn data set taken from [Kaggle](#).

This dataset includes records of over 7000 customers, and includes customer account information, demographic information, services they signed up for, and whether or not the customer “churned” or left within the last month.

The target we want to predict is whether the customer churned (“Yes”) or did not churn (“No”). In the dataset, approximately 73.5% of customers did not churn, and 26.5% did. We will refer to the customers who churned as the “positive” class and the customers who did not churn as the “negative” class.

```
[1]: from evalml.demos.churn import load_churn
      from evalml.preprocessing import split_data

      X, y = load_churn()
      X.ww.set_types({'PaymentMethod': 'Categorical', 'Contract': 'Categorical'}) # Update_
      ↪data types Woodwork did not correctly infer
      X_train, X_holdout, y_train, y_holdout = split_data(X, y, problem_type='binary', test_
      ↪size=0.3, random_seed=0)
```

Number of Features	
Categorical	16
Numeric	3

```

Number of training examples: 7043
Targets
No      73.46%
Yes     26.54%
Name: Churn, dtype: object

```

In this example, let's say that correctly identifying customers who will churn (true positive case) will give us a net profit of \$400, because it allows us to intervene, incentivize the customer to stay, and sign a new contract. Incorrectly classifying customers who were not going to churn as customers who will churn (false positive case) will cost \$100 to represent the marketing and effort used to try to retain the user. Not identifying customers who will churn (false negative case) will cost us \$200 to represent the lost in revenue from losing a customer. Finally, correctly identifying customers who will not churn (true negative case) will not cost us anything (\$0), as nothing needs to be done for that customer.

We can represent these values in our `CostBenefitMatrix` objective, where a negative value represents a cost and a positive value represents a profit—note that this means that the greater the score, the more profit we will make.

```
[2]: from evalml.objectives import CostBenefitMatrix
      cost_benefit_matrix = CostBenefitMatrix(true_positive=400,
                                              true_negative=0,
                                              false_positive=-100,
                                              false_negative=-200)
```

## AutoML Search with Log Loss

First, let us run AutoML search to train pipelines using the default objective for binary classification (log loss).

```
[3]: from evalml import AutoMLSearch
      automl = AutoMLSearch(X_train=X_train, y_train=y_train, problem_type='binary',
      ↪objective='log loss binary',
      ↪max_iterations=5)
      automl.search()

      ll_pipeline = automl.best_pipeline
      ll_pipeline.score(X_holdout, y_holdout, ['log loss binary'])
```

```

Generating pipelines to search over...
8 pipelines ready for search.

*****
* Beginning pipeline search *
*****

```

(continues on next page)



(continued from previous page)

```

Optimizing for Log Loss Binary.
Lower score is better.

Using SequentialEngine to train and score pipelines.
Searching up to 5 pipelines.
Allowed model families: random_forest, decision_tree, xgboost, linear_model, catboost,
↳ lightgbm, extra_trees

FigureWidget({
  'data': [{'mode': 'lines+markers',
            'name': 'Best Score',
            'type': ...

Evaluating Baseline Pipeline: Mode Baseline Binary Classification Pipeline
Mode Baseline Binary Classification Pipeline:
  Starting cross validation
  Finished cross validation - mean Log Loss Binary: 9.164

*****
* Evaluating Batch Number 1 *
*****

Elastic Net Classifier w/ Imputer + One Hot Encoder + Standard Scaler:
  Starting cross validation
  Finished cross validation - mean Log Loss Binary: 0.424
Decision Tree Classifier w/ Imputer + One Hot Encoder:
  Starting cross validation
  Finished cross validation - mean Log Loss Binary: 0.628
Random Forest Classifier w/ Imputer + One Hot Encoder:
  Starting cross validation
  Finished cross validation - mean Log Loss Binary: 0.426
LightGBM Classifier w/ Imputer + One Hot Encoder:
  Starting cross validation
  Finished cross validation - mean Log Loss Binary: 0.471

Search finished after 00:11
Best pipeline: Elastic Net Classifier w/ Imputer + One Hot Encoder + Standard Scaler
Best pipeline Log Loss Binary: 0.424184

[3]: OrderedDict([('Log Loss Binary', 0.4193730950794927)])

```

When we train our pipelines using log loss as our primary objective, we try to find pipelines that minimize log loss. However, our ultimate goal in training models is to find a model that gives us the most profit, so let's score our pipeline on the cost benefit matrix (using the costs outlined above) to determine the profit we would earn from the predictions made by this model:

```

[4]: ll_pipeline_score = ll_pipeline.score(X_holdout, y_holdout, [cost_benefit_matrix])
print (ll_pipeline_score)

OrderedDict([('Cost Benefit Matrix', 53.43114055844771)])

[5]: # Calculate total profit across all customers using pipeline optimized for Log Loss
total_profit_ll = ll_pipeline_score['Cost Benefit Matrix'] * len(X)
print (total_profit_ll)

376315.5229531472

```

## AutoML Search with Cost-Benefit Matrix

Let's try rerunning our AutoML search, but this time using the cost-benefit matrix as our primary objective to optimize.

```
[6]: automl = AutoMLSearch(X_train=X_train, y_train=y_train, problem_type='binary',
    ↪ objective=cost_benefit_matrix,
    ↪ max_iterations=5)
automl.search()

cbm_pipeline = automl.best_pipeline
```

Generating pipelines to search over...  
8 pipelines ready for search.

```
*****
* Beginning pipeline search *
*****
```

Optimizing for Cost Benefit Matrix.  
Greater score is better.

Using SequentialEngine to train and score pipelines.  
Searching up to 5 pipelines.  
Allowed model families: random\_forest, decision\_tree, xgboost, linear\_model, catboost,  
↪ lightgbm, extra\_trees

```
FigureWidget({
  'data': [{'mode': 'lines+markers',
            'name': 'Best Score',
            'type': ...}]
```

Evaluating Baseline Pipeline: Mode Baseline Binary Classification Pipeline  
Mode Baseline Binary Classification Pipeline:  
Starting cross validation  
Finished cross validation - mean Cost Benefit Matrix: -53.063

```
*****
* Evaluating Batch Number 1 *
*****
```

Elastic Net Classifier w/ Imputer + One Hot Encoder + Standard Scaler:  
Starting cross validation  
Finished cross validation - mean Cost Benefit Matrix: 59.392  
Decision Tree Classifier w/ Imputer + One Hot Encoder:  
Starting cross validation  
Finished cross validation - mean Cost Benefit Matrix: 52.982  
Random Forest Classifier w/ Imputer + One Hot Encoder:  
Starting cross validation  
Finished cross validation - mean Cost Benefit Matrix: 59.048  
LightGBM Classifier w/ Imputer + One Hot Encoder:  
Starting cross validation  
Finished cross validation - mean Cost Benefit Matrix: 50.692  
High coefficient of variation (cv >= 0.2) within cross validation scores.  
LightGBM Classifier w/ Imputer + One Hot Encoder may not perform as estimated,  
↪ on unseen data.

Search finished after 00:15  
Best pipeline: Elastic Net Classifier w/ Imputer + One Hot Encoder + Standard Scaler  
Best pipeline Cost Benefit Matrix: 59.391831

Now, if we calculate the cost-benefit matrix score on our best pipeline, we see that with this pipeline optimized for our cost-benefit matrix objective, we are able to generate more profit per customer. Across our 7043 customers, we generate much more profit using this best pipeline! Custom objectives like `CostBenefitMatrix` are just one example of how using EvalML can help find pipelines that can perform better on real-world problems, rather than on arbitrary standard statistical metrics.

```
[7]: cbm_pipeline_score = cbm_pipeline.score(X_holdout, y_holdout, [cost_benefit_matrix])
      print (cbm_pipeline_score)
```

OrderedDict([('Cost Benefit Matrix', 61.57122574538572)])

```
[8]: # Calculate total profit across all customers using pipeline optimized for_
      ↪ CostBenefitMatrix
      total_profit_cbm = cbm_pipeline_score['Cost Benefit Matrix'] * len(X)
      print (total_profit_cbm)
```

433646.1429247516

```
[9]: # Calculate difference in profit made using both pipelines
      profit_diff = total_profit_cbm - total_profit_ll
      print (profit_diff)
```

57330.619971604436

Finally, we can graph the confusion matrices for both pipelines to better understand why the pipeline trained using the cost-benefit matrix is able to correctly classify more samples than the pipeline trained with log loss: we were able to correctly predict more cases where the customer would have churned (true positive), allowing us to intervene and prevent those customers from leaving.

```
[10]: from evalml.model_understanding.graphs import graph_confusion_matrix

      # pipeline trained with log loss
      y_pred = ll_pipeline.predict(X_holdout)
      graph_confusion_matrix(y_holdout, y_pred)
```

Data type cannot be displayed: application/vnd.plotly.v1+json, text/html

```
[11]: # pipeline trained with cost-benefit matrix
      y_pred = cbm_pipeline.predict(X_holdout)
      graph_confusion_matrix(y_holdout, y_pred)
```

Data type cannot be displayed: application/vnd.plotly.v1+json, text/html

## 3.4 Using Text Data with EvalML

In this demo, we will show you how to use EvalML to build models which use text data.

```
[1]: import evalml
      from evalml import AutoMLSearch
```

### 3.4.1 Dataset

We will be utilizing a dataset of SMS text messages, some of which are categorized as spam, and others which are not (“ham”). This dataset is originally from [Kaggle](#), but modified to produce a slightly more even distribution of spam to ham.

```
[2]: from urllib.request import urlopen
      import pandas as pd

input_data = urlopen('https://featurelabs-static.s3.amazonaws.com/spam_text_messages_
↳modified.csv')
data = pd.read_csv(input_data)[:750]

X = data.drop(['Category'], axis=1)
y = data['Category']

display(X.head())
```

	Message
0	Free entry in 2 a wkly comp to win FA Cup fina...
1	FreeMsg Hey there darling it's been 3 week's n...
2	WINNER!! As a valued network customer you have...
3	Had your mobile 11 months or more? U R entitle...
4	SIX chances to win CASH! From 100 to 20,000 po...

The ham vs spam distribution of the data is 3:1, so any machine learning model must get above 75% accuracy in order to perform better than a trivial baseline model which simply classifies everything as ham.

```
[3]: y.value_counts(normalize=True)
```

```
[3]: spam      0.593333
      ham       0.406667
      Name: Category, dtype: float64
```

In order to properly utilize Woodwork’s ‘Natural Language’ typing, we need to pass this argument in during initialization. Otherwise, this will be treated as an ‘Unknown’ type and dropped in the search.

```
[4]: X.ww.init(logical_types={"Message": "NaturalLanguage"})
```

### 3.4.2 Search for best pipeline

In order to validate the results of the pipeline creation and optimization process, we will save some of our data as a holdout set.

```
[5]: X_train, X_holdout, y_train, y_holdout = evalml.preprocessing.split_data(X, y,
    ↪problem_type='binary', test_size=0.2, random_seed=0)
```

EvalML uses [Woodwork](#) to automatically detect which columns are text columns, so you can run search normally, as you would if there was no text data. We can print out the logical type of the Message column and assert that it is indeed inferred as a natural language column.

```
[6]: X_train.ww
```

	Physical Type	Logical Type	Semantic Tag(s)
Column			
Message	string	NaturalLanguage	[]

Because the spam/ham labels are binary, we will use `AutoMLSearch(X_train=X_train, y_train=y_train, problem_type='binary')`. When we call `.search()`, the search for the best pipeline will begin.

```
[7]: automl = AutoMLSearch(X_train=X_train, y_train=y_train,
    ↪problem_type='binary',
    ↪max_batches=1,
    ↪optimize_thresholds=True)

automl.search()

Generating pipelines to search over...
8 pipelines ready for search.

*****
* Beginning pipeline search *
*****

Optimizing for Log Loss Binary.
Lower score is better.

Using SequentialEngine to train and score pipelines.
Searching up to 1 batches for a total of 9 pipelines.
Allowed model families: lightgbm, xgboost, catboost, random_forest, linear_model,
    ↪extra_trees, decision_tree
```

```
FigureWidget({
  'data': [{'mode': 'lines+markers',
            'name': 'Best Score',
            'type': ...}]
})
```

```
Evaluating Baseline Pipeline: Mode Baseline Binary Classification Pipeline
Mode Baseline Binary Classification Pipeline:
  Starting cross validation
  Finished cross validation - mean Log Loss Binary: 14.046
```

```
*****
* Evaluating Batch Number 1 *
*****
```

(continues on next page)

(continued from previous page)

```
Elastic Net Classifier w/ Text Featurization Component + Imputer + Standard Scaler:
    Starting cross validation
    Finished cross validation - mean Log Loss Binary: 0.350
    High coefficient of variation (cv >= 0.2) within cross validation scores.
    Elastic Net Classifier w/ Text Featurization Component + Imputer + Standard
↳Scaler may not perform as estimated on unseen data.
Decision Tree Classifier w/ Text Featurization Component + Imputer:
    Starting cross validation
    Finished cross validation - mean Log Loss Binary: 3.386
Random Forest Classifier w/ Text Featurization Component + Imputer:
    Starting cross validation
    Finished cross validation - mean Log Loss Binary: 0.221
LightGBM Classifier w/ Text Featurization Component + Imputer:
    Starting cross validation
    Finished cross validation - mean Log Loss Binary: 0.292
    High coefficient of variation (cv >= 0.2) within cross validation scores.
    LightGBM Classifier w/ Text Featurization Component + Imputer may not perform
↳as estimated on unseen data.
Logistic Regression Classifier w/ Text Featurization Component + Imputer + Standard
↳Scaler:
    Starting cross validation
    Finished cross validation - mean Log Loss Binary: 0.350
    High coefficient of variation (cv >= 0.2) within cross validation scores.
    Logistic Regression Classifier w/ Text Featurization Component + Imputer +
↳Standard Scaler may not perform as estimated on unseen data.
```

```
/home/docs/checkouts/readthedocs.org/user_builds/feature-labs-inc-evalml/envs/v0.30.2/
↳lib/python3.8/site-packages/xgboost/sklearn.py:1146: UserWarning:
```

```
The use of label encoder in XGBClassifier is deprecated and will be removed in a
↳future release. To remove this warning, do the following: 1) Pass option use_label_
↳encoder=False when constructing XGBClassifier object; and 2) Encode your labels (y)
↳as integers starting with 0, i.e. 0, 1, 2, ..., [num_class - 1].
```

```
[20:21:06] WARNING: ../src/learner.cc:1095: Starting in XGBoost 1.3.0, the default
↳evaluation metric used with the objective 'binary:logistic' was changed from 'error
↳' to 'logloss'. Explicitly set eval_metric if you'd like to restore the old
↳behavior.
```

```
/home/docs/checkouts/readthedocs.org/user_builds/feature-labs-inc-evalml/envs/v0.30.2/
↳lib/python3.8/site-packages/xgboost/sklearn.py:1146: UserWarning:
```

```
The use of label encoder in XGBClassifier is deprecated and will be removed in a
↳future release. To remove this warning, do the following: 1) Pass option use_label_
↳encoder=False when constructing XGBClassifier object; and 2) Encode your labels (y)
↳as integers starting with 0, i.e. 0, 1, 2, ..., [num_class - 1].
```

```
[20:21:07] WARNING: ../src/learner.cc:1095: Starting in XGBoost 1.3.0, the default
↳evaluation metric used with the objective 'binary:logistic' was changed from 'error
↳' to 'logloss'. Explicitly set eval_metric if you'd like to restore the old
↳behavior.
```

```
/home/docs/checkouts/readthedocs.org/user_builds/feature-labs-inc-evalml/envs/v0.30.2/
↳lib/python3.8/site-packages/xgboost/sklearn.py:1146: UserWarning:
```

```
The use of label encoder in XGBClassifier is deprecated and will be removed in a
↳future release. To remove this warning, do the following: 1) Pass option use_label_
↳encoder=False when constructing XGBClassifier object; and 2) Encode your labels (y)
↳as integers starting with 0, i.e. 0, 1, 2, ..., [num_class - 1].
```

(continues on next page)

(continued from previous page)

```
[20:21:08] WARNING: ../src/learner.cc:1095: Starting in XGBoost 1.3.0, the default
↪evaluation metric used with the objective 'binary:logistic' was changed from 'error
↪' to 'logloss'. Explicitly set eval_metric if you'd like to restore the old
↪behavior.
XGBoost Classifier w/ Text Featurization Component + Imputer:
  Starting cross validation
  Finished cross validation - mean Log Loss Binary: 0.266
  High coefficient of variation (cv >= 0.2) within cross validation scores.
  XGBoost Classifier w/ Text Featurization Component + Imputer may not perform
↪as estimated on unseen data.
Extra Trees Classifier w/ Text Featurization Component + Imputer:
  Starting cross validation
  Finished cross validation - mean Log Loss Binary: 0.292
CatBoost Classifier w/ Text Featurization Component + Imputer:
  Starting cross validation
  Finished cross validation - mean Log Loss Binary: 0.589

Search finished after 00:28
Best pipeline: Random Forest Classifier w/ Text Featurization Component + Imputer
Best pipeline Log Loss Binary: 0.221422
```

## View rankings and select pipeline

Once the fitting process is done, we can see all of the pipelines that were searched.

```
[8]: automl.rankings
```

```
[8]:   id                pipeline_name  search_order  \
0   3  Random Forest Classifier w/ Text Featurization...      3
1   6  XGBoost Classifier w/ Text Featurization Compo...      6
2   4  LightGBM Classifier w/ Text Featurization Comp...      4
3   7  Extra Trees Classifier w/ Text Featurization C...      7
4   5  Logistic Regression Classifier w/ Text Featuri...      5
5   1  Elastic Net Classifier w/ Text Featurization C...      1
6   8  CatBoost Classifier w/ Text Featurization Comp...      8
7   2  Decision Tree Classifier w/ Text Featurization...      2
8   0      Mode Baseline Binary Classification Pipeline      0

   mean_cv_score  standard_deviation_cv_score  validation_score  \
0      0.221422                0.040958      0.221587
1      0.266164                0.106501      0.242896
2      0.291768                0.114862      0.291521
3      0.292373                0.029893      0.325764
4      0.350340                0.074833      0.349271
5      0.350471                0.074886      0.349437
6      0.588944                0.004016      0.592259
7      3.385551                0.672118      3.708759
8     14.045769                0.099705     13.988204

   percent_better_than_baseline  high_variance_cv  \
0                98.423568                False
1                98.105025                 True
2                97.922737                 True
3                97.918427                False
```

(continues on next page)

(continued from previous page)

```

4          97.505728          True
5          97.504795          True
6          95.806967          False
7          75.896294          False
8           0.000000          False

                                parameters
0  {'Imputer': {'categorical_impute_strategy': 'm...
1  {'Imputer': {'categorical_impute_strategy': 'm...
2  {'Imputer': {'categorical_impute_strategy': 'm...
3  {'Imputer': {'categorical_impute_strategy': 'm...
4  {'Imputer': {'categorical_impute_strategy': 'm...
5  {'Imputer': {'categorical_impute_strategy': 'm...
6  {'Imputer': {'categorical_impute_strategy': 'm...
7  {'Imputer': {'categorical_impute_strategy': 'm...
8      {'Baseline Classifier': {'strategy': 'mode'}}

```

To select the best pipeline we can call `automl.best_pipeline`.

```
[9]: best_pipeline = automl.best_pipeline
```

## Describe pipeline

You can get more details about any pipeline, including how it performed on other objective functions.

```
[10]: automl.describe_pipeline(automl.rankings.iloc[0]["id"])

*****
* Random Forest Classifier w/ Text Featurization Component + Imputer *
*****

Problem Type: binary
Model Family: Random Forest

Pipeline Steps
=====
1. Text Featurization Component
2. Imputer
   * categorical_impute_strategy : most_frequent
   * numeric_impute_strategy : mean
   * categorical_fill_value : None
   * numeric_fill_value : None
3. Random Forest Classifier
   * n_estimators : 100
   * max_depth : 6
   * n_jobs : -1

Training
=====
Training for binary problems.
Total training time (including CV): 3.3 seconds

Cross Validation
-----
                                Log Loss Binary  MCC Binary  Gini  AUC  Precision  F1  Balanced_
→Accuracy Binary  Accuracy Binary # Training # Validation
```

(continues on next page)



(continued from previous page)

0		0.222	0.817	0.950	0.975	0.862	0.893	
↪	0.913	0.910	400		200			↪
1		0.180	0.875	0.970	0.985	0.937	0.925	↪
↪	0.936	0.940	400		200			
2		0.262	0.783	0.925	0.963	0.918	0.865	↪
↪	0.883	0.895	400		200			
mean		0.221	0.825	0.948	0.974	0.906	0.894	↪
↪	0.910	0.915	-		-			
std		0.041	0.047	0.023	0.011	0.039	0.030	↪
↪	0.026	0.023	-		-			
coef of var		0.185	0.057	0.024	0.012	0.043	0.034	↪
↪	0.029	0.025	-		-			

```
[11]: best_pipeline.graph()
```

```
[11]:
```

Notice above that there is a `Text Featurization Component` as the first step in the pipeline. `AutoMLSearch` uses the `woodwork` accessor to recognize that `'Message'` is a text column, and converts this text into numerical values that can be handled by the estimator.

### 3.4.3 Evaluate on holdout

Now, we can score the pipeline on the holdout data using the core objectives for binary classification problems.

```
[12]: scores = best_pipeline.score(X_holdout, y_holdout, objectives=evalml.objectives.get_
↪core_objectives('binary'))
print(f'Accuracy Binary: {scores["Accuracy Binary"]}')

```

```
Accuracy Binary: 0.96
```

As you can see, this model performs relatively well on this dataset, even on unseen data.

### 3.4.4 What does the Text Featurization Component do?

Machine learning models cannot handle non-numeric data. Any text must be broken down into numeric features that provide useful information about that text. The `Text Featurization` component first normalizes your text by removing any punctuation and other non-alphanumeric characters and converting any capital letters to lowercase. From there, it passes the text into `featuretools`' `nlp_primitives` `dfs` search, resulting in several informative features that replace the original column in your dataset: `Diversity Score`, `Mean Characters per Word`, `Polarity Score`, and `LSA (Latent Semantic Analysis)`.

**Diversity Score** is the ratio of unique words to total words.

**Mean Characters per Word** is the average number of letters in each word.

**Polarity Score** is a prediction of how “polarized” the text is, on a scale from -1 (extremely negative) to 1 (extremely positive).

**Latent Semantic Analysis** is an abstract representation of how important each word is with respect to the entire text, reduced down into two values per text. While the other text features are each a single column, this feature adds two columns to your data, `LSA(column_name) [0]` and `LSA(column_name) [1]`.

Let's see what this looks like with our spam/ham example.

```
[13]: best_pipeline.input_feature_names
```

```
[13]: {'Text Featurization Component': ['Message'],
      'Imputer': ['DIVERSITY_SCORE(Message)',
                  'MEAN_CHARACTERS_PER_WORD(Message)',
                  'POLARITY_SCORE(Message)',
                  'LSA(Message) [0]',
                  'LSA(Message) [1]'],
      'Random Forest Classifier': ['DIVERSITY_SCORE(Message)',
                                   'MEAN_CHARACTERS_PER_WORD(Message)',
                                   'POLARITY_SCORE(Message)',
                                   'LSA(Message) [0]',
                                   'LSA(Message) [1]']}
```

Here, the Text Featurization component takes in a single “Message” column, but then the next component in the pipeline, the Imputer, receives five columns of input. These five columns are the result of featurizing the text-type “Message” column. Most importantly, these featurized columns are what ends up passed in to the estimator.

If the dataset had any non-text columns, those would be left alone by this process. If the dataset had more than one text column, each would be broken into these five feature columns independently.

### The features, more directly

Rather than just checking the new column names, let’s examine the output of this component directly. We can see this by running the component on its own.

```
[14]: text_featurizer = evalml.pipelines.components.TextFeaturizer()
      X_featurized = text_featurizer.fit_transform(X_train)
```

Now we can compare the input data to the output from the text featurizer:

```
[15]: X_train.head()
```

```
[15]:      Message
296  Sunshine Hols. To claim ur med holiday send a ...
652      Yup ü not comin :-(
526  Hello hun how ru? Its here by the way. Im good...
571  I tagged MY friends that you seemed to count a...
472      What happened to our yo date?
```

```
[16]: X_featurized.head()
```

```
[16]:      DIVERSITY_SCORE(Message)  MEAN_CHARACTERS_PER_WORD(Message)  \
296                        1.0                        4.344828
652                        1.0                        3.000000
526                        1.0                        3.363636
571                        0.8                        4.083333
472                        1.0                        3.833333

      POLARITY_SCORE(Message)  LSA(Message) [0]  LSA(Message) [1]
296                        0.003      0.150556      -0.072443
652                        0.000      0.017340      -0.005411
526                        0.162      0.169954      0.022670
571                        0.681      0.144713      0.036799
472                        0.000      0.109373      -0.042754
```

These numeric values now represent important information about the original text that the estimator at the end of the pipeline can successfully use to make predictions.

### 3.4.5 Why encode text this way?

To demonstrate the importance of text-specific modeling, let's train a model with the same dataset, without letting AutoMLSearch detect the text column. We can change this by explicitly setting the data type of the 'Message' column in Woodwork to Categorical using the utility method `infer_feature_types`.

```
[17]: from evalml.utils import infer_feature_types
X = infer_feature_types(X, {'Message': 'Categorical'})
X_train, X_holdout, y_train, y_holdout = evalml.preprocessing.split_data(X, y,
    problem_type='binary', test_size=0.2, random_seed=0)

[18]: automl_no_text = AutoMLSearch(X_train=X_train, y_train=y_train,
    problem_type='binary',
    max_batches=1,
    optimize_thresholds=True)

automl_no_text.search()

Generating pipelines to search over...
8 pipelines ready for search.

*****
* Beginning pipeline search *
*****

Optimizing for Log Loss Binary.
Lower score is better.

Using SequentialEngine to train and score pipelines.
Searching up to 1 batches for a total of 9 pipelines.
Allowed model families: lightgbm, xgboost, catboost, random_forest, linear_model,
    extra_trees, decision_tree

FigureWidget({
  'data': [{'mode': 'lines+markers',
            'name': 'Best Score',
            'type': ...

Evaluating Baseline Pipeline: Mode Baseline Binary Classification Pipeline
Mode Baseline Binary Classification Pipeline:
  Starting cross validation
  Finished cross validation - mean Log Loss Binary: 14.046

*****
* Evaluating Batch Number 1 *
*****

Elastic Net Classifier w/ Text Featurization Component + Imputer + Standard Scaler:
  Starting cross validation
  Finished cross validation - mean Log Loss Binary: 0.350
  High coefficient of variation (cv >= 0.2) within cross validation scores.
  Elastic Net Classifier w/ Text Featurization Component + Imputer + Standard
    ↪Scaler may not perform as estimated on unseen data.
Decision Tree Classifier w/ Text Featurization Component + Imputer:
  Starting cross validation
  Finished cross validation - mean Log Loss Binary: 3.386
Random Forest Classifier w/ Text Featurization Component + Imputer:
  Starting cross validation
```

(continues on next page)

(continued from previous page)

```

    Finished cross validation - mean Log Loss Binary: 0.221
LightGBM Classifier w/ Text Featurization Component + Imputer:
    Starting cross validation
    Finished cross validation - mean Log Loss Binary: 0.292
    High coefficient of variation (cv >= 0.2) within cross validation scores.
    LightGBM Classifier w/ Text Featurization Component + Imputer may not perform
    as estimated on unseen data.
Logistic Regression Classifier w/ Text Featurization Component + Imputer + Standard
    Scaler:
    Starting cross validation
    Finished cross validation - mean Log Loss Binary: 0.350
    High coefficient of variation (cv >= 0.2) within cross validation scores.
    Logistic Regression Classifier w/ Text Featurization Component + Imputer +
    Standard Scaler may not perform as estimated on unseen data.

/home/docs/checkouts/readthedocs.org/user_builds/feature-labs-inc-evalml/envs/v0.30.2/
lib/python3.8/site-packages/xgboost/sklearn.py:1146: UserWarning:

The use of label encoder in XGBClassifier is deprecated and will be removed in a
future release. To remove this warning, do the following: 1) Pass option use_label_
encoder=False when constructing XGBClassifier object; and 2) Encode your labels (y)
as integers starting with 0, i.e. 0, 1, 2, ..., [num_class - 1].

[20:21:33] WARNING: ../src/learner.cc:1095: Starting in XGBoost 1.3.0, the default
evaluation metric used with the objective 'binary:logistic' was changed from 'error
' to 'logloss'. Explicitly set eval_metric if you'd like to restore the old
behavior.

/home/docs/checkouts/readthedocs.org/user_builds/feature-labs-inc-evalml/envs/v0.30.2/
lib/python3.8/site-packages/xgboost/sklearn.py:1146: UserWarning:

The use of label encoder in XGBClassifier is deprecated and will be removed in a
future release. To remove this warning, do the following: 1) Pass option use_label_
encoder=False when constructing XGBClassifier object; and 2) Encode your labels (y)
as integers starting with 0, i.e. 0, 1, 2, ..., [num_class - 1].

[20:21:34] WARNING: ../src/learner.cc:1095: Starting in XGBoost 1.3.0, the default
evaluation metric used with the objective 'binary:logistic' was changed from 'error
' to 'logloss'. Explicitly set eval_metric if you'd like to restore the old
behavior.

/home/docs/checkouts/readthedocs.org/user_builds/feature-labs-inc-evalml/envs/v0.30.2/
lib/python3.8/site-packages/xgboost/sklearn.py:1146: UserWarning:

The use of label encoder in XGBClassifier is deprecated and will be removed in a
future release. To remove this warning, do the following: 1) Pass option use_label_
encoder=False when constructing XGBClassifier object; and 2) Encode your labels (y)
as integers starting with 0, i.e. 0, 1, 2, ..., [num_class - 1].

[20:21:35] WARNING: ../src/learner.cc:1095: Starting in XGBoost 1.3.0, the default
evaluation metric used with the objective 'binary:logistic' was changed from 'error
' to 'logloss'. Explicitly set eval_metric if you'd like to restore the old
behavior.
XGBoost Classifier w/ Text Featurization Component + Imputer:
    Starting cross validation
    Finished cross validation - mean Log Loss Binary: 0.266
    High coefficient of variation (cv >= 0.2) within cross validation scores.

```

(continues on next page)

(continued from previous page)

```

XGBoost Classifier w/ Text Featurization Component + Imputer may not perform
↳ as estimated on unseen data.
Extra Trees Classifier w/ Text Featurization Component + Imputer:
  Starting cross validation
  Finished cross validation - mean Log Loss Binary: 0.292
CatBoost Classifier w/ Text Featurization Component + Imputer:
  Starting cross validation
  Finished cross validation - mean Log Loss Binary: 0.589

Search finished after 00:25
Best pipeline: Random Forest Classifier w/ Text Featurization Component + Imputer
Best pipeline Log Loss Binary: 0.221422

```

Like before, we can look at the rankings and pick the best pipeline.

```
[19]: automl_no_text.rankings
```

```

[19]:   id                pipeline_name  search_order  \
0    3  Random Forest Classifier w/ Text Featurization...    3
1    6  XGBoost Classifier w/ Text Featurization Compo...    6
2    4  LightGBM Classifier w/ Text Featurization Comp...    4
3    7  Extra Trees Classifier w/ Text Featurization C...    7
4    5  Logistic Regression Classifier w/ Text Featuri...    5
5    1  Elastic Net Classifier w/ Text Featurization C...    1
6    8  CatBoost Classifier w/ Text Featurization Comp...    8
7    2  Decision Tree Classifier w/ Text Featurization...    2
8    0      Mode Baseline Binary Classification Pipeline    0

   mean_cv_score  standard_deviation_cv_score  validation_score  \
0         0.221422             0.040958         0.221587
1         0.266164             0.106501         0.242896
2         0.291768             0.114862         0.291521
3         0.292373             0.029893         0.325764
4         0.350340             0.074833         0.349271
5         0.350471             0.074886         0.349437
6         0.588944             0.004016         0.592259
7         3.385551             0.672118         3.708759
8        14.045769             0.099705        13.988204

   percent_better_than_baseline  high_variance_cv  \
0                98.423568                False
1                98.105025                 True
2                97.922737                 True
3                97.918427                False
4                97.505728                 True
5                97.504795                 True
6                95.806967                False
7                75.896294                False
8                 0.000000                False

                        parameters
0  {'Imputer': {'categorical_impute_strategy': 'm...
1  {'Imputer': {'categorical_impute_strategy': 'm...
2  {'Imputer': {'categorical_impute_strategy': 'm...
3  {'Imputer': {'categorical_impute_strategy': 'm...
4  {'Imputer': {'categorical_impute_strategy': 'm...
5  {'Imputer': {'categorical_impute_strategy': 'm...

```

(continues on next page)

(continued from previous page)

```

6 {'Imputer': {'categorical_impute_strategy': 'm...
7 {'Imputer': {'categorical_impute_strategy': 'm...
8     {'Baseline Classifier': {'strategy': 'mode'}}

```

```
[20]: best_pipeline_no_text = automl_no_text.best_pipeline
```

Here, changing the data type of the text column removed the Text Featurization Component from the pipeline.

```
[21]: best_pipeline_no_text.graph()
```

```
[21]:
```

```
[22]: automl_no_text.describe_pipeline(automl_no_text.rankings.iloc[0]["id"])
```

```

*****
* Random Forest Classifier w/ Text Featurization Component + Imputer *
*****

Problem Type: binary
Model Family: Random Forest

Pipeline Steps
=====
1. Text Featurization Component
2. Imputer
    * categorical_impute_strategy : most_frequent
    * numeric_impute_strategy : mean
    * categorical_fill_value : None
    * numeric_fill_value : None
3. Random Forest Classifier
    * n_estimators : 100
    * max_depth : 6
    * n_jobs : -1

Training
=====
Training for binary problems.
Total training time (including CV): 3.3 seconds

Cross Validation
-----

```

	Log Loss Binary	MCC Binary	Gini	AUC	Precision	F1	Balanced_
→ Accuracy Binary	Accuracy Binary	# Training	# Validation				
0	0.222	0.817	0.950	0.975	0.862	0.893	
→ 0.913	0.910	400		200			
1	0.180	0.875	0.970	0.985	0.937	0.925	
→ 0.936	0.940	400		200			
2	0.262	0.783	0.925	0.963	0.918	0.865	
→ 0.883	0.895	400		200			
mean	0.221	0.825	0.948	0.974	0.906	0.894	
→ 0.910	0.915	-		-			
std	0.041	0.047	0.023	0.011	0.039	0.030	
→ 0.026	0.023	-		-			
coef of var	0.185	0.057	0.024	0.012	0.043	0.034	
→ 0.029	0.025	-		-			

```
[23]: # get standard performance metrics on holdout data
scores = best_pipeline_no_text.score(X_holdout, y_holdout, objectives=evalml.
↳objectives.get_core_objectives('binary'))
print(f'Accuracy Binary: {scores["Accuracy Binary"]}')

```

```
Accuracy Binary: 0.96

```

Without the Text Featurization Component, the 'Message' column was treated as a categorical column, and therefore the conversion of this text to numerical features happened in the One Hot Encoder. The best pipeline encoded the top 10 most frequent “categories” of these texts, meaning 10 text messages were one-hot encoded and all the others were dropped. Clearly, this removed almost all of the information from the dataset, as we can see the `best_pipeline_no_text` performs very similarly to randomly guessing “ham” in every case.





These guides include in-depth descriptions and explanations of EvalML's features.

## 4.1 Automated Machine Learning (AutoML) Search

### 4.1.1 Background

#### Machine Learning

**Machine learning** (ML) is the process of constructing a mathematical model of a system based on a sample dataset collected from that system.

One of the main goals of training an ML model is to teach the model to separate the signal present in the data from the noise inherent in system and in the data collection process. If this is done effectively, the model can then be used to make accurate predictions about the system when presented with new, similar data. Additionally, introspecting on an ML model can reveal key information about the system being modeled, such as which inputs and transformations of the inputs are most useful to the ML model for learning the signal in the data, and are therefore the most predictive.

There are a **variety** of ML problem types. Supervised learning describes the case where the collected data contains an output value to be modeled and a set of inputs with which to train the model. EvalML focuses on training supervised learning models.

EvalML supports three common supervised ML problem types. The first is regression, where the target value to model is a continuous numeric value. Next are binary and multiclass classification, where the target value to model consists of two or more discrete values or categories. The choice of which supervised ML problem type is most appropriate depends on domain expertise and on how the model will be evaluated and used.

EvalML is currently building support for supervised time series problems: time series regression, time series binary classification, and time series multiclass classification. While we've added some features to tackle these kinds of problems, our functionality is still being actively developed so please be mindful of that before using it.

#### AutoML and Search

**AutoML** is the process of automating the construction, training and evaluation of ML models. Given a data and some configuration, AutoML searches for the most effective and accurate ML model or models to fit the dataset. During the search, AutoML will explore different combinations of model type, model parameters and model architecture.

An effective AutoML solution offers several advantages over constructing and tuning ML models by hand. AutoML can assist with many of the difficult aspects of ML, such as avoiding overfitting and underfitting, imbalanced data, detecting data leakage and other potential issues with the problem setup, and automatically applying best-practice data cleaning, feature engineering, feature selection and various modeling techniques. AutoML can also leverage

search algorithms to optimally sweep the hyperparameter search space, resulting in model performance which would be difficult to achieve by manual training.

### 4.1.2 AutoML in EvalML

EvalML supports all of the above and more.

In its simplest usage, the AutoML search interface requires only the input data, the target data and a `problem_type` specifying what kind of supervised ML problem to model.

\*\* Graphing methods, like AutoMLSearch, on Jupyter Notebook and Jupyter Lab require `ipywidgets` to be installed.

\*\* If graphing on Jupyter Lab, `jupyterlab-plotly` required. To download this, make sure you have `npm` installed.

```
[1]: import evalml
from evalml.utils import infer_feature_types
X, y = evalml.demos.load_fraud(n_rows=250)
```

```

      Number of Features
Boolean                      1
Categorical                  6
Numeric                      5

Number of training examples: 250
Targets
False      88.40%
True       11.60%
Name: fraud, dtype: object
```

To provide data to EvalML, it is recommended that you initialize a `Woodwork` accessor on your data. This allows you to easily control how EvalML will treat each of your features before training a model.

EvalML also accepts `pandas` input, and will run type inference on top of the input `pandas` data. If you'd like to change the types inferred by EvalML, you can use the `infer_feature_types` utility method, which takes `pandas` or `numpy` input and converts it to a `Woodwork` data structure. The `feature_types` parameter can be used to specify what types specific columns should be.

Feature types such as `Natural Language` must be specified in this way, otherwise `Woodwork` will infer it as `Unknown` type and drop it during the `AutoMLSearch`.

In the example below, we reformat a couple features to make them easily consumable by the model, and then specify that the provider, which would have otherwise been inferred as a column with natural language, is a categorical column.

```
[2]: X.ww['expiration_date'] = X['expiration_date'].apply(lambda x: '20{}-01-{}'.format(x.
↳split("/") [1], x.split("/") [0]))
X = infer_feature_types(X, feature_types= {'store_id': 'categorical',
                                          'expiration_date': 'datetime',
                                          'lat': 'categorical',
                                          'lng': 'categorical',
                                          'provider': 'categorical'})
```

In order to validate the results of the pipeline creation and optimization process, we will save some of our data as a holdout set.

```
[3]: X_train, X_holdout, y_train, y_holdout = evalml.preprocessing.split_data(X, y,
↳problem_type='binary', test_size=.2)
```

## Data Checks

Before calling `AutoMLSearch.search`, we should run some sanity checks on our data to ensure that the input data being passed will not run into some common issues before running a potentially time-consuming search. EvalML has various data checks that makes this easy. Each data check will return a collection of warnings and errors if it detects potential issues with the input data. This allows users to inspect their data to avoid confusing errors that may arise during the search process. You can learn about each of the data checks available through our [data checks guide](#)

Here, we will run the `DefaultDataChecks` class, which contains a series of data checks that are generally useful.

```
[4]: from evalml.data_checks import DefaultDataChecks

data_checks = DefaultDataChecks("binary", "log loss binary")
data_checks.validate(X_train, y_train)

[4]: {'warnings': [], 'errors': [], 'actions': []}
```

Since there were no warnings or errors returned, we can safely continue with the search process.

```
[5]: automl = evalml.automl.AutoMLSearch(X_train=X_train, y_train=y_train, problem_type=
    ↪ 'binary')
automl.search()

Using default limit of max_batches=1.

Generating pipelines to search over...
8 pipelines ready for search.

*****
* Beginning pipeline search *
*****

Optimizing for Log Loss Binary.
Lower score is better.

Using SequentialEngine to train and score pipelines.
Searching up to 1 batches for a total of 9 pipelines.
Allowed model families: linear_model, random_forest, xgboost, decision_tree, lightgbm,
    ↪ extra_trees, catboost

FigureWidget({
  'data': [{'mode': 'lines+markers',
            'name': 'Best Score',
            'type': ...

Evaluating Baseline Pipeline: Mode Baseline Binary Classification Pipeline
Mode Baseline Binary Classification Pipeline:
  Starting cross validation
  Finished cross validation - mean Log Loss Binary: 3.970

*****
* Evaluating Batch Number 1 *
*****

Elastic Net Classifier w/ Imputer + DateTime Featurization Component + One Hot_
    ↪ Encoder + SMOTENC Oversampler + Standard Scaler:
  Starting cross validation
  Finished cross validation - mean Log Loss Binary: 0.512
Decision Tree Classifier w/ Imputer + DateTime Featurization Component + One Hot_
    ↪ Encoder + SMOTENC Oversampler: (continues on next page)
```

(continued from previous page)

```

Starting cross validation
Finished cross validation - mean Log Loss Binary: 2.957
High coefficient of variation (cv >= 0.2) within cross validation scores.
Decision Tree Classifier w/ Imputer + DateTime Featurization Component + One_
↳Hot Encoder + SMOTENC Oversampler may not perform as estimated on unseen data.
Random Forest Classifier w/ Imputer + DateTime Featurization Component + One Hot_
↳Encoder + SMOTENC Oversampler:
Starting cross validation
Finished cross validation - mean Log Loss Binary: 0.286
LightGBM Classifier w/ Imputer + DateTime Featurization Component + One Hot Encoder +_
↳SMOTENC Oversampler:
Starting cross validation
Finished cross validation - mean Log Loss Binary: 0.309
High coefficient of variation (cv >= 0.2) within cross validation scores.
LightGBM Classifier w/ Imputer + DateTime Featurization Component + One Hot_
↳Encoder + SMOTENC Oversampler may not perform as estimated on unseen data.
Logistic Regression Classifier w/ Imputer + DateTime Featurization Component + One_
↳Hot Encoder + SMOTENC Oversampler + Standard Scaler:
Starting cross validation
Finished cross validation - mean Log Loss Binary: 0.552

```

```

/home/docs/checkouts/readthedocs.org/user_builds/feature-labs-inc-evalml/envs/v0.30.2/
↳lib/python3.8/site-packages/xgboost/sklearn.py:1146: UserWarning:

```

```

The use of label encoder in XGBClassifier is deprecated and will be removed in a_
↳future release. To remove this warning, do the following: 1) Pass option use_label_
↳encoder=False when constructing XGBClassifier object; and 2) Encode your labels (y)_
↳as integers starting with 0, i.e. 0, 1, 2, ..., [num_class - 1].

```

```

[20:22:45] WARNING: ../src/learner.cc:1095: Starting in XGBoost 1.3.0, the default_
↳evaluation metric used with the objective 'binary:logistic' was changed from 'error
↳' to 'logloss'. Explicitly set eval_metric if you'd like to restore the old_
↳behavior.

```

```

/home/docs/checkouts/readthedocs.org/user_builds/feature-labs-inc-evalml/envs/v0.30.2/
↳lib/python3.8/site-packages/xgboost/sklearn.py:1146: UserWarning:

```

```

The use of label encoder in XGBClassifier is deprecated and will be removed in a_
↳future release. To remove this warning, do the following: 1) Pass option use_label_
↳encoder=False when constructing XGBClassifier object; and 2) Encode your labels (y)_
↳as integers starting with 0, i.e. 0, 1, 2, ..., [num_class - 1].

```

```

[20:22:46] WARNING: ../src/learner.cc:1095: Starting in XGBoost 1.3.0, the default_
↳evaluation metric used with the objective 'binary:logistic' was changed from 'error
↳' to 'logloss'. Explicitly set eval_metric if you'd like to restore the old_
↳behavior.

```

```

/home/docs/checkouts/readthedocs.org/user_builds/feature-labs-inc-evalml/envs/v0.30.2/
↳lib/python3.8/site-packages/xgboost/sklearn.py:1146: UserWarning:

```

```

The use of label encoder in XGBClassifier is deprecated and will be removed in a_
↳future release. To remove this warning, do the following: 1) Pass option use_label_
↳encoder=False when constructing XGBClassifier object; and 2) Encode your labels (y)_
↳as integers starting with 0, i.e. 0, 1, 2, ..., [num_class - 1].

```

```

[20:22:47] WARNING: ../src/learner.cc:1095: Starting in XGBoost 1.3.0, the default_
↳evaluation metric used with the objective 'binary:logistic' was changed from 'error
↳' to 'logloss'. Explicitly set eval_metric if you'd like to restore the old_
↳behavior.

```

(continues on next page)

(continued from previous page)

```

XGBoost Classifier w/ Imputer + DateTime Featurization Component + One Hot Encoder +
↳ SMOTENC Oversampler:
    Starting cross validation
    Finished cross validation - mean Log Loss Binary: 0.279
    High coefficient of variation (cv >= 0.2) within cross validation scores.
    XGBoost Classifier w/ Imputer + DateTime Featurization Component + One Hot
↳ Encoder + SMOTENC Oversampler may not perform as estimated on unseen data.
Extra Trees Classifier w/ Imputer + DateTime Featurization Component + One Hot
↳ Encoder + SMOTENC Oversampler:
    Starting cross validation
    Finished cross validation - mean Log Loss Binary: 0.338
CatBoost Classifier w/ Imputer + DateTime Featurization Component + SMOTENC
↳ Oversampler:
    Starting cross validation
    Finished cross validation - mean Log Loss Binary: 0.602

Search finished after 00:22

/home/docs/checkouts/readthedocs.org/user_builds/feature-labs-inc-evalml/envs/v0.30.2/
↳ lib/python3.8/site-packages/xgboost/sklearn.py:1146: UserWarning:

The use of label encoder in XGBClassifier is deprecated and will be removed in a
↳ future release. To remove this warning, do the following: 1) Pass option use_label_
↳ encoder=False when constructing XGBClassifier object; and 2) Encode your labels (y)
↳ as integers starting with 0, i.e. 0, 1, 2, ..., [num_class - 1].

[20:22:52] WARNING: ../src/learner.cc:1095: Starting in XGBoost 1.3.0, the default
↳ evaluation metric used with the objective 'binary:logistic' was changed from 'error
↳ ' to 'logloss'. Explicitly set eval_metric if you'd like to restore the old
↳ behavior.
Best pipeline: XGBoost Classifier w/ Imputer + DateTime Featurization Component + One
↳ Hot Encoder + SMOTENC Oversampler
Best pipeline Log Loss Binary: 0.278707

```

The AutoML search will log its progress, reporting each pipeline and parameter set evaluated during the search.

There are a number of mechanisms to control the AutoML search time. One way is to set the `max_batches` parameter which controls the maximum number of rounds of AutoML to evaluate, where each round may train and score a variable number of pipelines. Another way is to set the `max_iterations` parameter which controls the maximum number of candidate models to be evaluated during AutoML. By default, AutoML will search for a single batch. The first pipeline to be evaluated will always be a baseline model representing a trivial solution.

The AutoML interface supports a variety of other parameters. For a comprehensive list, please [refer to the API reference](#).

We also provide [a standalone search method](#) which does all of the above in a single line, and returns the `AutoMLSearch` instance and data check results. If there were data check errors, AutoML will not be run and no `AutoMLSearch` instance will be returned.

## Detecting Problem Type

EvalML includes a simple method, `detect_problem_type`, to help determine the problem type given the target data.

This function can return the predicted problem type as a `ProblemType` enum, choosing from `ProblemType.BINARY`, `ProblemType.MULTICLASS`, and `ProblemType.REGRESSION`. If the target data is invalid (for instance when there is only 1 unique label), the function will throw an error instead.

```
[6]: import pandas as pd
      from evalml.problem_types import detect_problem_type

      y_binary = pd.Series([0, 1, 1, 0, 1, 1])
      detect_problem_type(y_binary)

[6]: <ProblemTypes.BINARY: 'binary'>
```

## Objective parameter

`AutoMLSearch` takes in an objective parameter to determine which objective to optimize for. By default, this parameter is set to `auto`, which allows AutoML to choose `LogLossBinary` for binary classification problems, `LogLossMulticlass` for multiclass classification problems, and `R2` for regression problems.

It should be noted that the objective parameter is only used in ranking and helping choose the pipelines to iterate over, but is not used to optimize each individual pipeline during fit-time.

To get the default objective for each problem type, you can use the `get_default_primary_search_objective` function.

```
[7]: from evalml.automl import get_default_primary_search_objective

      binary_objective = get_default_primary_search_objective("binary")
      multiclass_objective = get_default_primary_search_objective("multiclass")
      regression_objective = get_default_primary_search_objective("regression")

      print(binary_objective.name)
      print(multiclass_objective.name)
      print(regression_objective.name)

      Log Loss Binary
      Log Loss Multiclass
      R2
```

## Using custom pipelines

EvalML's AutoML algorithm generates a set of pipelines to search with. To provide a custom set instead, set `allowed_component_graphs` to a dictionary of custom component graphs. `AutoMLSearch` will use these to generate Pipeline instances. Note: this will prevent AutoML from generating other pipelines to search over.

```
[8]: from evalml.pipelines import MulticlassClassificationPipeline

      automl_custom = evalml.automl.AutoMLSearch(X_train=X_train,
                                                  y_train=y_train,
                                                  problem_type='multiclass',
```

(continues on next page)

(continued from previous page)

```

allowed_component_graphs={"My_pipeline": [
↳ 'Simple Imputer', 'Random Forest Classifier'],
                               "My_other_
↳ pipeline": ['One Hot Encoder', 'Random Forest Classifier']})

Using default limit of max_batches=1.

2 pipelines ready for search.

/home/docs/checkouts/readthedocs.org/user_builds/feature-labs-inc-evalml/envs/v0.30.2/
↳ lib/python3.8/site-packages/evalml/automl/automl_search.py:676:
↳ ParameterNotUsedWarning:

Parameters for components {'SMOTENC Oversampler'} will not be used to instantiate the
↳ pipeline since they don't appear in the pipeline

```

## Stopping the search early

To stop the search early, hit `Ctrl-C`. This will bring up a prompt asking for confirmation. Responding with `y` will immediately stop the search. Responding with `n` will continue the search.

## Callback functions

`AutoMLSearch` supports several callback functions, which can be specified as parameters when initializing an `AutoMLSearch` object. They are:

- `start_iteration_callback`
- `add_result_callback`
- `error_callback`

## Start Iteration Callback

Users can set `start_iteration_callback` to set what function is called before each pipeline training iteration. This callback function must take three positional parameters: the pipeline class, the pipeline parameters, and the `AutoMLSearch` object.

```

[9]: ## start_iteration_callback example function
def start_iteration_callback_example(pipeline_class, pipeline_params, automl_obj):
    print ("Training pipeline with the following parameters:", pipeline_params)

```

## Add Result Callback

Users can set `add_result_callback` to set what function is called after each pipeline training iteration. This callback function must take three positional parameters: a dictionary containing the training results for the new pipeline, an `untrained_pipeline` containing the parameters used during training, and the `AutoMLSearch` object.

```
[10]: ## add_result_callback example function
def add_result_callback_example(pipeline_results_dict, untrained_pipeline, automl_
    obj):
    print ("Results for trained pipeline with the following parameters:", pipeline_
    results_dict)
```

## Error Callback

Users can set the `error_callback` to set what function called when `search()` errors and raises an `Exception`. This callback function takes three positional parameters: the `Exception` raised, the `traceback`, and the `AutoMLSearch` object. This callback function must also accept `kwargs`, so `AutoMLSearch` is able to pass along other parameters used by default.

Evalml defines several error callback functions, which can be found under `evalml.automl.callbacks`. They are:

- `silent_error_callback`
- `raise_error_callback`
- `log_and_save_error_callback`
- `raise_and_save_error_callback`
- `log_error_callback` (default used when `error_callback` is `None`)

```
[11]: # error_callback example; this is implemented in the evalml library
def raise_error_callback(exception, traceback, automl, **kwargs):
    """Raises the exception thrown by the AutoMLSearch object. Also logs the_
    exception as an error."""
    logger.error(f'AutoMLSearch raised a fatal exception: {str(exception)}')
    logger.error("\n".join(traceback))
    raise exception
```

## 4.1.3 View Rankings

A summary of all the pipelines built can be returned as a pandas `DataFrame` which is sorted by score. The `score` column contains the average score across all cross-validation folds while the `validation_score` column is computed from the first cross-validation fold.

```
[12]: automl.rankings
```

```
[12]:   id  pipeline_name  search_order  \
0   6  XGBoost Classifier w/ Imputer + DateTime Featu...    6
1   3  Random Forest Classifier w/ Imputer + DateTime...    3
2   4  LightGBM Classifier w/ Imputer + DateTime Feat...    4
3   7  Extra Trees Classifier w/ Imputer + DateTime F...    7
4   1  Elastic Net Classifier w/ Imputer + DateTime F...    1
5   5  Logistic Regression Classifier w/ Imputer + Da...    5
6   8  CatBoost Classifier w/ Imputer + DateTime Feat...    8
```

(continues on next page)



(continued from previous page)

7	2	Decision Tree Classifier w/ Imputer + DateTime...	2
8	0	Mode Baseline Binary Classification Pipeline	0
	mean_cv_score	standard_deviation_cv_score	validation_score \
0	0.278707	0.190725	0.202638
1	0.285613	0.041116	0.263695
2	0.308636	0.203878	0.234947
3	0.338286	0.009381	0.329015
4	0.511808	0.074992	0.590517
5	0.551960	0.082695	0.628646
6	0.601819	0.007246	0.593693
7	2.956956	3.084439	0.651557
8	3.970423	0.266060	4.124033
	percent_better_than_baseline	high_variance_cv \	
0	92.980409	True	
1	92.806475	False	
2	92.226623	True	
3	91.479857	False	
4	87.109474	False	
5	86.098206	False	
6	84.842444	False	
7	25.525413	True	
8	0.000000	False	
	parameters		
0	{ 'Imputer': { 'categorical_impute_strategy': 'm...		
1	{ 'Imputer': { 'categorical_impute_strategy': 'm...		
2	{ 'Imputer': { 'categorical_impute_strategy': 'm...		
3	{ 'Imputer': { 'categorical_impute_strategy': 'm...		
4	{ 'Imputer': { 'categorical_impute_strategy': 'm...		
5	{ 'Imputer': { 'categorical_impute_strategy': 'm...		
6	{ 'Imputer': { 'categorical_impute_strategy': 'm...		
7	{ 'Imputer': { 'categorical_impute_strategy': 'm...		
8	{ 'Baseline Classifier': { 'strategy': 'mode' }}		

#### 4.1.4 Describe Pipeline

Each pipeline is given an `id`. We can get more information about any particular pipeline using that `id`. Here, we will get more information about the pipeline with `id = 1`.

```
[13]: automl.describe_pipeline(1)
```

```
*****
* Elastic Net Classifier w/ Imputer + DateTime Featurization Component + One Hot_
↳ Encoder + SMOTENC Oversampler + Standard Scaler *
*****

Problem Type: binary
Model Family: Linear

Pipeline Steps
=====
1. Imputer
```

(continues on next page)

(continued from previous page)

```

    * categorical_impute_strategy : most_frequent
    * numeric_impute_strategy : mean
    * categorical_fill_value : None
    * numeric_fill_value : None
2. DateTime Featurization Component
    * features_to_extract : ['year', 'month', 'day_of_week', 'hour']
    * encode_as_categories : False
    * date_index : None
3. One Hot Encoder
    * top_n : 10
    * features_to_encode : None
    * categories : None
    * drop : if_binary
    * handle_unknown : ignore
    * handle_missing : error
4. SMOTENC Oversampler
    * sampling_ratio : 0.25
    * k_neighbors_default : 5
    * n_jobs : -1
    * sampling_ratio_dict : None
    * categorical_features : [3, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21,
→22, 23, 24, 25, 26, 27, 28, 29, 30, 31, 32, 33, 34, 35, 36, 37, 38, 39, 40, 41, 42,
→43, 44, 45, 46, 47, 48, 49, 50, 51, 52, 53, 54, 55, 56, 57, 58, 59]
    * k_neighbors : 5
5. Standard Scaler
6. Elastic Net Classifier
    * penalty : elasticnet
    * C : 1.0
    * l1_ratio : 0.15
    * n_jobs : -1
    * multi_class : auto
    * solver : saga

```

#### Training

```
=====
```

Training for binary problems.

Total training time (including CV): 2.8 seconds

#### Cross Validation

```
-----
```

	Log Loss Binary	MCC Binary	Gini	AUC	Precision	F1	Balanced
→Accuracy Binary	Accuracy Binary	# Training	# Validation				
0	0.591	0.175	0.195	0.597	0.286	0.267	
→ 0.583	0.836	133	67				
1	0.441	0.296	0.419	0.710	0.500	0.333	
→ 0.608	0.881	133	67				
2	0.504	0.035	0.259	0.630	0.120	0.188	
→ 0.528	0.606	134	66				
mean	0.512	0.169	0.291	0.646	0.302	0.263	
→ 0.573	0.774	-	-				
std	0.075	0.130	0.116	0.058	0.191	0.073	
→ 0.041	0.147	-	-				
coef of var	0.147	0.772	0.397	0.090	0.631	0.278	
→ 0.072	0.190	-	-				

## 4.1.5 Get Pipeline

We can get the object of any pipeline via their id as well:

```
[14]: pipeline = automl.get_pipeline(1)
      print(pipeline.name)
      print(pipeline.parameters)

Elastic Net Classifier w/ Imputer + DateTime Featurization Component + One Hot
↳Encoder + SMOTENC Oversampler + Standard Scaler
{'Imputer': {'categorical_impute_strategy': 'most_frequent', 'numeric_impute_strategy': 'mean', 'categorical_fill_value': None, 'numeric_fill_value': None}, 'DateTimeFeaturization Component': {'features_to_extract': ['year', 'month', 'day_of_week', 'hour'], 'encode_as_categories': False, 'date_index': None}, 'One Hot Encoder': {'top_n': 10, 'features_to_encode': None, 'categories': None, 'drop': 'if_binary', 'handle_unknown': 'ignore', 'handle_missing': 'error'}, 'SMOTENC Oversampler': {'sampling_ratio': 0.25, 'k_neighbors_default': 5, 'n_jobs': -1, 'sampling_ratio_dict': None, 'categorical_features': [3, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31, 32, 33, 34, 35, 36, 37, 38, 39, 40, 41, 42, 43, 44, 45, 46, 47, 48, 49, 50, 51, 52, 53, 54, 55, 56, 57, 58, 59], 'k_neighbors': 5}, 'Elastic Net Classifier': {'penalty': 'elasticnet', 'C': 1.0, 'l1_ratio': 0.15, 'n_jobs': -1, 'multi_class': 'auto', 'solver': 'saga'}}
```

### Get best pipeline

If you specifically want to get the best pipeline, there is a convenient accessor for that. The pipeline returned is already fitted on the input X, y data that we passed to AutoMLSearch. To turn off this default behavior, set `train_best_pipeline=False` when initializing AutoMLSearch.

```
[15]: best_pipeline = automl.best_pipeline
      print(best_pipeline.name)
      print(best_pipeline.parameters)
      best_pipeline.predict(X_train)

XGBoost Classifier w/ Imputer + DateTime Featurization Component + One Hot Encoder +
↳SMOTENC Oversampler
{'Imputer': {'categorical_impute_strategy': 'most_frequent', 'numeric_impute_strategy': 'mean', 'categorical_fill_value': None, 'numeric_fill_value': None}, 'DateTimeFeaturization Component': {'features_to_extract': ['year', 'month', 'day_of_week', 'hour'], 'encode_as_categories': False, 'date_index': None}, 'One Hot Encoder': {'top_n': 10, 'features_to_encode': None, 'categories': None, 'drop': 'if_binary', 'handle_unknown': 'ignore', 'handle_missing': 'error'}, 'SMOTENC Oversampler': {'sampling_ratio': 0.25, 'k_neighbors_default': 5, 'n_jobs': -1, 'sampling_ratio_dict': None, 'categorical_features': [3, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31, 32, 33, 34, 35, 36, 37, 38, 39, 40, 41, 42, 43, 44, 45, 46, 47, 48, 49, 50, 51, 52, 53, 54, 55, 56, 57, 58, 59], 'k_neighbors': 5}, 'XGBoost Classifier': {'eta': 0.1, 'max_depth': 6, 'min_child_weight': 1, 'n_estimators': 100, 'n_jobs': -1}}

[15]: 0      False
      1      False
      2      False
      3       True
      4      False
      ...
     195     False
     196     False
```

(continues on next page)

(continued from previous page)

```

197     False
198     False
199     False
Name: fraud, Length: 200, dtype: bool

```

## 4.1.6 Training and Scoring Multiple Pipelines using AutoMLSearch

AutoMLSearch will automatically fit the best pipeline on the entire training data. It also provides an easy API for training and scoring other pipelines.

If you'd like to train one or more pipelines on the entire training data, you can use the `train_pipelines` method

Similarly, if you'd like to score one or more pipelines on a particular dataset, you can use the `train_pipelines` method

```

[16]: trained_pipelines = automl.train_pipelines([automl.get_pipeline(i) for i in [0, 1, 2]])
trained_pipelines

```

```

[16]: {'Mode Baseline Binary Classification Pipeline': pipeline =
  ↳ BinaryClassificationPipeline(component_graph={'Baseline Classifier': ['Baseline_
  ↳ Classifier', 'X', 'y']}, parameters={'Baseline Classifier':{'strategy': 'mode'}},
  ↳ custom_name='Mode Baseline Binary Classification Pipeline', random_seed=0),
  'Elastic Net Classifier w/ Imputer + DateTime Featurization Component + One Hot_
  ↳ Encoder + SMOTENC Oversampler + Standard Scaler': pipeline =
  ↳ BinaryClassificationPipeline(component_graph={'Imputer': ['Imputer', 'X', 'y'],
  ↳ 'DateTime Featurization Component': ['DateTime Featurization Component', 'Imputer.x
  ↳ ', 'y'], 'One Hot Encoder': ['One Hot Encoder', 'DateTime Featurization Component.x
  ↳ ', 'y'], 'SMOTENC Oversampler': ['SMOTENC Oversampler', 'One Hot Encoder.x', 'y'],
  ↳ 'Standard Scaler': ['Standard Scaler', 'SMOTENC Oversampler.x', 'SMOTENC_
  ↳ Oversampler.y'], 'Elastic Net Classifier': ['Elastic Net Classifier', 'Standard_
  ↳ Scaler.x', 'SMOTENC Oversampler.y']}, parameters={'Imputer':{'categorical_impute_
  ↳ strategy': 'most_frequent', 'numeric_impute_strategy': 'mean', 'categorical_fill_
  ↳ value': None, 'numeric_fill_value': None}, 'DateTime Featurization Component':{'
  ↳ features_to_extract': ['year', 'month', 'day_of_week', 'hour'], 'encode_as_
  ↳ categories': False, 'date_index': None}, 'One Hot Encoder':{'top_n': 10, 'features_
  ↳ to_encode': None, 'categories': None, 'drop': 'if_binary', 'handle_unknown': 'ignore
  ↳ ', 'handle_missing': 'error'}, 'SMOTENC Oversampler':{'sampling_ratio': 0.25, 'k_
  ↳ neighbors_default': 5, 'n_jobs': -1, 'sampling_ratio_dict': None, 'categorical_
  ↳ features': [3, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26,
  ↳ 27, 28, 29, 30, 31, 32, 33, 34, 35, 36, 37, 38, 39, 40, 41, 42, 43, 44, 45, 46, 47,
  ↳ 48, 49, 50, 51, 52, 53, 54, 55, 56, 57, 58, 59], 'k_neighbors': 5}, 'Elastic Net_
  ↳ Classifier':{'penalty': 'elasticnet', 'C': 1.0, 'l1_ratio': 0.15, 'n_jobs': -1,
  ↳ 'multi_class': 'auto', 'solver': 'saga'}}}, random_seed=0),
  'Decision Tree Classifier w/ Imputer + DateTime Featurization Component + One Hot_
  ↳ Encoder + SMOTENC Oversampler': pipeline = BinaryClassificationPipeline(component_
  ↳ graph={'Imputer': ['Imputer', 'X', 'y'], 'DateTime Featurization Component': [
  ↳ 'DateTime Featurization Component', 'Imputer.x', 'y'], 'One Hot Encoder': ['One Hot_
  ↳ Encoder', 'DateTime Featurization Component.x', 'y'], 'SMOTENC Oversampler': [
  ↳ 'SMOTENC Oversampler', 'One Hot Encoder.x', 'y'], 'Decision Tree Classifier': [
  ↳ 'Decision Tree Classifier', 'SMOTENC Oversampler.x', 'SMOTENC Oversampler.y']},
  ↳ parameters={'Imputer':{'categorical_impute_strategy': 'most_frequent', 'numeric_
  ↳ impute_strategy': 'mean', 'categorical_fill_value': None, 'numeric_fill_value':
  ↳ None}, 'DateTime Featurization Component':{'features_to_extract': ['year', 'month',
  ↳ 'day_of_week', 'hour'], 'encode_as_categories': False, 'date_index': None}, 'One_
  ↳ Hot Encoder':{'top_n': 10, 'features_to_encode': None, 'categories': None, 'drop':
  ↳ 'if_binary', 'handle_unknown': 'ignore', 'handle_missing': 'error'}, 'SMOTENC_
  ↳ Oversampler':{'sampling_ratio': 0.25, 'k_neighbors_default': 5, 'n_jobs': -1,
  ↳ 'sampling_ratio_dict': None, 'categorical_features': [3, 10, 11, 12, 13, 14, 15,
  ↳ 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31, 32, 33, 34, 35, 36, 37,
  ↳ 38, 39, 40, 41, 42, 43, 44, 45, 46, 47, 48, 49, 50, 51, 52, 53, 54, 55, 56, 57, 58,
  ↳ 59], 'k_neighbors': 5}, 'Decision Tree Classifier':{'criterion': 'gini', 'max_
  ↳ features': 'auto', 'max_depth': 6, 'min_samples_split': 2, 'min_weight_fraction_leaf
  ↳

```

(continues on next page)

(continued from previous page)

```
[17]: pipeline_holdout_scores = automl.score_pipelines([trained_pipelines[name] for name in
↳ trained_pipelines.keys()],
                                                    X_holdout,
                                                    y_holdout,
                                                    ['Accuracy Binary', 'F1', 'AUC'])
pipeline_holdout_scores

[17]: {'Mode Baseline Binary Classification Pipeline': OrderedDict([('Accuracy Binary',
0.88),
('F1', 0.0),
('AUC', 0.5)]),
'Elastic Net Classifier w/ Imputer + DateTime Featurization Component + One Hot_
↳ Encoder + SMOTENC Oversampler + Standard Scaler': OrderedDict([('Accuracy Binary',
0.66),
('F1', 0.19047619047619044),
('AUC', 0.5265151515151515)]),
'Decision Tree Classifier w/ Imputer + DateTime Featurization Component + One Hot_
↳ Encoder + SMOTENC Oversampler': OrderedDict([('Accuracy Binary',
0.92),
('F1', 0.6),
('AUC', 0.7234848484848486)])}
```

### 4.1.7 Saving AutoMLSearch and pipelines from AutoMLSearch

There are two ways to save results from AutoMLSearch.

- You can save the AutoMLSearch object itself, calling `.save(<filepath>)` to do so. This will allow you to save the AutoMLSearch state and reload all pipelines from this.
- If you want to save a pipeline from AutoMLSearch for future use, pipeline classes themselves have a `.save(<filepath>)` method.

```
[18]: # saving the entire automl search
automl.save("automl.cloudpickle")
automl2 = evalml.automl.AutoMLSearch.load("automl.cloudpickle")
# saving the best pipeline using .save()
best_pipeline.save("pipeline.cloudpickle")
best_pipeline_copy = evalml.pipelines.PipelineBase.load("pipeline.cloudpickle")
```

### 4.1.8 Limiting the AutoML Search Space

The AutoML search algorithm first trains each component in the pipeline with their default values. After the first iteration, it then tweaks the parameters of these components using the pre-defined hyperparameter ranges that these components have. To limit the search over certain hyperparameter ranges, you can specify a `custom_hyperparameters` argument with your AutoMLSearch parameters. These parameters will limit the hyperparameter search space.

Hyperparameter ranges can be found through the [API reference](#) for each component. Parameter arguments must be specified as dictionaries, but the associated values can be single values or `skopt.space` Real, Integer, Categorical values.

If however you'd like to specify certain values for the initial batch of the AutoML search algorithm, you can use the `pipeline_parameters` argument. This will set the initial batch's component parameters to the values passed by

this argument.

```
[19]: from evalml import AutoMLSearch
from evalml.demos import load_fraud
from skopt.space import Categorical
from evalml.model_family import ModelFamily
import woodwork as ww

X, y = load_fraud(n_rows=1000)

# example of setting parameter to just one value
custom_hyperparameters = {'Imputer': {
    'numeric_impute_strategy': 'mean'
}}

# limit the numeric impute strategy to include only `median` and `most_frequent`
# `mean` is the default value for this argument, but it doesn't need to be included,
# in the specified hyperparameter range for this to work
custom_hyperparameters = {'Imputer': {
    'numeric_impute_strategy': Categorical(['median', 'most_frequent'])
}}
# set the initial batch numeric impute strategy strategy to 'median'
pipeline_parameters = {'Imputer': {
    'numeric_impute_strategy': 'median'
}}

# using this custom hyperparameter means that our Imputer components in these
# pipelines will only search through
# 'median' and 'most_frequent' strategies for 'numeric_impute_strategy', and the
# initial batch parameter will be
# set to 'median'
automl_constrained = AutoMLSearch(X_train=X, y_train=y, problem_type='binary',
    pipeline_parameters=pipeline_parameters,
    custom_hyperparameters=custom_hyperparameters)
```

	Number of Features
Boolean	1
Categorical	6
Numeric	5

Number of training examples: 1000  
Targets  
False 85.90%  
True 14.10%  
Name: fraud, dtype: object  
Using default limit of max\_batches=1.

Generating pipelines to search over...  
8 pipelines ready for search.

### 4.1.9 Imbalanced Data

The AutoML search algorithm now has functionality to handle imbalanced data during classification! AutoMLSearch now provides two additional parameters, `sampler_method` and `sampler_balanced_ratio`, that allow you to let AutoMLSearch know whether to sample imbalanced data, and how to do so. `sampler_method` takes in either `Undersampler`, `Oversampler`, `auto`, or `None` as the sampler to use, and `sampler_balanced_ratio` specifies the minority/majority ratio that you want to sample to. Details on the `Undersampler` and `Oversampler` components can be found in the [documentation](#).

This can be used for imbalanced datasets, like the fraud dataset, which has a ‘minority:majority’ ratio of  $< 0.2$ .

```
[20]: automl_auto = AutoMLSearch(X_train=X, y_train=y, problem_type='binary')
      automl_auto.allowed_pipelines[-1]

Using default limit of max_batches=1.

Generating pipelines to search over...
8 pipelines ready for search.

[20]: pipeline = BinaryClassificationPipeline(component_graph={'Imputer': ['Imputer', 'X',
↪ 'y'], 'DateTime Featurization Component': ['DateTime Featurization Component',
↪ 'Imputer.x', 'y'], 'One Hot Encoder': ['One Hot Encoder', 'DateTime Featurization_
↪ Component.x', 'y'], 'SMOTENC Oversampler': ['SMOTENC Oversampler', 'One Hot Encoder.
↪ x', 'y'], 'Standard Scaler': ['Standard Scaler', 'SMOTENC Oversampler.x', 'SMOTENC_
↪ Oversampler.y'], 'Logistic Regression Classifier': ['Logistic Regression Classifier
↪ ', 'Standard Scaler.x', 'SMOTENC Oversampler.y']}, parameters={'Imputer':{
↪ 'categorical_impute_strategy': 'most_frequent', 'numeric_impute_strategy': 'mean',
↪ 'categorical_fill_value': None, 'numeric_fill_value': None}, 'DateTime_
↪ Featurization Component':{'features_to_extract': ['year', 'month', 'day_of_week',
↪ 'hour'], 'encode_as_categories': False, 'date_index': None}, 'One Hot Encoder':{'
↪ 'top_n': 10, 'features_to_encode': None, 'categories': None, 'drop': 'if_binary',
↪ 'handle_unknown': 'ignore', 'handle_missing': 'error'}, 'SMOTENC Oversampler':{'
↪ 'sampling_ratio': 0.25, 'k_neighbors_default': 5, 'n_jobs': -1, 'sampling_ratio_dict
↪ ': None}, 'Logistic Regression Classifier':{'penalty': 'l2', 'C': 1.0, 'n_jobs': -1,
↪ 'multi_class': 'auto', 'solver': 'lbfgs'}}}, random_seed=0)
```

The `SMOTENC Oversampler` is chosen as the default sampling component here, since the `sampler_balanced_ratio = 0.25`. If you specified a lower ratio, for instance `sampler_balanced_ratio = 0.1`, then there would be no sampling component added here. This is because if a ratio of 0.1 would be considered balanced, then a ratio of 0.2 would also be balanced.

```
[21]: automl_auto_ratio = AutoMLSearch(X_train=X, y_train=y, problem_type='binary', sampler_
↪ balanced_ratio=0.1)
      automl_auto_ratio.allowed_pipelines[-1]

Using default limit of max_batches=1.

Generating pipelines to search over...
8 pipelines ready for search.

[21]: pipeline = BinaryClassificationPipeline(component_graph={'Imputer': ['Imputer', 'X',
↪ 'y'], 'DateTime Featurization Component': ['DateTime Featurization Component',
↪ 'Imputer.x', 'y'], 'One Hot Encoder': ['One Hot Encoder', 'DateTime Featurization_
↪ Component.x', 'y'], 'Standard Scaler': ['Standard Scaler', 'One Hot Encoder.x', 'y
↪ '], 'Logistic Regression Classifier': ['Logistic Regression Classifier', 'Standard_
↪ Scaler.x', 'y']}, parameters={'Imputer':{'categorical_impute_strategy': 'most_
↪ frequent', 'numeric_impute_strategy': 'mean', 'categorical_fill_value': None,
↪ 'numeric_fill_value': None}, 'DateTime Featurization Component':{'features_to_
↪ extract': ['year', 'month', 'day_of_week', 'hour'], 'encode_as_categories': False,
↪ 'date_index': None}, 'One Hot Encoder':{'top_n': 10, 'features_to_encode': None,
↪ 'categories': None, 'drop': 'if_binary', 'handle_unknown': 'ignore', 'handle_missing'
↪ ': 'error'}, 'Logistic Regression Classifier':{'penalty': 'l2', 'C': 1.0, 'n_jobs': -
↪ 1, 'multi_class': 'auto', 'solver': 'lbfgs'}}}, random_seed=0)
```

(continued from previous page)

Additionally, you can add more fine-grained sampling ratios by passing in a `sampling_ratio_dict` in pipeline parameters. For this dictionary, AutoMLSearch expects the keys to be int values from 0 to  $n-1$  for the classes, and the values would be the `sampler_balanced_ratio` associated with each target. This dictionary would override the AutoML argument `sampler_balanced_ratio`. Below, you can see the scenario for Oversampler component on this dataset. Note that the logic for Undersamplers is included in the commented section.

```
[22]: # In this case, the majority class is the negative class
# for the oversampler, we don't want to oversample this class, so class 0 (majority)
# will have a ratio of 1 to itself
# for the minority class 1, we want to oversample it to have a minority/majority
# ratio of 0.5, which means we want minority to have 1/2 the samples as the majority
sampler_ratio_dict = {0: 1, 1: 0.5}
pipeline_parameters = {"SMOTENC Oversampler": {"sampler_balanced_ratio": sampler_ratio_dict}}
automl_auto_ratio_dict = AutoMLSearch(X_train=X, y_train=y, problem_type='binary',
                                     pipeline_parameters=pipeline_parameters)
automl_auto_ratio_dict.allowed_pipelines[-1]

# Undersampler case
# we don't want to undersample this class, so class 1 (minority) will have a ratio of
# 1 to itself
# for the majority class 0, we want to undersample it to have a minority/majority
# ratio of 0.5, which means we want majority to have 2x the samples as the minority
# sampler_ratio_dict = {0: 0.5, 1: 1}
# pipeline_parameters = {"SMOTENC Oversampler": {"sampler_balanced_ratio": sampler_ratio_dict}}
# automl_auto_ratio_dict = AutoMLSearch(X_train=X, y_train=y, problem_type='binary',
#                                     pipeline_parameters=pipeline_parameters)
```

Using default limit of `max_batches=1`.

Generating pipelines to search over...

8 pipelines ready for search.

```
[22]: pipeline = BinaryClassificationPipeline(component_graph={'Imputer': ['Imputer', 'X',
# 'y'], 'DateTime Featurization Component': ['DateTime Featurization Component',
# 'Imputer.x', 'y'], 'One Hot Encoder': ['One Hot Encoder', 'DateTime Featurization
# Component.x', 'y'], 'SMOTENC Oversampler': ['SMOTENC Oversampler', 'One Hot Encoder.
# x', 'y'], 'Standard Scaler': ['Standard Scaler', 'SMOTENC Oversampler.x', 'SMOTENC
# Oversampler.y'], 'Logistic Regression Classifier': ['Logistic Regression Classifier
# ', 'Standard Scaler.x', 'SMOTENC Oversampler.y']}, parameters={'Imputer':{
# 'categorical_impute_strategy': 'most_frequent', 'numeric_impute_strategy': 'mean',
# 'categorical_fill_value': None, 'numeric_fill_value': None}, 'DateTime
# Featurization Component':{'features_to_extract': ['year', 'month', 'day_of_week',
# 'hour'], 'encode_as_categories': False, 'date_index': None}, 'One Hot Encoder':{'
# top_n': 10, 'features_to_encode': None, 'categories': None, 'drop': 'if_binary',
# 'handle_unknown': 'ignore', 'handle_missing': 'error'}, 'SMOTENC Oversampler':{'
# 'sampling_ratio': 0.25, 'k_neighbors_default': 5, 'n_jobs': -1, 'sampling_ratio_dict
# ': None, 'sampler_balanced_ratio': {0: 1, 1: 0.5}}, 'Logistic Regression Classifier
# ':{'penalty': 'l2', 'C': 1.0, 'n_jobs': -1, 'multi_class': 'auto', 'solver': 'lbfgs
# '}}, random_seed=0)
```



## 4.1.10 Adding ensemble methods to AutoML

### Stacking

**Stacking** is an ensemble machine learning algorithm that involves training a model to best combine the predictions of several base learning algorithms. First, each base learning algorithm is trained using the given data. Then, the combining algorithm or meta-learner is trained on the predictions made by those base learning algorithms to make a final prediction.

AutoML enables stacking using the `ensembling` flag during initialization; this is set to `False` by default. The stacking ensemble pipeline runs in its own batch after a whole cycle of training has occurred (each allowed pipeline trains for one batch). Note that this means **a large number of iterations may need to run before the stacking ensemble runs**. It is also important to note that **only the first CV fold is calculated for stacking ensembles** because the model internally uses CV folds.

```
[23]: X, y = evalml.demos.load_breast_cancer()

automl_with_ensembling = AutoMLSearch(X_train=X, y_train=y,
                                     problem_type="binary",
                                     allowed_model_families=[ModelFamily.LINEAR_
                                     ↪MODEL],
                                     max_batches=4,
                                     ensembling=True)

automl_with_ensembling.search()
```

```

      Number of Features
Numeric                30

Number of training examples: 569
Targets
benign          62.74%
malignant       37.26%
Name: target, dtype: object
Generating pipelines to search over...
2 pipelines ready for search.
Ensembling will run every 3 batches.
```

```
*****
* Beginning pipeline search *
*****
```

```
Optimizing for Log Loss Binary.
Lower score is better.
```

```
Using SequentialEngine to train and score pipelines.
Searching up to 4 batches for a total of 14 pipelines.
Allowed model families: linear_model
```

```
FigureWidget({
  'data': [{'mode': 'lines+markers',
            'name': 'Best Score',
            'type': ...
```

```
Evaluating Baseline Pipeline: Mode Baseline Binary Classification Pipeline
Mode Baseline Binary Classification Pipeline:
  Starting cross validation
  Finished cross validation - mean Log Loss Binary: 12.868
```

(continues on next page)

(continued from previous page)

```

*****
* Evaluating Batch Number 1 *
*****

Elastic Net Classifier w/ Imputer + Standard Scaler:
    Starting cross validation
    Finished cross validation - mean Log Loss Binary: 0.077
Logistic Regression Classifier w/ Imputer + Standard Scaler:
    Starting cross validation
    Finished cross validation - mean Log Loss Binary: 0.077
    High coefficient of variation (cv >= 0.2) within cross validation scores.
    Logistic Regression Classifier w/ Imputer + Standard Scaler may not perform
↳as estimated on unseen data.

*****
* Evaluating Batch Number 2 *
*****

Logistic Regression Classifier w/ Imputer + Standard Scaler:
    Starting cross validation
    Finished cross validation - mean Log Loss Binary: 0.097
    High coefficient of variation (cv >= 0.2) within cross validation scores.
    Logistic Regression Classifier w/ Imputer + Standard Scaler may not perform
↳as estimated on unseen data.
Logistic Regression Classifier w/ Imputer + Standard Scaler:
    Starting cross validation
    Finished cross validation - mean Log Loss Binary: 0.085
    High coefficient of variation (cv >= 0.2) within cross validation scores.
    Logistic Regression Classifier w/ Imputer + Standard Scaler may not perform
↳as estimated on unseen data.
Logistic Regression Classifier w/ Imputer + Standard Scaler:
    Starting cross validation
    Finished cross validation - mean Log Loss Binary: 0.097
    High coefficient of variation (cv >= 0.2) within cross validation scores.
    Logistic Regression Classifier w/ Imputer + Standard Scaler may not perform
↳as estimated on unseen data.
Logistic Regression Classifier w/ Imputer + Standard Scaler:
    Starting cross validation
    Finished cross validation - mean Log Loss Binary: 0.091
    High coefficient of variation (cv >= 0.2) within cross validation scores.
    Logistic Regression Classifier w/ Imputer + Standard Scaler may not perform
↳as estimated on unseen data.
Logistic Regression Classifier w/ Imputer + Standard Scaler:
    Starting cross validation
    Finished cross validation - mean Log Loss Binary: 0.080

*****
* Evaluating Batch Number 3 *
*****

Elastic Net Classifier w/ Imputer + Standard Scaler:
    Starting cross validation
    Finished cross validation - mean Log Loss Binary: 0.075
    High coefficient of variation (cv >= 0.2) within cross validation scores.
    Elastic Net Classifier w/ Imputer + Standard Scaler may not perform as
↳estimated on unseen data.
Elastic Net Classifier w/ Imputer + Standard Scaler:

```

(continues on next page)

(continued from previous page)

```

    Starting cross validation
    Finished cross validation - mean Log Loss Binary: 0.075
    High coefficient of variation (cv >= 0.2) within cross validation scores.
    Elastic Net Classifier w/ Imputer + Standard Scaler may not perform as
    ↪estimated on unseen data.
Elastic Net Classifier w/ Imputer + Standard Scaler:
    Starting cross validation
    Finished cross validation - mean Log Loss Binary: 0.079
Elastic Net Classifier w/ Imputer + Standard Scaler:
    Starting cross validation
    Finished cross validation - mean Log Loss Binary: 0.076
    High coefficient of variation (cv >= 0.2) within cross validation scores.
    Elastic Net Classifier w/ Imputer + Standard Scaler may not perform as
    ↪estimated on unseen data.
Elastic Net Classifier w/ Imputer + Standard Scaler:
    Starting cross validation
    Finished cross validation - mean Log Loss Binary: 0.075
    High coefficient of variation (cv >= 0.2) within cross validation scores.
    Elastic Net Classifier w/ Imputer + Standard Scaler may not perform as
    ↪estimated on unseen data.

*****
* Evaluating Batch Number 4 *
*****

Sklearn Stacked Ensemble Classification Pipeline:
    Starting cross validation
    Finished cross validation - mean Log Loss Binary: 0.123

Search finished after 00:26
Best pipeline: Elastic Net Classifier w/ Imputer + Standard Scaler
Best pipeline Log Loss Binary: 0.075387

```

We can view more information about the stacking ensemble pipeline (which was the best performing pipeline) by calling `.describe()`.

```
[24]: automl_with_ensembling.best_pipeline.describe()
```

```

*****
* Elastic Net Classifier w/ Imputer + Standard Scaler *
*****

Problem Type: binary
Model Family: Linear
Number of features: 30

Pipeline Steps
=====
1. Imputer
    * categorical_impute_strategy : most_frequent
    * numeric_impute_strategy : median
    * categorical_fill_value : None
    * numeric_fill_value : None
2. Standard Scaler
3. Elastic Net Classifier
    * penalty : elasticnet

```

(continues on next page)

(continued from previous page)

```
* C : 8.123565600467177
* ll_ratio : 0.47997717237505744
* n_jobs : -1
* multi_class : auto
* solver : saga
```

### 4.1.11 Access raw results

The `AutoMLSearch` class records detailed results information under the `results` field, including information about the cross-validation scoring and parameters.

```
[25]: automl.results
```

```
[25]: {'pipeline_results': {0: {'id': 0,
    'pipeline_name': 'Mode Baseline Binary Classification Pipeline',
    'pipeline_class': evalml.pipelines.binary_classification_pipeline.
    ↪BinaryClassificationPipeline,
    'pipeline_summary': 'Baseline Classifier',
    'parameters': {'Baseline Classifier': {'strategy': 'mode'}},
    'mean_cv_score': 3.970423187263591,
    'standard_deviation_cv_score': 0.26606000431837074,
    'high_variance_cv': False,
    'training_time': 0.9191300868988037,
    'cv_data': [{'all_objective_scores': OrderedDict([('Log Loss Binary',
        4.124033002377396),
        ('MCC Binary', 0.0),
        ('Gini', 0.0),
        ('AUC', 0.5),
        ('Precision', 0.0),
        ('F1', 0.0),
        ('Balanced Accuracy Binary', 0.5),
        ('Accuracy Binary', 0.8805970149253731),
        ('# Training', 133),
        ('# Validation', 67)]),
    'mean_cv_score': 4.124033002377396,
    'binary_classification_threshold': 9.16384630183206e-53},
    {'all_objective_scores': OrderedDict([('Log Loss Binary',
        4.124033002377395),
        ('MCC Binary', 0.0),
        ('Gini', 0.0),
        ('AUC', 0.5),
        ('Precision', 0.0),
        ('F1', 0.0),
        ('Balanced Accuracy Binary', 0.5),
        ('Accuracy Binary', 0.8805970149253731),
        ('# Training', 133),
        ('# Validation', 67)]),
    'mean_cv_score': 4.124033002377395,
    'binary_classification_threshold': 9.16384630183206e-53},
    {'all_objective_scores': OrderedDict([('Log Loss Binary',
        3.6632035570359824),
        ('MCC Binary', 0.0),
        ('Gini', 0.0),
        ('AUC', 0.5),
        ('Precision', 0.0),
```

(continues on next page)

(continued from previous page)

```

        ('F1', 0.0),
        ('Balanced Accuracy Binary', 0.5),
        ('Accuracy Binary', 0.8939393939393939),
        ('# Training', 134),
        ('# Validation', 66)]),
    'mean_cv_score': 3.6632035570359824,
    'binary_classification_threshold': 9.16384630183206e-53}},
    'percent_better_than_baseline_all_objectives': {'Log Loss Binary': 0,
    'MCC Binary': 0,
    'Gini': 0,
    'AUC': 0,
    'Precision': 0,
    'F1': 0,
    'Balanced Accuracy Binary': 0,
    'Accuracy Binary': 0},
    'percent_better_than_baseline': 0,
    'validation_score': 4.124033002377396},
1: {'id': 1,
    'pipeline_name': 'Elastic Net Classifier w/ Imputer + DateTime Featurization_
↳Component + One Hot Encoder + SMOTENC Oversampler + Standard Scaler',
    'pipeline_class': evalml.pipelines.binary_classification_pipeline.
↳BinaryClassificationPipeline,
    'pipeline_summary': 'Elastic Net Classifier w/ Imputer + DateTime Featurization_
↳Component + One Hot Encoder + SMOTENC Oversampler + Standard Scaler',
    'parameters': {'Imputer': {'categorical_impute_strategy': 'most_frequent',
    'numeric_impute_strategy': 'mean',
    'categorical_fill_value': None,
    'numeric_fill_value': None},
    'DateTime Featurization Component': {'features_to_extract': ['year',
    'month',
    'day_of_week',
    'hour'],
    'encode_as_categories': False,
    'date_index': None},
    'One Hot Encoder': {'top_n': 10,
    'features_to_encode': None,
    'categories': None,
    'drop': 'if_binary',
    'handle_unknown': 'ignore',
    'handle_missing': 'error'},
    'SMOTENC Oversampler': {'sampling_ratio': 0.25,
    'k_neighbors_default': 5,
    'n_jobs': -1,
    'sampling_ratio_dict': None,
    'categorical_features': [3,
    10,
    11,
    12,
    13,
    14,
    15,
    16,
    17,
    18,
    19,
    20,
    21,

```

(continues on next page)

(continued from previous page)

```

22,
23,
24,
25,
26,
27,
28,
29,
30,
31,
32,
33,
34,
35,
36,
37,
38,
39,
40,
41,
42,
43,
44,
45,
46,
47,
48,
49,
50,
51,
52,
53,
54,
55,
56,
57,
58,
59],
'k_neighbors': 5},
'Elastic Net Classifier': {'penalty': 'elasticnet',
'C': 1.0,
'l1_ratio': 0.15,
'n_jobs': -1,
'multi_class': 'auto',
'solver': 'saga'}},
'mean_cv_score': 0.5118084226551022,
'standard_deviation_cv_score': 0.07499222153032764,
'high_variance_cv': False,
'training_time': 2.7962565422058105,
'cv_data': [{'all_objective_scores': OrderedDict([('Log Loss Binary',
0.590516914375052),
('MCC Binary', 0.17518582316850065),
('Gini', 0.19491525423728806),
('AUC', 0.597457627118644),
('Precision', 0.2857142857142857),
('F1', 0.26666666666666666),
('Balanced Accuracy Binary', 0.5826271186440678),

```

(continues on next page)

(continued from previous page)

```

        ('Accuracy Binary', 0.835820895522388),
        ('# Training', 133),
        ('# Validation', 67)]),
    'mean_cv_score': 0.590516914375052,
    'binary_classification_threshold': 0.48089822234602875},
    {'all_objective_scores': OrderedDict([('Log Loss Binary',
        0.44118816828058105),
        ('MCC Binary', 0.2957528252716689),
        ('Gini', 0.4194915254237288),
        ('AUC', 0.7097457627118644),
        ('Precision', 0.5),
        ('F1', 0.3333333333333333),
        ('Balanced Accuracy Binary', 0.6080508474576272),
        ('Accuracy Binary', 0.8805970149253731),
        ('# Training', 133),
        ('# Validation', 67)]),
    'mean_cv_score': 0.44118816828058105,
    'binary_classification_threshold': 0.7518116371406225},
    {'all_objective_scores': OrderedDict([('Log Loss Binary',
        0.5037201853096736),
        ('MCC Binary', 0.035350118786872956),
        ('Gini', 0.2590799031476996),
        ('AUC', 0.6295399515738498),
        ('Precision', 0.12),
        ('F1', 0.1875),
        ('Balanced Accuracy Binary', 0.5278450363196125),
        ('Accuracy Binary', 0.6060606060606061),
        ('# Training', 134),
        ('# Validation', 66)]),
    'mean_cv_score': 0.5037201853096736,
    'binary_classification_threshold': 0.10568708806540947}},
    'percent_better_than_baseline_all_objectives': {'Log Loss Binary': 87.
↪10947426720426,
    'MCC Binary': inf,
    'Gini': inf,
    'AUC': 14.558111380145277,
    'Precision': 30.19047619047619,
    'F1': 26.25,
    'Balanced Accuracy Binary': 7.284100080710254,
    'Accuracy Binary': -11.08849690939243},
    'percent_better_than_baseline': 87.10947426720426,
    'validation_score': 0.590516914375052},
    2: {'id': 2,
    'pipeline_name': 'Decision Tree Classifier w/ Imputer + DateTime Featurization_
↪Component + One Hot Encoder + SMOTENC Oversampler',
    'pipeline_class': evalml.pipelines.binary_classification_pipeline.
↪BinaryClassificationPipeline,
    'pipeline_summary': 'Decision Tree Classifier w/ Imputer + DateTime Featurization_
↪Component + One Hot Encoder + SMOTENC Oversampler',
    'parameters': {'Imputer': {'categorical_impute_strategy': 'most_frequent',
    'numeric_impute_strategy': 'mean',
    'categorical_fill_value': None,
    'numeric_fill_value': None},
    'DateTime Featurization Component': {'features_to_extract': ['year',
    'month',
    'day_of_week',
    'hour'],

```

(continues on next page)

(continued from previous page)

```
'encode_as_categories': False,
'date_index': None},
'One Hot Encoder': {'top_n': 10,
'features_to_encode': None,
'categories': None,
'drop': 'if_binary',
'handle_unknown': 'ignore',
'handle_missing': 'error'},
'SMOTENC Oversampler': {'sampling_ratio': 0.25,
'k_neighbors_default': 5,
'n_jobs': -1,
'sampling_ratio_dict': None,
'categorical_features': [3,
10,
11,
12,
13,
14,
15,
16,
17,
18,
19,
20,
21,
22,
23,
24,
25,
26,
27,
28,
29,
30,
31,
32,
33,
34,
35,
36,
37,
38,
39,
40,
41,
42,
43,
44,
45,
46,
47,
48,
49,
50,
51,
52,
53,
```

(continues on next page)



(continued from previous page)

```

54,
55,
56,
57,
58,
59],
'k_neighbors': 5},
'Decision Tree Classifier': {'criterion': 'gini',
'max_features': 'auto',
'max_depth': 6,
'min_samples_split': 2,
'min_weight_fraction_leaf': 0.0}},
'mean_cv_score': 2.956956288246977,
'standard_deviation_cv_score': 3.084438905133218,
'high_variance_cv': True,
'training_time': 2.0632965564727783,
'cv_data': [{'all_objective_scores': OrderedDict([('Log Loss Binary',
0.651556943513563),
('MCC Binary', 0.7712055916006633),
('Gini', 0.6207627118644068),
('AUC', 0.8103813559322034),
('Precision', 1.0),
('F1', 0.7692307692307693),
('Balanced Accuracy Binary', 0.8125),
('Accuracy Binary', 0.9552238805970149),
('# Training', 133),
('# Validation', 67)]),
'mean_cv_score': 0.651556943513563,
'binary_classification_threshold': 0.9999999885588493},
{'all_objective_scores': OrderedDict([('Log Loss Binary',
1.7585681792927779),
('MCC Binary', 0.4208952550769721),
('Gini', 0.379237288135593),
('AUC', 0.6896186440677965),
('Precision', 0.6),
('F1', 0.4615384615384615),
('Balanced Accuracy Binary', 0.6705508474576272),
('Accuracy Binary', 0.8955223880597015),
('# Training', 133),
('# Validation', 67)]),
'mean_cv_score': 1.7585681792927779,
'binary_classification_threshold': 0.9999999885588493},
{'all_objective_scores': OrderedDict([('Log Loss Binary',
6.460743741934591),
('MCC Binary', -0.20116515012026231),
('Gini', -0.29539951573849876),
('AUC', 0.3523002421307506),
('Precision', 0.06521739130434782),
('F1', 0.11320754716981131),
('Balanced Accuracy Binary', 0.3498789346246973),
('Accuracy Binary', 0.2878787878787879),
('# Training', 134),
('# Validation', 66)]),
'mean_cv_score': 6.460743741934591,
'binary_classification_threshold': 0.1568627424310231}},
'percent_better_than_baseline_all_objectives': {'Log Loss Binary': 25.
↪52541256225873,

```

(continues on next page)

(continued from previous page)

```

'MCC Binary': inf,
'Gini': inf,
'AUC': 11.743341404358354,
'Precision': 55.5072463768116,
'F1': 44.79922593130141,
'Balanced Accuracy Binary': 11.097659402744153,
'Accuracy Binary': -17.21694557515452},
'percent_better_than_baseline': 25.52541256225873,
'validation_score': 0.651556943513563},
3: {'id': 3,
    'pipeline_name': 'Random Forest Classifier w/ Imputer + DateTime Featurization_
↳Component + One Hot Encoder + SMOTENC Oversampler',
    'pipeline_class': evalml.pipelines.binary_classification_pipeline.
↳BinaryClassificationPipeline,
    'pipeline_summary': 'Random Forest Classifier w/ Imputer + DateTime Featurization_
↳Component + One Hot Encoder + SMOTENC Oversampler',
    'parameters': {'Imputer': {'categorical_impute_strategy': 'most_frequent',
                                'numeric_impute_strategy': 'mean',
                                'categorical_fill_value': None,
                                'numeric_fill_value': None},
                    'DateTime Featurization Component': {'features_to_extract': ['year',
                                        'month',
                                        'day_of_week',
                                        'hour'],
                    'encode_as_categories': False,
                    'date_index': None},
                    'One Hot Encoder': {'top_n': 10,
                                        'features_to_encode': None,
                                        'categories': None,
                                        'drop': 'if_binary',
                                        'handle_unknown': 'ignore',
                                        'handle_missing': 'error'},
                    'SMOTENC Oversampler': {'sampling_ratio': 0.25,
                                        'k_neighbors_default': 5,
                                        'n_jobs': -1,
                                        'sampling_ratio_dict': None,
                                        'categorical_features': [3,
                                                                10,
                                                                11,
                                                                12,
                                                                13,
                                                                14,
                                                                15,
                                                                16,
                                                                17,
                                                                18,
                                                                19,
                                                                20,
                                                                21,
                                                                22,
                                                                23,
                                                                24,
                                                                25,
                                                                26,
                                                                27,
                                                                28,
                                                                29,

```

(continues on next page)

(continued from previous page)

```

30,
31,
32,
33,
34,
35,
36,
37,
38,
39,
40,
41,
42,
43,
44,
45,
46,
47,
48,
49,
50,
51,
52,
53,
54,
55,
56,
57,
58,
59],
'k_neighbors': 5},
'Random Forest Classifier': {'n_estimators': 100,
'max_depth': 6,
'n_jobs': -1}},
'mean_cv_score': 0.2856133806139974,
'standard_deviation_cv_score': 0.041115539258178,
'high_variance_cv': False,
'training_time': 2.662137269973755,
'cv_data': [{'all_objective_scores': OrderedDict([('Log Loss Binary',
0.26369520661129536),
('MCC Binary', 0.47636443708895493),
('Gini', 0.6483050847457628),
('AUC', 0.8241525423728814),
('Precision', 1.0),
('F1', 0.4),
('Balanced Accuracy Binary', 0.625),
('Accuracy Binary', 0.9104477611940298),
('# Training', 133),
('# Validation', 67)])},
'mean_cv_score': 0.26369520661129536,
'binary_classification_threshold': 0.4988241268809967},
{'all_objective_scores': OrderedDict([('Log Loss Binary',
0.33304413887923806),
('MCC Binary', 0.4900218379501181),
('Gini', 0.423728813559322),
('AUC', 0.711864406779661),
('Precision', 0.75),

```

(continues on next page)

(continued from previous page)

```

        ('F1', 0.5),
        ('Balanced Accuracy Binary', 0.6790254237288136),
        ('Accuracy Binary', 0.9104477611940298),
        ('# Training', 133),
        ('# Validation', 67)]),
    'mean_cv_score': 0.33304413887923806,
    'binary_classification_threshold': 0.5065550313835498},
{'all_objective_scores': OrderedDict([('Log Loss Binary',
    0.2601007963514588),
    ('MCC Binary', 0.6335302236023843),
    ('Gini', 0.757869249394673),
    ('AUC', 0.8789346246973365),
    ('Precision', 1.0),
    ('F1', 0.6),
    ('Balanced Accuracy Binary', 0.7142857142857143),
    ('Accuracy Binary', 0.9393939393939394),
    ('# Training', 134),
    ('# Validation', 66)]),
    'mean_cv_score': 0.2601007963514588,
    'binary_classification_threshold': 0.4459844896470452}],
'percent_better_than_baseline_all_objectives': {'Log Loss Binary': 92.
↪80647509992905,
    'MCC Binary': inf,
    'Gini': inf,
    'AUC': 30.498385794995965,
    'Precision': 91.66666666666666,
    'F1': 50.0,
    'Balanced Accuracy Binary': 17.277037933817596,
    'Accuracy Binary': 3.505201266395308},
'percent_better_than_baseline': 92.80647509992905,
'validation_score': 0.26369520661129536},
4: {'id': 4,
    'pipeline_name': 'LightGBM Classifier w/ Imputer + DateTime Featurization_
↪Component + One Hot Encoder + SMOTENC Oversampler',
    'pipeline_class': evalml.pipelines.binary_classification_pipeline.
↪BinaryClassificationPipeline,
    'pipeline_summary': 'LightGBM Classifier w/ Imputer + DateTime Featurization_
↪Component + One Hot Encoder + SMOTENC Oversampler',
    'parameters': {'Imputer': {'categorical_impute_strategy': 'most_frequent',
    'numeric_impute_strategy': 'mean',
    'categorical_fill_value': None,
    'numeric_fill_value': None},
    'DateTime Featurization Component': {'features_to_extract': ['year',
    'month',
    'day_of_week',
    'hour'],
    'encode_as_categories': False,
    'date_index': None},
    'One Hot Encoder': {'top_n': 10,
    'features_to_encode': None,
    'categories': None,
    'drop': 'if_binary',
    'handle_unknown': 'ignore',
    'handle_missing': 'error'},
    'SMOTENC Oversampler': {'sampling_ratio': 0.25,
    'k_neighbors_default': 5,
    'n_jobs': -1,

```

(continues on next page)

(continued from previous page)

```
'sampling_ratio_dict': None,
'categorical_features': [3,
10,
11,
12,
13,
14,
15,
16,
17,
18,
19,
20,
21,
22,
23,
24,
25,
26,
27,
28,
29,
30,
31,
32,
33,
34,
35,
36,
37,
38,
39,
40,
41,
42,
43,
44,
45,
46,
47,
48,
49,
50,
51,
52,
53,
54,
55,
56,
57,
58,
59],
'k_neighbors': 5},
'LightGBM Classifier': {'boosting_type': 'gbdt',
'learning_rate': 0.1,
'n_estimators': 100,
'max_depth': 0,
```

(continues on next page)

(continued from previous page)

```

    'num_leaves': 31,
    'min_child_samples': 20,
    'n_jobs': -1,
    'bagging_freq': 0,
    'bagging_fraction': 0.9}},
'mean_cv_score': 0.30863594948686357,
'standard_deviation_cv_score': 0.20387814963979614,
'high_variance_cv': True,
'training_time': 2.451615333557129,
'cv_data': [{ 'all_objective_scores': OrderedDict([('Log Loss Binary',
    0.23494740261438743),
    ('MCC Binary', 0.4900218379501181),
    ('Gini', 0.7881355932203391),
    ('AUC', 0.8940677966101696),
    ('Precision', 0.75),
    ('F1', 0.5),
    ('Balanced Accuracy Binary', 0.6790254237288136),
    ('Accuracy Binary', 0.9104477611940298),
    ('# Training', 133),
    ('# Validation', 67)]),
'mean_cv_score': 0.23494740261438743,
'binary_classification_threshold': 0.5037478165498673},
{'all_objective_scores': OrderedDict([('Log Loss Binary',
    0.5391133772263208),
    ('MCC Binary', 0.4900218379501181),
    ('Gini', 0.27118644067796605),
    ('AUC', 0.635593220338983),
    ('Precision', 0.75),
    ('F1', 0.5),
    ('Balanced Accuracy Binary', 0.6790254237288136),
    ('Accuracy Binary', 0.9104477611940298),
    ('# Training', 133),
    ('# Validation', 67)]),
'mean_cv_score': 0.5391133772263208,
'binary_classification_threshold': 0.7736716431132877},
{'all_objective_scores': OrderedDict([('Log Loss Binary',
    0.15184706861988254),
    ('MCC Binary', 0.3600976668493281),
    ('Gini', 0.7869249394673123),
    ('AUC', 0.8934624697336562),
    ('Precision', 1.0),
    ('F1', 0.25),
    ('Balanced Accuracy Binary', 0.5714285714285714),
    ('Accuracy Binary', 0.9090909090909091),
    ('# Training', 134),
    ('# Validation', 66)]),
'mean_cv_score': 0.15184706861988254,
'binary_classification_threshold': 0.9385567379765596}],
'percent_better_than_baseline_all_objectives': {'Log Loss Binary': 92.
→ 22662333635083,
'MCC Binary': inf,
'Gini': inf,
'AUC': 30.77078288942697,
'Precision': 83.33333333333334,
'F1': 41.66666666666667,
'Balanced Accuracy Binary': 14.315980629539949,
'Accuracy Binary': 2.4951002562942914},

```

(continues on next page)

(continued from previous page)

```

    'percent_better_than_baseline': 92.22662333635083,
    'validation_score': 0.23494740261438743},
5: {'id': 5,
    'pipeline_name': 'Logistic Regression Classifier w/ Imputer + DateTime_
↳Featurization Component + One Hot Encoder + SMOTENC Oversampler + Standard Scaler',
    'pipeline_class': evalml.pipelines.binary_classification_pipeline.
↳BinaryClassificationPipeline,
    'pipeline_summary': 'Logistic Regression Classifier w/ Imputer + DateTime_
↳Featurization Component + One Hot Encoder + SMOTENC Oversampler + Standard Scaler',
    'parameters': {'Imputer': {'categorical_impute_strategy': 'most_frequent',
    'numeric_impute_strategy': 'mean',
    'categorical_fill_value': None,
    'numeric_fill_value': None},
    'DateTime Featurization Component': {'features_to_extract': ['year',
    'month',
    'day_of_week',
    'hour'],
    'encode_as_categories': False,
    'date_index': None},
    'One Hot Encoder': {'top_n': 10,
    'features_to_encode': None,
    'categories': None,
    'drop': 'if_binary',
    'handle_unknown': 'ignore',
    'handle_missing': 'error'},
    'SMOTENC Oversampler': {'sampling_ratio': 0.25,
    'k_neighbors_default': 5,
    'n_jobs': -1,
    'sampling_ratio_dict': None,
    'categorical_features': [3,
    10,
    11,
    12,
    13,
    14,
    15,
    16,
    17,
    18,
    19,
    20,
    21,
    22,
    23,
    24,
    25,
    26,
    27,
    28,
    29,
    30,
    31,
    32,
    33,
    34,
    35,
    36,

```

(continues on next page)

(continued from previous page)

```

37,
38,
39,
40,
41,
42,
43,
44,
45,
46,
47,
48,
49,
50,
51,
52,
53,
54,
55,
56,
57,
58,
59],
'k_neighbors': 5},
'Logistic Regression Classifier': {'penalty': 'l2',
'C': 1.0,
'n_jobs': -1,
'multi_class': 'auto',
'solver': 'lbfgs'}},
'mean_cv_score': 0.5519600330274397,
'standard_deviation_cv_score': 0.08269486722266915,
'high_variance_cv': False,
'training_time': 3.9947738647460938,
'cv_data': [{'all_objective_scores': OrderedDict([('Log Loss Binary',
0.6286458411705291),
('MCC Binary', 0.17518582316850065),
('Gini', 0.1652542372881356),
('AUC', 0.5826271186440678),
('Precision', 0.2857142857142857),
('F1', 0.26666666666666666),
('Balanced Accuracy Binary', 0.5826271186440678),
('Accuracy Binary', 0.835820895522388),
('# Training', 133),
('# Validation', 67)]),
'mean_cv_score': 0.6286458411705291,
'binary_classification_threshold': 0.4313366338786227},
{'all_objective_scores': OrderedDict([('Log Loss Binary',
0.46434334092355634),
('MCC Binary', 0.14283901342792224),
('Gini', 0.4067796610169492),
('AUC', 0.7033898305084746),
('Precision', 0.3333333333333333),
('F1', 0.18181818181818182),
('Balanced Accuracy Binary', 0.5455508474576272),
('Accuracy Binary', 0.8656716417910447),
('# Training', 133),
('# Validation', 67)]),

```

(continues on next page)



(continued from previous page)

```

    'mean_cv_score': 0.46434334092355634,
    'binary_classification_threshold': 0.7298626702409028},
    {'all_objective_scores': OrderedDict([('Log Loss Binary',
                                           0.5628909169882338),
                                           ('MCC Binary', 0.13468820490061437),
                                           ('Gini', 0.21549636803874073),
                                           ('AUC', 0.6077481840193704),
                                           ('Precision', 0.17647058823529413),
                                           ('F1', 0.25),
                                           ('Balanced Accuracy Binary', 0.5956416464891041),
                                           ('Accuracy Binary', 0.7272727272727273),
                                           ('# Training', 134),
                                           ('# Validation', 66)]),
     'mean_cv_score': 0.5628909169882338,
     'binary_classification_threshold': 0.16920770831511384}},
    'percent_better_than_baseline_all_objectives': {'Log Loss Binary': 86.
    ↪09820648846629,
     'MCC Binary': inf,
     'Gini': inf,
     'AUC': 13.125504439063763,
     'Precision': 26.517273576097107,
     'F1': 23.282828282828284,
     'Balanced Accuracy Binary': 7.4606537530266355,
     'Accuracy Binary': -7.5456053067993185},
    'percent_better_than_baseline': 86.09820648846629,
    'validation_score': 0.6286458411705291},
    6: {'id': 6,
        'pipeline_name': 'XGBoost Classifier w/ Imputer + DateTime Featurization Component_
    ↪+ One Hot Encoder + SMOTENC Oversampler',
        'pipeline_class': evalml.pipelines.binary_classification_pipeline.
    ↪BinaryClassificationPipeline,
        'pipeline_summary': 'XGBoost Classifier w/ Imputer + DateTime Featurization_
    ↪Component + One Hot Encoder + SMOTENC Oversampler',
        'parameters': {'Imputer': {'categorical_impute_strategy': 'most_frequent',
                                     'numeric_impute_strategy': 'mean',
                                     'categorical_fill_value': None,
                                     'numeric_fill_value': None},
                        'DateTime Featurization Component': {'features_to_extract': ['year',
                                           'month',
                                           'day_of_week',
                                           'hour'],
                                                              'encode_as_categories': False,
                                                              'date_index': None},
                        'One Hot Encoder': {'top_n': 10,
                                             'features_to_encode': None,
                                             'categories': None,
                                             'drop': 'if_binary',
                                             'handle_unknown': 'ignore',
                                             'handle_missing': 'error'},
                        'SMOTENC Oversampler': {'sampling_ratio': 0.25,
                                                 'k_neighbors_default': 5,
                                                 'n_jobs': -1,
                                                 'sampling_ratio_dict': None,
                                                 'categorical_features': [3,
                                           10,
                                           11,
                                           12,

```

(continues on next page)

(continued from previous page)

```
13,  
14,  
15,  
16,  
17,  
18,  
19,  
20,  
21,  
22,  
23,  
24,  
25,  
26,  
27,  
28,  
29,  
30,  
31,  
32,  
33,  
34,  
35,  
36,  
37,  
38,  
39,  
40,  
41,  
42,  
43,  
44,  
45,  
46,  
47,  
48,  
49,  
50,  
51,  
52,  
53,  
54,  
55,  
56,  
57,  
58,  
59],  
'k_neighbors': 5},  
'XGBoost Classifier': {'eta': 0.1,  
  'max_depth': 6,  
  'min_child_weight': 1,  
  'n_estimators': 100,  
  'n_jobs': -1}},  
'mean_cv_score': 0.2787074593034545,  
'standard_deviation_cv_score': 0.19072502137726743,  
'high_variance_cv': True,  
'training_time': 2.770864486694336,
```

(continues on next page)

(continued from previous page)

```

'cv_data': [{'all_objective_scores': OrderedDict([('Log Loss Binary',
0.20263786688400182),
('MCC Binary', 0.6842908506285673),
('Gini', 0.7838983050847457),
('AUC', 0.8919491525423728),
('Precision', 1.0),
('F1', 0.6666666666666666),
('Balanced Accuracy Binary', 0.75),
('Accuracy Binary', 0.9402985074626866),
('# Training', 133),
('# Validation', 67)]),
'mean_cv_score': 0.20263786688400182,
'binary_classification_threshold': 0.8503084678293262},
{'all_objective_scores': OrderedDict([('Log Loss Binary',
0.4957285583724021),
('MCC Binary', 0.20588632450454833),
('Gini', 0.2966101694915255),
('AUC', 0.6483050847457628),
('Precision', 0.5),
('F1', 0.2),
('Balanced Accuracy Binary', 0.5540254237288136),
('Accuracy Binary', 0.8805970149253731),
('# Training', 133),
('# Validation', 67)]),
'mean_cv_score': 0.4957285583724021,
'binary_classification_threshold': 0.9268544884142358},
{'all_objective_scores': OrderedDict([('Log Loss Binary',
0.13775595265395954),
('MCC Binary', 0.3600976668493281),
('Gini', 0.7772397094430992),
('AUC', 0.8886198547215496),
('Precision', 1.0),
('F1', 0.25),
('Balanced Accuracy Binary', 0.5714285714285714),
('Accuracy Binary', 0.9090909090909091),
('# Training', 134),
('# Validation', 66)]),
'mean_cv_score': 0.13775595265395954,
'binary_classification_threshold': 0.9457477876739043}],
'percent_better_than_baseline_all_objectives': {'Log Loss Binary': 92.
↪98040923704308,
'MCC Binary': inf,
'Gini': inf,
'AUC': 30.96246973365617,
'Precision': 83.33333333333334,
'F1': 37.22222222222222,
'Balanced Accuracy Binary': 12.51513317191283,
'Accuracy Binary': 2.4951002562942914},
'percent_better_than_baseline': 92.98040923704308,
'validation_score': 0.20263786688400182},
7: {'id': 7,
'pipeline_name': 'Extra Trees Classifier w/ Imputer + DateTime Featurization_
↪Component + One Hot Encoder + SMOTENC Oversampler',
'pipeline_class': evalml.pipelines.binary_classification_pipeline.
↪BinaryClassificationPipeline,
'pipeline_summary': 'Extra Trees Classifier w/ Imputer + DateTime Featurization_
↪Component + One Hot Encoder + SMOTENC Oversampler',

```

(continues on next page)

(continued from previous page)

```
'parameters': {'Imputer': {'categorical_impute_strategy': 'most_frequent',
    'numeric_impute_strategy': 'mean',
    'categorical_fill_value': None,
    'numeric_fill_value': None},
'Datetime Featurization Component': {'features_to_extract': ['year',
    'month',
    'day_of_week',
    'hour'],
    'encode_as_categories': False,
    'date_index': None},
'One Hot Encoder': {'top_n': 10,
    'features_to_encode': None,
    'categories': None,
    'drop': 'if_binary',
    'handle_unknown': 'ignore',
    'handle_missing': 'error'},
'SMOTENC Oversampler': {'sampling_ratio': 0.25,
    'k_neighbors_default': 5,
    'n_jobs': -1,
    'sampling_ratio_dict': None,
    'categorical_features': [3,
    10,
    11,
    12,
    13,
    14,
    15,
    16,
    17,
    18,
    19,
    20,
    21,
    22,
    23,
    24,
    25,
    26,
    27,
    28,
    29,
    30,
    31,
    32,
    33,
    34,
    35,
    36,
    37,
    38,
    39,
    40,
    41,
    42,
    43,
    44,
    45,
```

(continues on next page)

(continued from previous page)

```

46,
47,
48,
49,
50,
51,
52,
53,
54,
55,
56,
57,
58,
59],
'k_neighbors': 5},
'Extra Trees Classifier': {'n_estimators': 100,
'max_features': 'auto',
'max_depth': 6,
'min_samples_split': 2,
'min_weight_fraction_leaf': 0.0,
'n_jobs': -1}},
'mean_cv_score': 0.33828572013370833,
'standard_deviation_cv_score': 0.009380537483031018,
'high_variance_cv': False,
'training_time': 2.6858813762664795,
'cv_data': [{'all_objective_scores': OrderedDict([('Log Loss Binary',
0.32901507195041363),
('MCC Binary', 0.3624510999859961),
('Gini', 0.5677966101694916),
('AUC', 0.7838983050847458),
('Precision', 0.4),
('F1', 0.4444444444444445),
('Balanced Accuracy Binary', 0.6991525423728814),
('Accuracy Binary', 0.8507462686567164),
('# Training', 133),
('# Validation', 67)]),
'mean_cv_score': 0.32901507195041363,
'binary_classification_threshold': 0.24464061936922427},
{'all_objective_scores': OrderedDict([('Log Loss Binary',
0.3380696737832299),
('MCC Binary', -0.045325960909933655),
('Gini', 0.4279661016949152),
('AUC', 0.7139830508474576),
('Precision', 0.0),
('F1', 0.0),
('Balanced Accuracy Binary', 0.4915254237288136),
('Accuracy Binary', 0.8656716417910447),
('# Training', 133),
('# Validation', 67)]),
'mean_cv_score': 0.3380696737832299,
'binary_classification_threshold': 0.3074199530907704},
{'all_objective_scores': OrderedDict([('Log Loss Binary',
0.3477724146674814),
('MCC Binary', 0.27835560025568845),
('Gini', 0.43825665859564156),
('AUC', 0.7191283292978208),
('Precision', 0.22727272727272727),

```

(continues on next page)

(continued from previous page)

```

        ('F1', 0.3448275862068965),
        ('Balanced Accuracy Binary', 0.7130750605326877),
        ('Accuracy Binary', 0.7121212121212122),
        ('# Training', 134),
        ('# Validation', 66)]),
    'mean_cv_score': 0.3477724146674814,
    'binary_classification_threshold': 0.2199210188616361}},
    'percent_better_than_baseline_all_objectives': {'Log Loss Binary': 91.
↪47985733060223,
    'MCC Binary': inf,
    'Gini': inf,
    'AUC': 23.900322841000797,
    'Precision': 20.90909090909091,
    'F1': 26.309067688378036,
    'Balanced Accuracy Binary': 13.458434221146087,
    'Accuracy Binary': -7.553143374038884},
    'percent_better_than_baseline': 91.47985733060223,
    'validation_score': 0.32901507195041363},
8: {'id': 8,
    'pipeline_name': 'CatBoost Classifier w/ Imputer + DateTime Featurization_
↪Component + SMOTENC Oversampler',
    'pipeline_class': evalml.pipelines.binary_classification_pipeline.
↪BinaryClassificationPipeline,
    'pipeline_summary': 'CatBoost Classifier w/ Imputer + DateTime Featurization_
↪Component + SMOTENC Oversampler',
    'parameters': {'Imputer': {'categorical_impute_strategy': 'most_frequent',
    'numeric_impute_strategy': 'mean',
    'categorical_fill_value': None,
    'numeric_fill_value': None},
    'DateTime Featurization Component': {'features_to_extract': ['year',
    'month',
    'day_of_week',
    'hour'],
    'encode_as_categories': False,
    'date_index': None},
    'SMOTENC Oversampler': {'sampling_ratio': 0.25,
    'k_neighbors_default': 5,
    'n_jobs': -1,
    'sampling_ratio_dict': None,
    'categorical_features': [3, 4, 5, 6, 9, 10],
    'k_neighbors': 5},
    'CatBoost Classifier': {'n_estimators': 10,
    'eta': 0.03,
    'max_depth': 6,
    'bootstrap_type': None,
    'silent': True,
    'allow_writing_files': False,
    'n_jobs': -1}},
    'mean_cv_score': 0.6018191346985708,
    'standard_deviation_cv_score': 0.007246095254215931,
    'high_variance_cv': False,
    'training_time': 1.081974744796753,
    'cv_data': [{'all_objective_scores': OrderedDict([('Log Loss Binary',
    0.5936927052183711),
    ('MCC Binary', 0.7712055916006633),
    ('Gini', 0.7796610169491525),
    ('AUC', 0.8898305084745762),

```

(continues on next page)

(continued from previous page)

```

        ('Precision', 1.0),
        ('F1', 0.7692307692307693),
        ('Balanced Accuracy Binary', 0.8125),
        ('Accuracy Binary', 0.9552238805970149),
        ('# Training', 133),
        ('# Validation', 67)]),
    'mean_cv_score': 0.5936927052183711,
    'binary_classification_threshold': 0.498623875482405},
{'all_objective_scores': OrderedDict([('Log Loss Binary',
    0.6076076767585368),
    ('MCC Binary', 0.47636443708895493),
    ('Gini', 0.12076271186440679),
    ('AUC', 0.5603813559322034),
    ('Precision', 1.0),
    ('F1', 0.4),
    ('Balanced Accuracy Binary', 0.625),
    ('Accuracy Binary', 0.9104477611940298),
    ('# Training', 133),
    ('# Validation', 67)]),
    'mean_cv_score': 0.6076076767585368,
    'binary_classification_threshold': 0.5007469614241519},
{'all_objective_scores': OrderedDict([('Log Loss Binary',
    0.6041570221188047),
    ('MCC Binary', 0.7469064529073725),
    ('Gini', 0.8692493946731235),
    ('AUC', 0.9346246973365617),
    ('Precision', 0.8333333333333334),
    ('F1', 0.7692307692307692),
    ('Balanced Accuracy Binary', 0.8486682808716708),
    ('Accuracy Binary', 0.9545454545454546),
    ('# Training', 134),
    ('# Validation', 66)]),
    'mean_cv_score': 0.6041570221188047,
    'binary_classification_threshold': 0.47437131050216075}],
'percent_better_than_baseline_all_objectives': {'Log Loss Binary': 84.
↪84244358059615,
'MCC Binary': inf,
'Gini': inf,
'AUC': 29.49455205811139,
'Precision': 94.44444444444446,
'F1': 64.61538461538461,
'Balanced Accuracy Binary': 26.205609362389026,
'Accuracy Binary': 5.502789084878645},
'percent_better_than_baseline': 84.84244358059615,
'validation_score': 0.5936927052183711}},
'search_order': [0, 1, 2, 3, 4, 5, 6, 7, 8]}

```

### 4.1.12 Parallel AutoML

By default, all pipelines in an AutoML batch are evaluated in series. Pipelines can be evaluated in parallel to improve performance during AutoML search. This is accomplished by a futures style submission and evaluation of pipelines in a batch. As of this writing, the pipelines use a threaded model for concurrent evaluation. This is similar to the currently implemented `n_jobs` parameter in the estimators, which uses increased numbers of threads to train and evaluate estimators.

#### Parallelism with Concurrent Futures

The `EngineBase` class is robust and extensible enough to support futures-like implementations from a variety of libraries. The `CFEngine` extends the `EngineBase` to use the native Python `concurrent.futures` library. The `CFEngine` supports both thread- and process-level parallelism. The type of parallelism can be chosen using either the `ThreadPoolExecutor` or the `ProcessPoolExecutor`. If either executor is passed a `max_workers` parameter, it will set the number of processes and threads spawned. If not, the default number of processes will be equal to the number of processors available and the number of threads set to five times the number of processors available.

```
[26]: from concurrent.futures import ThreadPoolExecutor

from evalml.automl.engine import CFEngine
from evalml.automl.engine.cf_engine import CFClient

# Use thread-level parallelism
threaded_cf_engine = CFEngine(CFClient(ThreadPoolExecutor()))

automl_cf_threaded = AutoMLSearch(X_train=X, y_train=y,
                                  problem_type="binary",
                                  allowed_model_families=[ModelFamily.LINEAR_MODEL],
                                  engine=threaded_cf_engine)
automl_cf_threaded.search(show_iteration_plot = False)
```

Using default limit of max\_batches=1.

Generating pipelines to search over...  
2 pipelines ready for search.

```
*****
* Beginning pipeline search *
*****
```

Optimizing for Log Loss Binary.  
Lower score is better.

Using CFEngine to train and score pipelines.  
Searching up to 1 batches for a total of 3 pipelines.  
Allowed model families: linear\_model

Evaluating Baseline Pipeline: Mode Baseline Binary Classification Pipeline  
Mode Baseline Binary Classification Pipeline:  
Starting cross validation  
Finished cross validation - mean Log Loss Binary: 12.868

```
*****
* Evaluating Batch Number 1 *
*****
```

(continues on next page)



(continued from previous page)

```

Logistic Regression Classifier w/ Imputer + Standard Scaler:
    Starting cross validation
    Finished cross validation - mean Log Loss Binary: 0.077
    High coefficient of variation (cv >= 0.2) within cross validation scores.
    Logistic Regression Classifier w/ Imputer + Standard Scaler may not perform_
    ↪as estimated on unseen data.
Elastic Net Classifier w/ Imputer + Standard Scaler:
    Starting cross validation
    Finished cross validation - mean Log Loss Binary: 0.077

Search finished after 00:04
Best pipeline: Logistic Regression Classifier w/ Imputer + Standard Scaler
Best pipeline Log Loss Binary: 0.076807

```

Note: the cell demonstrating process-level parallelism is commented out due to incompatibility with our ReadTheDocs build. It can be run successfully locally.

```

from concurrent.futures import ProcessPoolExecutor

# Repeat the process but using process-level parallelism
process_cf_engine = CFEngine(CFClient(ProcessPoolExecutor()))

automl_cf_process = AutoMLSearch(X_train=X, y_train=y,
                                problem_type="binary",
                                engine=process_cf_engine)
automl_cf_process.search(show_iteration_plot = False)

```

## Parallelism with Dask

Thread or process level parallelism can be explicitly invoked for the DaskEngine (as well as the CFEngine). The processes can be set to True and the number of processes set using n\_workers. If processes is set to False, then the resulting parallelism will be threaded and n\_workers will represent the threads used. Examples of both follow.

```

[27]: from dask.distributed import Client, LocalCluster

from evalml.automl.engine import DaskEngine

dask_engine_p2 = DaskEngine(Client(LocalCluster(processes=True, n_workers = 2)))
automl_dask_p2 = AutoMLSearch(X_train=X, y_train=y,
                              problem_type="binary",
                              allowed_model_families=[ModelFamily.LINEAR_MODEL],
                              engine=dask_engine_p2)
automl_dask_p2.search(show_iteration_plot = False)
del(dask_engine_p2)

```

Using default limit of max\_batches=1.

Generating pipelines to search over...  
2 pipelines ready for search.

```

*****
* Beginning pipeline search *
*****

```

(continues on next page)

(continued from previous page)

```
Optimizing for Log Loss Binary.
Lower score is better.

Using DaskEngine to train and score pipelines.
Searching up to 1 batches for a total of 3 pipelines.
Allowed model families: linear_model

Evaluating Baseline Pipeline: Mode Baseline Binary Classification Pipeline
Mode Baseline Binary Classification Pipeline:
    Starting cross validation
    Finished cross validation - mean Log Loss Binary: 12.868

*****
* Evaluating Batch Number 1 *
*****

Elastic Net Classifier w/ Imputer + Standard Scaler:
    Starting cross validation
    Finished cross validation - mean Log Loss Binary: 0.077
Logistic Regression Classifier w/ Imputer + Standard Scaler:
    Starting cross validation
    Finished cross validation - mean Log Loss Binary: 0.077
    High coefficient of variation (cv >= 0.2) within cross validation scores.
    Logistic Regression Classifier w/ Imputer + Standard Scaler may not perform
↳as estimated on unseen data.

Search finished after 00:09
Best pipeline: Logistic Regression Classifier w/ Imputer + Standard Scaler
Best pipeline Log Loss Binary: 0.076807
```

```
[28]: dask_engine_t4 = DaskEngine(Client(LocalCluster(processes=False, n_workers = 4)))

automl_dask_t4 = AutoMLSearch(X_train=X, y_train=y,
                             problem_type="binary",
                             allowed_model_families=[ModelFamily.LINEAR_MODEL],
                             engine=dask_engine_t4)
automl_dask_t4.search(show_iteration_plot = False)
```

```
Using default limit of max_batches=1.

Generating pipelines to search over...
2 pipelines ready for search.

*****
* Beginning pipeline search *
*****

Optimizing for Log Loss Binary.
Lower score is better.

Using DaskEngine to train and score pipelines.
Searching up to 1 batches for a total of 3 pipelines.
Allowed model families: linear_model

Evaluating Baseline Pipeline: Mode Baseline Binary Classification Pipeline
```

```
/home/docs/checkouts/readthedocs.org/user_builds/feature-labs-inc-evalml/envs/v0.30.2/
↳ lib/python3.8/site-packages/distributed/node.py:160: UserWarning:
```

```
Port 8787 is already in use.
Perhaps you already have a cluster running?
Hosting the HTTP server on port 38285 instead
```

```
Mode Baseline Binary Classification Pipeline:
  Starting cross validation
  Finished cross validation - mean Log Loss Binary: 12.868
```

```
*****
* Evaluating Batch Number 1 *
*****
```

```
Logistic Regression Classifier w/ Imputer + Standard Scaler:
  Starting cross validation
  Finished cross validation - mean Log Loss Binary: 0.077
  High coefficient of variation (cv >= 0.2) within cross validation scores.
  Logistic Regression Classifier w/ Imputer + Standard Scaler may not perform_
↳ as estimated on unseen data.
```

```
Elastic Net Classifier w/ Imputer + Standard Scaler:
  Starting cross validation
  Finished cross validation - mean Log Loss Binary: 0.077
```

```
Search finished after 00:05
Best pipeline: Logistic Regression Classifier w/ Imputer + Standard Scaler
Best pipeline Log Loss Binary: 0.076807
```

As we can see, a significant performance gain can result from simply using something other than the default SequentialEngine, ranging from a 100% speed up with multiple processes to 250% speedup with multiple threads!

```
[29]: print("Sequential search duration: %s" % str(automl.search_duration))
      print("Concurrent futures (threaded) search duration: %s" % str(automl_cf_threaded.
      ↳ search_duration))
      print("Dask (two processes) search duration: %s" % str(automl_dask_p2.search_
      ↳ duration))
      print("Dask (four threads) search duration: %s" % str(automl_dask_t4.search_duration))
```

```
Sequential search duration: 22.410727500915527
Concurrent futures (threaded) search duration: 4.713129043579102
Dask (two processes) search duration: 9.325072050094604
Dask (four threads) search duration: 5.496379375457764
```

## 4.2 Objectives

### 4.2.1 Overview

One of the key choices to make when training an ML model is what metric to choose by which to measure the efficacy of the model at learning the signal. Such metrics are useful for comparing how well the trained models generalize to new similar data.

This choice of metric is a key component of AutoML because it defines the cost function the AutoML search will seek to optimize. In EvalML, these metrics are called **objectives**. AutoML will seek to minimize (or maximize)

the objective score as it explores more pipelines and parameters and will use the feedback from scoring pipelines to tune the available hyperparameters and continue the search. Therefore, it is critical to have an objective function that represents how the model will be applied in the intended domain of use.

EvalML supports a variety of objectives from traditional supervised ML including [mean squared error](#) for regression problems and [cross entropy](#) or [area under the ROC curve](#) for classification problems. EvalML also allows the user to define a custom objective using their domain expertise, so that AutoML can search for models which provide the most value for the user's problem.

## 4.2.2 Core Objectives

Use the `get_core_objectives` method to get a list of which objectives are included with EvalML for each problem type:

```
[1]: from evalml.objectives import get_core_objectives
from evalml.problem_types import ProblemTypes

for objective in get_core_objectives(ProblemTypes.BINARY):
    print(objective.name)
```

```
MCC Binary
Log Loss Binary
Gini
AUC
Precision
F1
Balanced Accuracy Binary
Accuracy Binary
```

EvalML defines a base objective class for each problem type: `RegressionObjective`, `BinaryClassificationObjective` and `MulticlassClassificationObjective`. All EvalML objectives are a subclass of one of these.

### Binary Classification Objectives and Thresholds

All binary classification objectives have a `threshold` property. Some binary classification objectives like log loss and AUC are unaffected by the choice of binary classification threshold, because they score based on predicted probabilities or examine a range of threshold values. These metrics are defined with `score_needs_proba` set to `False`. For all other binary classification objectives, we can compute the optimal binary classification threshold from the predicted probabilities and the target.

```
[2]: from evalml.pipelines import BinaryClassificationPipeline
from evalml.demos import load_fraud
from evalml.objectives import F1

X, y = load_fraud(n_rows=100)
X.ww.init(logical_types={"provider": "Categorical", "region": "Categorical"})
objective = F1()
pipeline = BinaryClassificationPipeline(component_graph=['Simple Imputer', 'DateTime_
↳Featurization Component', 'One Hot Encoder', 'Random Forest Classifier'])
pipeline.fit(X, y)
print(pipeline.threshold)
print(pipeline.score(X, y, objectives=[objective]))

y_pred_proba = pipeline.predict_proba(X)[True]
```

(continues on next page)

(continued from previous page)

```

pipeline.threshold = objective.optimize_threshold(y_pred_proba, y)
print(pipeline.threshold)
print(pipeline.score(X, y, objectives=[objective]))

```

```

                Number of Features
Boolean                      1
Categorical                   6
Numeric                       5

Number of training examples: 100
Targets
False      91.00%
True       9.00%
Name: fraud, dtype: object
None
OrderedDict([('F1', 1.0)])
0.5202757772593112
OrderedDict([('F1', 1.0)])

```

### 4.2.3 Custom Objectives

Often times, the objective function is very specific to the use-case or business problem. To get the right objective to optimize requires thinking through the decisions or actions that will be taken using the model and assigning a cost/benefit to doing that correctly or incorrectly based on known outcomes in the training data.

Once you have determined the objective for your business, you can provide that to EvalML to optimize by defining a custom objective function.

#### Defining a Custom Objective Function

To create a custom objective class, we must define several elements:

- `name`: The printable name of this objective.
- `objective_function`: This function takes the predictions, true labels, and an optional reference to the inputs, and returns a score of how well the model performed.
- `greater_is_better`: True if a higher `objective_function` value represents a better solution, and otherwise False.
- `score_needs_proba`: Only for classification objectives. True if the objective is intended to function with predicted probabilities as opposed to predicted values (example: cross entropy for classifiers).
- `decision_function`: Only for binary classification objectives. This function takes predicted probabilities that were output from the model and a binary classification threshold, and returns predicted values.
- `perfect_score`: The score achieved by a perfect model on this objective.

## Example: Fraud Detection

To give a concrete example, let's look at how the *fraud detection* objective function is built.

```
[3]: from evalml.objectives.binary_classification_objective import
      ↪ BinaryClassificationObjective
      import pandas as pd

class FraudCost(BinaryClassificationObjective):
    """Score the percentage of money lost of the total transaction amount process due
    ↪ to fraud"""
    name = "Fraud Cost"
    greater_is_better = False
    score_needs_proba = False
    perfect_score = 0.0

    def __init__(self, retry_percentage=.5, interchange_fee=.02,
                  fraud_payout_percentage=1.0, amount_col='amount'):
        """Create instance of FraudCost

        Arguments:
            retry_percentage (float): What percentage of customers that will retry a
            ↪ transaction if it
                is declined. Between 0 and 1. Defaults to .5

            interchange_fee (float): How much of each successful transaction you can
            ↪ collect.
                Between 0 and 1. Defaults to .02

            fraud_payout_percentage (float): Percentage of fraud you will not be able
            ↪ to collect.
                Between 0 and 1. Defaults to 1.0

            amount_col (str): Name of column in data that contains the amount.
            ↪ Defaults to "amount"
        """
        self.retry_percentage = retry_percentage
        self.interchange_fee = interchange_fee
        self.fraud_payout_percentage = fraud_payout_percentage
        self.amount_col = amount_col

    def decision_function(self, ypred_proba, threshold=0.0, X=None):
        """Determine if a transaction is fraud given predicted probabilities,
        ↪ threshold, and dataframe with transaction amount

        Arguments:
            ypred_proba (pd.Series): Predicted probabilities
            X (pd.DataFrame): Dataframe containing transaction amount
            threshold (float): Dollar threshold to determine if transaction is
            ↪ fraud

        Returns:
            pd.Series: Series of predicted fraud labels using X and threshold
        """
        if not isinstance(X, pd.DataFrame):
            X = pd.DataFrame(X)
```

(continues on next page)

(continued from previous page)

```

    if not isinstance(ypred_proba, pd.Series):
        ypred_proba = pd.Series(ypred_proba)

    transformed_probs = (ypred_proba.values * X[self.amount_col])
    return transformed_probs > threshold

def objective_function(self, y_true, y_predicted, X):
    """Calculate amount lost to fraud per transaction given predictions, true_
    ↪ values, and dataframe with transaction amount

    Arguments:
        y_predicted (pd.Series): predicted fraud labels
        y_true (pd.Series): true fraud labels
        X (pd.DataFrame): dataframe with transaction amounts

    Returns:
        float: amount lost to fraud per transaction
    """
    if not isinstance(X, pd.DataFrame):
        X = pd.DataFrame(X)

    if not isinstance(y_predicted, pd.Series):
        y_predicted = pd.Series(y_predicted)

    if not isinstance(y_true, pd.Series):
        y_true = pd.Series(y_true)

    # extract transaction using the amount columns in users data
    try:
        transaction_amount = X[self.amount_col]
    except KeyError:
        ↪ raise ValueError("`{}` is not a valid column in X.".format(self.amount_
        ↪ col))

    # amount paid if transaction is fraud
    fraud_cost = transaction_amount * self.fraud_payout_percentage

    # money made from interchange fees on transaction
    interchange_cost = transaction_amount * (1 - self.retry_percentage) * self.
    ↪ interchange_fee

    # calculate cost of missing fraudulent transactions
    false_negatives = (y_true & ~y_predicted) * fraud_cost

    # calculate money lost from fees
    false_positives = (~y_true & y_predicted) * interchange_cost

    loss = false_negatives.sum() + false_positives.sum()

    loss_per_total_processed = loss / transaction_amount.sum()

    return loss_per_total_processed

```

## 4.3 Components

Components are the lowest level of building blocks in EvalML. Each component represents a fundamental operation to be applied to data.

All components accept parameters as keyword arguments to their `__init__` methods. These parameters can be used to configure behavior.

Each component class definition must include a human-readable `name` for the component. Additionally, each component class may expose parameters for AutoML search by defining a `hyperparameter_ranges` attribute containing the parameters in question.

EvalML splits components into two categories: **transformers** and **estimators**.

### 4.3.1 Transformers

Transformers subclass the `Transformer` class, and define a `fit` method to learn information from training data and a `transform` method to apply a learned transformation to new data.

For example, an *imputer* is configured with the desired impute strategy to follow, for instance the mean value. The imputers `fit` method would learn the mean from the training data, and the `transform` method would fill the learned mean value in for any missing values in new data.

All transformers can execute `fit` and `transform` separately or in one step by calling `fit_transform`. Defining a custom `fit_transform` method can facilitate useful performance optimizations in some cases.

```
[1]: import numpy as np
import pandas as pd
from evalml.pipelines.components import SimpleImputer

X = pd.DataFrame([[1, 2, 3], [1, np.nan, 3]])
display(X)
```

```
   0    1    2
0  1    2    3
1  1  NaN    3
```

```
[2]: import woodwork as ww
imp = SimpleImputer(impute_strategy="mean")

X.ww.init()
X = imp.fit_transform(X)
display(X)
```

```
   0    1    2
0  1    2    3
1  1    2    3
```

Below is a list of all transformers included with EvalML:

```
[3]: from evalml.pipelines.components.utils import all_components, Estimator, Transformer
for component in all_components():
    if issubclass(component, Transformer):
        print(f"Transformer: {component.name}")
```

```
Transformer: URL Featurizer
Transformer: Email Featurizer
Transformer: DFS Transformer
```

(continues on next page)



(continued from previous page)

```

Transformer: Delayed Feature Transformer
Transformer: Text Featurization Component
Transformer: LSA Transformer
Transformer: Drop Null Columns Transformer
Transformer: DateTime Featurization Component
Transformer: PCA Transformer
Transformer: Linear Discriminant Analysis Transformer
Transformer: Select Columns By Type Transformer
Transformer: Select Columns Transformer
Transformer: Drop Columns Transformer
Transformer: Undersampler
Transformer: SMOTEN Oversampler
Transformer: SMOTENC Oversampler
Transformer: SMOTE Oversampler
Transformer: Standard Scaler
Transformer: Target Imputer
Transformer: Imputer
Transformer: Per Column Imputer
Transformer: Simple Imputer
Transformer: RF Regressor Select From Model
Transformer: RF Classifier Select From Model
Transformer: Target Encoder
Transformer: One Hot Encoder
Transformer: Log Transformer
Transformer: Polynomial Detrender

```

### 4.3.2 Estimators

Each estimator wraps an ML algorithm. Estimators subclass the `Estimator` class, and define a `fit` method to learn information from training data and a `predict` method for generating predictions from new data. Classification estimators should also define a `predict_proba` method for generating predicted probabilities.

Estimator classes each define a `model_family` attribute indicating what type of model is used.

Here's an example of using the *LogisticRegressionClassifier* estimator to fit and predict on a simple dataset:

```

[4]: from evalml.pipelines.components import LogisticRegressionClassifier

clf = LogisticRegressionClassifier()

X = X
y = [1, 0]

clf.fit(X, y)
clf.predict(X)

[4]: 0      0
     1      0
dtype: int64

```

Below is a list of all estimators included with EvalML:

```

[5]: from evalml.pipelines.components.utils import all_components, Estimator, Transformer
     for component in all_components():
         if issubclass(component, Estimator):
             print(f"Estimator: {component.name}")

```

```
Estimator: Sklearn Stacked Ensemble Regressor
Estimator: Sklearn Stacked Ensemble Classifier
Estimator: ARIMA Regressor
Estimator: SVM Regressor
Estimator: Time Series Baseline Estimator
Estimator: Decision Tree Regressor
Estimator: Baseline Regressor
Estimator: Extra Trees Regressor
Estimator: XGBoost Regressor
Estimator: CatBoost Regressor
Estimator: Random Forest Regressor
Estimator: LightGBM Regressor
Estimator: Linear Regressor
Estimator: Elastic Net Regressor
Estimator: SVM Classifier
Estimator: KNN Classifier
Estimator: Decision Tree Classifier
Estimator: LightGBM Classifier
Estimator: Baseline Classifier
Estimator: Extra Trees Classifier
Estimator: Elastic Net Classifier
Estimator: CatBoost Classifier
Estimator: XGBoost Classifier
Estimator: Random Forest Classifier
Estimator: Logistic Regression Classifier
```

### 4.3.3 Defining Custom Components

EvalML allows you to easily create your own custom components by following the steps below.

#### Custom Transformers

Your transformer must inherit from the correct subclass. In this case *Transformer* for components that transform data. Next we will use EvalML's *DropNullColumns* as an example.

```
[6]: from evalml.pipelines.components import Transformer
    from evalml.utils import (
        infer_feature_types,
    )

    class DropNullColumns(Transformer):
        """Transformer to drop features whose percentage of NaN values exceeds a
        ↪specified threshold"""
        name = "Drop Null Columns Transformer"
        hyperparameter_ranges = {}

        def __init__(self, pct_null_threshold=1.0, random_seed=0, **kwargs):
            """Initializes an transformer to drop features whose percentage of NaN values
            ↪exceeds a specified threshold.

            Arguments:
                pct_null_threshold(float): The percentage of NaN values in an input
            ↪feature to drop.
                Must be a value between [0, 1] inclusive. If equal to 0.0, will drop
            ↪columns with any null values.
```

(continues on next page)

(continued from previous page)

```

        If equal to 1.0, will drop columns with all null values. Defaults to 0.95.
    """
    if pct_null_threshold < 0 or pct_null_threshold > 1:
        raise ValueError("pct_null_threshold must be a float between 0 and 1, inclusive.")
    parameters = {"pct_null_threshold": pct_null_threshold}
    parameters.update(kwargs)

    self._cols_to_drop = None
    super().__init__(parameters=parameters,
                     component_obj=None,
                     random_seed=random_seed)

    def fit(self, X, y=None):
        """Fits DropNullColumns component to data

        Arguments:
            X (pd.DataFrame): The input training data of shape [n_samples, n_features]
            y (pd.Series, optional): The target training data of length [n_samples]

        Returns:
            self
        """
        pct_null_threshold = self.parameters["pct_null_threshold"]
        X_t = infer_feature_types(X)
        percent_null = X_t.isnull().mean()
        if pct_null_threshold == 0.0:
            null_cols = percent_null[percent_null > 0]
        else:
            null_cols = percent_null[percent_null >= pct_null_threshold]
        self._cols_to_drop = list(null_cols.index)
        return self

    def transform(self, X, y=None):
        """Transforms data X by dropping columns that exceed the threshold of null values.

        Arguments:
            X (pd.DataFrame): Data to transform
            y (pd.Series, optional): Ignored.

        Returns:
            pd.DataFrame: Transformed X
        """
        X_t = infer_feature_types(X)
        return X_t.drop(self._cols_to_drop)

```

## Required fields

- `name`: A human-readable name.
- `modifies_features`: A boolean that specifies whether this component modifies (subsets or transforms) the features variable during `transform`.
- `modifies_target`: A boolean that specifies whether this component modifies (subsets or transforms) the target variable during `transform`.

## Required methods

Likewise, there are select methods you need to override as `Transformer` is an abstract base class:

- `__init__()`: The `__init__()` method of your transformer will need to call `super().__init__()` and pass three parameters in: a `parameters` dictionary holding the parameters to the component, the `component_obj`, and the `random_seed` value. You can see that `component_obj` is set to `None` above and we will discuss `component_obj` in depth later on.
- `fit()`: The `fit()` method is responsible for fitting your component on training data. It should return the component object.
- `transform()`: After fitting a component, the `transform()` method will take in new data and transform accordingly. It should return a pandas dataframe with woodwork initialized. Note: a component must call `fit()` before `transform()`.

You can also call or override `fit_transform()` that combines `fit()` and `transform()` into one method.

## Custom Estimators

Your estimator must inherit from the correct subclass. In this case *Estimator* for components that predict new target values. Next we will use EvalML's *BaselineRegressor* as an example.

```
[7]: import numpy as np
import pandas as pd

from evalml.model_family import ModelFamily
from evalml.pipelines.components.estimators import Estimator
from evalml.problem_types import ProblemTypes

class BaselineRegressor(Estimator):
    """Regressor that predicts using the specified strategy.

    This is useful as a simple baseline regressor to compare with other regressors.
    """
    name = "Baseline Regressor"
    hyperparameter_ranges = {}
    model_family = ModelFamily.BASELINE
    supported_problem_types = [ProblemTypes.REGRESSION, ProblemTypes.TIME_SERIES_
↪REGRESSION]

    def __init__(self, strategy="mean", random_seed=0, **kwargs):
        """Baseline regressor that uses a simple strategy to make predictions.

        Arguments:
```

(continues on next page)

(continued from previous page)

```

        strategy (str): Method used to predict. Valid options are "mean", "median
→ ". Defaults to "mean".
        random_seed (int): Seed for the random number generator. Defaults to 0.

    """
    if strategy not in ["mean", "median"]:
        raise ValueError("'strategy' parameter must equal either 'mean' or 'median
→ ")
    parameters = {"strategy": strategy}
    parameters.update(kwargs)

    self._prediction_value = None
    self._num_features = None
    super().__init__(parameters=parameters,
                     component_obj=None,
                     random_seed=random_seed)

    def fit(self, X, y=None):
        if y is None:
            raise ValueError("Cannot fit Baseline regressor if y is None")
        X = infer_feature_types(X)
        y = infer_feature_types(y)

        if self.parameters["strategy"] == "mean":
            self._prediction_value = y.mean()
        elif self.parameters["strategy"] == "median":
            self._prediction_value = y.median()
        self._num_features = X.shape[1]
        return self

    def predict(self, X):
        X = infer_feature_types(X)
        predictions = pd.Series([self._prediction_value] * len(X))
        return infer_feature_types(predictions)

    @property
    def feature_importance(self):
        """Returns importance associated with each feature. Since baseline regressors
→ do not use input features to calculate predictions, returns an array of zeroes.

        Returns:
            np.ndarray (float): An array of zeroes

    """
    return np.zeros(self._num_features)

```

## Required fields

- `name`: A human-readable name.
- `model_family` - EvalML *model\_family* that this component belongs to
- `supported_problem_types` - list of EvalML *problem\_types* that this component supports
- `modifies_features`: A boolean that specifies whether the return value from `predict` or `predict_proba` should be used as features.
- `modifies_target`: A boolean that specifies whether the return value from `predict` or `predict_proba` should be used as the target variable.

Model families and problem types include:

```
[8]: from evalml.model_family import ModelFamily
     from evalml.problem_types import ProblemTypes

     print("Model Families:\n", [m.value for m in ModelFamily])
     print("Problem Types:\n", [p.value for p in ProblemTypes])

Model Families:
['k_neighbors', 'random_forest', 'svm', 'xgboost', 'lightgbm', 'linear_model',
↪ 'catboost', 'extra_trees', 'ensemble', 'decision_tree', 'arima', 'baseline',
↪ 'prophet', 'none']
Problem Types:
['binary', 'multiclass', 'regression', 'time series regression', 'time series binary
↪ ', 'time series multiclass']
```

## Required methods

- `__init__()` - the `__init__()` method of your estimator will need to call `super().__init__()` and pass three parameters in: a `parameters` dictionary holding the parameters to the component, the `component_obj`, and the `random_seed` value.
- `fit()` - the `fit()` method is responsible for fitting your component on training data.
- `predict()` - after fitting a component, the `predict()` method will take in new data and predict new target values. Note: a component must call `fit()` before `predict()`.
- `feature_importance` - `feature_importance` is a [Python property](#) that returns a list of importances associated with each feature.

If your estimator handles classification problems it also requires an additional method:

- `predict_proba()` - this method predicts probability estimates for classification labels

## Components Wrapping Third-Party Objects

The `component_obj` parameter is used for wrapping third-party objects and using them in component implementation. If you're using a `component_obj` you will need to define `__init__()` and pass in the relevant object that has also implemented the required methods mentioned above. However, if the `component_obj` does not follow EvalML component conventions, you may need to override methods as needed. Below is an example of EvalML's *LinearRegressor*.

```
[9]: from sklearn.linear_model import LinearRegression as SKLinearRegression

from evalml.model_family import ModelFamily
from evalml.pipelines.components.estimators import Estimator
from evalml.problem_types import ProblemTypes

class LinearRegressor(Estimator):
    """Linear Regressor."""
    name = "Linear Regressor"
    model_family = ModelFamily.LINEAR_MODEL
    supported_problem_types = [ProblemTypes.REGRESSION]

    def __init__(self, fit_intercept=True, normalize=False, n_jobs=-1, random_seed=0,
↳ **kwargs):
        parameters = {
            'fit_intercept': fit_intercept,
            'normalize': normalize,
            'n_jobs': n_jobs
        }
        parameters.update(kwargs)
        linear_regressor = SKLinearRegression(**parameters)
        super().__init__(parameters=parameters,
                          component_obj=linear_regressor,
                          random_seed=random_seed)

    @property
    def feature_importance(self):
        return self._component_obj.coef_
```

## Hyperparameter Ranges for AutoML

`hyperparameter_ranges` is a dictionary mapping the parameter name (str) to an allowed range (SkOpt Space) for that parameter. Both lists and `skopt.space.Categorical` values are accepted for categorical spaces.

AutoML will perform a search over the allowed ranges for each parameter to select models which produce optimal performance within those ranges. AutoML gets the allowed ranges for each component from the component's `hyperparameter_ranges` class attribute. Any component parameter you add an entry for in `hyperparameter_ranges` will be included in the AutoML search. If parameters are omitted, AutoML will use the default value in all pipelines.

### 4.3.4 Generate Component Code

Once you have a component defined in EvalML, you can generate string Python code to recreate this component, which can then be saved and run elsewhere with EvalML. `generate_component_code` requires a component instance as the input. This method works for custom components as well, although it won't return the code required to define the custom component.

```
[10]: from evalml.pipelines.components import LogisticRegressionClassifier
from evalml.pipelines.components.utils import generate_component_code

lr = LogisticRegressionClassifier(C=5)
code = generate_component_code(lr)
print(code)
```

```
from evalml.pipelines.components.estimators.classifiers.logistic_regression_
↳classifier import LogisticRegressionClassifier

logisticRegressionClassifier = LogisticRegressionClassifier(**{'penalty': 'l2', 'C':
↳5, 'n_jobs': -1, 'multi_class': 'auto', 'solver': 'lbfgs'})
```

```
[11]: # this string can then be copy and pasted into a separate window and executed as
↳python code
exec(code)
```

```
[12]: # We can also do this for custom components
from evalml.pipelines.components.utils import generate_component_code

myDropNull = DropNullColumns()
print(generate_component_code(myDropNull))

dropNullColumnsTransformer = DropNullColumns(**{'pct_null_threshold': 1.0})
```

## Expectations for Custom Classification Components

EvalML expects the following from custom classification component implementations:

- Classification targets will range from 0 to n-1 and are integers.
- For classification estimators, the order of `predict_proba`'s columns must match the order of the target, and the column names must be integers ranging from 0 to n-1

## 4.4 Pipelines

EvalML pipelines represent a sequence of operations to be applied to data, where each operation is either a data transformation or an ML modeling algorithm.

A pipeline holds a combination of one or more components, which will be applied to new input data in sequence.

Each component and pipeline supports a set of parameters which configure its behavior. The AutoML search process seeks to find the combination of pipeline structure and pipeline parameters which perform the best on the data.

### 4.4.1 Defining a Pipeline Instance

Pipeline instances can be instantiated using any of the following classes:

- `RegressionPipeline`
- `BinaryClassificationPipeline`
- `MulticlassClassificationPipeline`
- `TimeSeriesRegressionPipeline`
- `TimeSeriesBinaryClassificationPipeline`
- `TimeSeriesMulticlassClassificationPipeline`

The class you want to use will depend on your problem type. The only required parameter input for instantiating a pipeline instance is `component_graph`, which is either a list or a dictionary containing a sequence of components to be fit and evaluated.



A `component_graph` list is the default representation, which represents a linear order of transforming components with an estimator as the final component. A `component_graph` dictionary is used to represent a non-linear graph of components, where the key is a unique name for each component and the value is a list with the component's class as the first element and any parents of the component as the following element(s). For either `component_graph` format, each component can be provided as a reference to the component class for custom components, and as either a string name or as a reference to the component class for components defined in EvalML.

```
[1]: from evalml.pipelines import MulticlassClassificationPipeline

component_graph_as_list = ['Imputer', 'Random Forest Classifier']
MulticlassClassificationPipeline(component_graph=component_graph_as_list)

[1]: pipeline = MulticlassClassificationPipeline(component_graph={'Imputer': ['Imputer', 'X'
→, 'y'], 'Random Forest Classifier': ['Random Forest Classifier', 'Imputer.x', 'y']}
→, parameters={'Imputer':{'categorical_impute_strategy': 'most_frequent', 'numeric_
→impute_strategy': 'mean', 'categorical_fill_value': None, 'numeric_fill_value': _
→None}, 'Random Forest Classifier':{'n_estimators': 100, 'max_depth': 6, 'n_jobs': -
→1}}, random_seed=0)

[2]: component_graph_as_dict = {
    'Imputer': ['Imputer', 'X', 'y'],
    'Encoder': ['One Hot Encoder', 'Imputer.x', 'y'],
    'Random Forest Clf': ['Random Forest Classifier', 'Encoder.x', 'y'],
    'Elastic Net Clf': ['Elastic Net Classifier', 'Encoder.x', 'y'],
    'Final Estimator': ['Logistic Regression Classifier', 'Random Forest Clf.x',
→ 'Elastic Net Clf.x', 'y']
}

MulticlassClassificationPipeline(component_graph=component_graph_as_dict)

[2]: pipeline = MulticlassClassificationPipeline(component_graph={'Imputer': ['Imputer', 'X'
→, 'y'], 'Encoder': ['One Hot Encoder', 'Imputer.x', 'y'], 'Random Forest Clf': [
→ 'Random Forest Classifier', 'Encoder.x', 'y'], 'Elastic Net Clf': ['Elastic Net _
→Classifier', 'Encoder.x', 'y'], 'Final Estimator': ['Logistic Regression Classifier
→, 'Random Forest Clf.x', 'Elastic Net Clf.x', 'y']}, parameters={'Imputer':{'
→ 'categorical_impute_strategy': 'most_frequent', 'numeric_impute_strategy': 'mean',
→ 'categorical_fill_value': None, 'numeric_fill_value': None}, 'Encoder':{'top_n': 10,
→ 'features_to_encode': None, 'categories': None, 'drop': 'if_binary', 'handle_
→unknown': 'ignore', 'handle_missing': 'error'}, 'Random Forest Clf':{'n_estimators':
→ 100, 'max_depth': 6, 'n_jobs': -1}, 'Elastic Net Clf':{'penalty': 'elasticnet', 'C
→': 1.0, 'l1_ratio': 0.15, 'n_jobs': -1, 'multi_class': 'auto', 'solver': 'saga'},
→ 'Final Estimator':{'penalty': 'l2', 'C': 1.0, 'n_jobs': -1, 'multi_class': 'auto',
→ 'solver': 'lbfgs'}}}, random_seed=0)
```

If you're using your own *custom components* you can refer to them like so:

```
[3]: from evalml.pipelines.components import Transformer

class NewTransformer(Transformer):
    name = 'New Transformer'
    hyperparameter_ranges = {
        "parameter_1": ['a', 'b', 'c']
    }

    def __init__(self, parameter_1=1, random_seed=0):
        parameters = {"parameter_1": parameter_1}
        super().__init__(parameters=parameters,
                         random_seed=random_seed)
```

(continues on next page)

(continued from previous page)

```
MulticlassClassificationPipeline([NewTransformer, 'Random Forest Classifier'])
```

```
[3]: pipeline = MulticlassClassificationPipeline(component_graph={'New Transformer':  
↪[NewTransformer, 'X', 'y'], 'Random Forest Classifier': ['Random Forest Classifier',  
↪ 'New Transformer.x', 'y']}, parameters={'New Transformer':{'parameter_1': 1},  
↪ 'Random Forest Classifier':{'n_estimators': 100, 'max_depth': 6, 'n_jobs': -1}},  
↪ random_seed=0)
```

## 4.4.2 Pipeline Usage

All pipelines define the following methods:

- `fit` fits each component on the provided training data, in order.
- `predict` computes the predictions of the component graph on the provided data.
- `score` computes the value of *an objective* on the provided data.

```
[4]: from evalml.demos import load_wine  
X, y = load_wine()  
  
pipeline = MulticlassClassificationPipeline(['Imputer', 'Random Forest Classifier'])  
pipeline.fit(X, y)  
print(pipeline.predict(X))  
print(pipeline.score(X, y, objectives=['log loss multiclass']))
```

```

      Number of Features
Numeric                13

Number of training examples: 178
Targets
class_1    39.89%
class_0    33.15%
class_2    26.97%
Name: target, dtype: object
0      class_0
1      class_0
2      class_0
3      class_0
4      class_0
...
173    class_2
174    class_2
175    class_2
176    class_2
177    class_2
Length: 178, dtype: category
Categories (3, object): ['class_0', 'class_1', 'class_2']
OrderedDict([('Log Loss Multiclass', 0.04132737017536148)])

```

### 4.4.3 Custom Name

By default, a pipeline's name is created using the component graph that makes up the pipeline. E.g. A pipeline with an imputer, one-hot encoder, and logistic regression classifier will have the name 'Logistic Regression Classifier w/ Imputer + One Hot Encoder'.

If you'd like to override the pipeline's name attribute, you can set the `custom_name` parameter when initializing a pipeline, like so:

```
[5]: component_graph = ['Imputer', 'One Hot Encoder', 'Logistic Regression Classifier']
pipeline = MulticlassClassificationPipeline(component_graph)
print("Pipeline with default name:", pipeline.name)

pipeline_with_name = MulticlassClassificationPipeline(component_graph, custom_name=
↳ "My cool custom pipeline")
print("Pipeline with custom name:", pipeline_with_name.name)

Pipeline with default name: Logistic Regression Classifier w/ Imputer + One Hot_
↳ Encoder
Pipeline with custom name: My cool custom pipeline
```

### 4.4.4 Pipeline Parameters

You can also pass in custom parameters by using the `parameters` parameter, which will then be used when instantiating each component in `component_graph`. The parameters dictionary needs to be in the format of a two-layered dictionary where the key-value pairs are the component name and corresponding component parameters dictionary. The component parameters dictionary consists of (parameter name, parameter values) key-value pairs.

An example will be shown below. The API reference for component parameters can also be found [\[here\]](#) (`../api_reference.rst#components`).

```
[6]: parameters = {
    'Imputer': {
        'categorical_impute_strategy': "most_frequent",
        'numeric_impute_strategy': "median"
    },
    'Logistic Regression Classifier': {
        'penalty': 'l2',
        'C': 1.0,
    }
}

component_graph = ['Imputer', 'One Hot Encoder', 'Standard Scaler', 'Logistic_
↳ Regression Classifier']
MulticlassClassificationPipeline(component_graph=component_graph,
↳ parameters=parameters)

[6]: pipeline = MulticlassClassificationPipeline(component_graph={'Imputer': ['Imputer', 'X
↳ ', 'y'], 'One Hot Encoder': ['One Hot Encoder', 'Imputer.x', 'y'], 'Standard Scaler
↳ ': ['Standard Scaler', 'One Hot Encoder.x', 'y'], 'Logistic Regression Classifier':
↳ ['Logistic Regression Classifier', 'Standard Scaler.x', 'y']}, parameters={'Imputer
↳ ': {'categorical_impute_strategy': 'most_frequent', 'numeric_impute_strategy':
↳ 'median', 'categorical_fill_value': None, 'numeric_fill_value': None}, 'One Hot_
↳ Encoder': {'top_n': 10, 'features_to_encode': None, 'categories': None, 'drop': 'if_
↳ binary', 'handle_unknown': 'ignore', 'handle_missing': 'error'}, 'Logistic_
↳ Regression Classifier': {'penalty': 'l2', 'C': 1.0, 'n_jobs': -1, 'multi_class':
↳ 'auto', 'solver': 'lbfgs'}}}, random_seed=0)
```

### 4.4.5 Pipeline Description

You can call `.graph()` to see each component and its parameters. Each component takes in data and feeds it to the next.

```
[7]: component_graph = ['Imputer', 'One Hot Encoder', 'Standard Scaler', 'Logistic_
↳Regression Classifier']
pipeline = MulticlassClassificationPipeline(component_graph=component_graph,
↳parameters=parameters)
pipeline.graph()
```

```
[7]:
```

```
[8]: component_graph_as_dict = {
    'Imputer': ['Imputer', 'X', 'y'],
    'Encoder': ['One Hot Encoder', 'Imputer.x', 'y'],
    'Random Forest Clf': ['Random Forest Classifier', 'Encoder.x', 'y'],
    'Elastic Net Clf': ['Elastic Net Classifier', 'Encoder.x', 'y'],
    'Final Estimator': ['Logistic Regression Classifier', 'Random Forest Clf.x',
↳'Elastic Net Clf.x', 'y']
}

nonlinear_pipeline = MulticlassClassificationPipeline(component_graph=component_graph_
↳as_dict)
nonlinear_pipeline.graph()
```

```
[8]:
```

You can see a textual representation of the pipeline by calling `.describe()`:

```
[9]: pipeline.describe()

*****
* Logistic Regression Classifier w/ Imputer + One Hot Encoder + Standard Scaler *
*****

Problem Type: multiclass
Model Family: Linear

Pipeline Steps
=====
1. Imputer
    * categorical_impute_strategy : most_frequent
    * numeric_impute_strategy : median
    * categorical_fill_value : None
    * numeric_fill_value : None
2. One Hot Encoder
    * top_n : 10
    * features_to_encode : None
    * categories : None
    * drop : if_binary
    * handle_unknown : ignore
    * handle_missing : error
3. Standard Scaler
4. Logistic Regression Classifier
    * penalty : l2
    * C : 1.0
    * n_jobs : -1
    * multi_class : auto
    * solver : lbfgs
```

```
[10]: nonlinear_pipeline.describe()
```

```
*****
* Logistic Regression Classifier w/ Imputer + One Hot Encoder + Random Forest_
↳Classifier + Elastic Net Classifier *
*****

Problem Type: multiclass
Model Family: Linear

Pipeline Steps
=====
1. Imputer
    * categorical_impute_strategy : most_frequent
    * numeric_impute_strategy : mean
    * categorical_fill_value : None
    * numeric_fill_value : None
2. One Hot Encoder
    * top_n : 10
    * features_to_encode : None
    * categories : None
    * drop : if_binary
    * handle_unknown : ignore
    * handle_missing : error
3. Random Forest Classifier
    * n_estimators : 100
    * max_depth : 6
    * n_jobs : -1
4. Elastic Net Classifier
    * penalty : elasticnet
    * C : 1.0
    * l1_ratio : 0.15
    * n_jobs : -1
    * multi_class : auto
    * solver : saga
5. Logistic Regression Classifier
    * penalty : l2
    * C : 1.0
    * n_jobs : -1
    * multi_class : auto
    * solver : lbfgs
```

#### 4.4.6 Component Graph

You can use `pipeline.get_component(name)` and provide the component name to access any component (API reference [here](#)):

```
[11]: pipeline.get_component('Imputer')
```

```
[11]: Imputer(categorical_impute_strategy='most_frequent', numeric_impute_strategy='median',
↳ categorical_fill_value=None, numeric_fill_value=None)
```

```
[12]: nonlinear_pipeline.get_component('Elastic Net Clf')
```

```
[12]: ElasticNetClassifier(penalty='elasticnet', C=1.0, l1_ratio=0.15, n_jobs=-1, multi_
↳ class='auto', solver='saga')
```

Alternatively, you can index directly into the pipeline to get a component

```
[13]: first_component = pipeline[0]
      print(first_component.name)

Imputer

[14]: nonlinear_pipeline['Final Estimator']

[14]: LogisticRegressionClassifier(penalty='l2', C=1.0, n_jobs=-1, multi_class='auto',
      ↪solver='lbfgs')
```

## 4.4.7 Pipeline Estimator

EvalML enforces that the last component of a linear pipeline is an estimator. You can access this estimator directly by using `pipeline.estimator`.

```
[15]: pipeline.estimator

[15]: LogisticRegressionClassifier(penalty='l2', C=1.0, n_jobs=-1, multi_class='auto',
      ↪solver='lbfgs')
```

## 4.4.8 Input Feature Names

After a pipeline is fitted, you can access a pipeline's `input_feature_names` attribute to obtain a dictionary containing a list of feature names passed to each component of the pipeline. This could be especially useful for debugging where a feature might have been dropped or detecting unexpected behavior.

```
[16]: pipeline = MulticlassClassificationPipeline(['Imputer', 'Random Forest Classifier'])
      pipeline.fit(X, y)
      pipeline.input_feature_names

[16]: {'Imputer': ['alcohol',
                  'malic_acid',
                  'ash',
                  'alcalinity_of_ash',
                  'magnesium',
                  'total_phenols',
                  'flavanoids',
                  'nonflavanoid_phenols',
                  'proanthocyanins',
                  'color_intensity',
                  'hue',
                  'od280/od315_of_diluted_wines',
                  'proline'],
      'Random Forest Classifier': ['alcohol',
                                   'malic_acid',
                                   'ash',
                                   'alcalinity_of_ash',
                                   'magnesium',
                                   'total_phenols',
                                   'flavanoids',
                                   'nonflavanoid_phenols',
                                   'proanthocyanins',
                                   'color_intensity',
                                   'hue',
```

(continues on next page)

(continued from previous page)

```
'od280/od315_of_diluted_wines',
'proline']}]}
```

### 4.4.9 Saving and Loading Pipelines

You can save and load trained or untrained pipeline instances using the Python `pickle` format, like so:

```
[17]: import pickle

pipeline_to_pickle = MulticlassClassificationPipeline(['Imputer', 'Random Forest_
↳Classifier'])

with open("pipeline.pkl", 'wb') as f:
    pickle.dump(pipeline_to_pickle, f)

pickled_pipeline = None
with open('pipeline.pkl', 'rb') as f:
    pickled_pipeline = pickle.load(f)

assert pickled_pipeline == pipeline_to_pickle
pickled_pipeline.fit(X, y)

[17]: pipeline = MulticlassClassificationPipeline(component_graph={'Imputer': ['Imputer', 'X
↳', 'y'], 'Random Forest Classifier': ['Random Forest Classifier', 'Imputer.x', 'y']}]
↳, parameters={'Imputer':{'categorical_impute_strategy': 'most_frequent', 'numeric_
↳impute_strategy': 'mean', 'categorical_fill_value': None, 'numeric_fill_value':_
↳None}, 'Random Forest Classifier':{'n_estimators': 100, 'max_depth': 6, 'n_jobs': -
↳1}}, random_seed=0)
```

### 4.4.10 Generate Code

Once you have instantiated a pipeline, you can generate string Python code to recreate this pipeline, which can then be saved and run elsewhere with EvalML. `generate_pipeline_code` requires a pipeline instance as the input. It can also handle custom components, but it won't return the code required to define the component. Note that any external libraries used in creating the pipeline instance will also need to be imported to execute the returned code.

Code generation is not yet supported for nonlinear pipelines.

```
[18]: from evalml.pipelines.utils import generate_pipeline_code
from evalml.pipelines import MulticlassClassificationPipeline
import pandas as pd
from evalml.utils import infer_feature_types
from skopt.space import Integer

class MyDropNullColumns(Transformer):
    """Transformer to drop features whose percentage of NaN values exceeds a_
↳specified threshold"""
    name = "My Drop Null Columns Transformer"
    hyperparameter_ranges = {}

    def __init__(self, pct_null_threshold=1.0, random_seed=0, **kwargs):
        """Initializes an transformer to drop features whose percentage of NaN values_
↳exceeds a specified threshold.
```

(continues on next page)

(continued from previous page)

```

    Arguments:
        pct_null_threshold(float): The percentage of NaN values in an input
        ↪ feature to drop.
            Must be a value between [0, 1] inclusive. If equal to 0.0, will drop
        ↪ columns with any null values.
            If equal to 1.0, will drop columns with all null values. Defaults to
        ↪ 0.95.
    """
    if pct_null_threshold < 0 or pct_null_threshold > 1:
        raise ValueError("pct_null_threshold must be a float between 0 and 1,
        ↪ inclusive.")
    parameters = {"pct_null_threshold": pct_null_threshold}
    parameters.update(kwargs)

    self._cols_to_drop = None
    super().__init__(parameters=parameters,
                     component_obj=None,
                     random_seed=random_seed)

    def fit(self, X, y=None):
        pct_null_threshold = self.parameters["pct_null_threshold"]
        X = infer_feature_types(X)
        percent_null = X.isnull().mean()
        if pct_null_threshold == 0.0:
            null_cols = percent_null[percent_null > 0]
        else:
            null_cols = percent_null[percent_null >= pct_null_threshold]
        self._cols_to_drop = list(null_cols.index)
        return self

    def transform(self, X, y=None):
        """Transforms data X by dropping columns that exceed the threshold of null
        ↪ values.
        Arguments:
            X (pd.DataFrame): Data to transform
            y (pd.Series, optional): Targets
        Returns:
            pd.DataFrame: Transformed X
        """

        X = infer_feature_types(X)
        return X.drop(columns=self._cols_to_drop)

pipeline_instance = MulticlassClassificationPipeline(['Imputer', MyDropNullColumns,
        ↪ 'DateTime Featurization',
        ↪ 'Text Featurization Component',
        ↪ 'One Hot Encoder', 'Random
        ↪ Forest Classifier'],
        ↪ custom_name="Pipeline with
        ↪ Custom Component",
        ↪ random_seed=20)

code = generate_pipeline_code(pipeline_instance)
print(code)

```

(continues on next page)



(continued from previous page)

```
# This string can then be pasted into a separate window and run, although since the_
↳ pipeline has custom component `MyDropNullColumns`,
# the code for that component must also be included
from evalml.demos import load_fraud
X, y = load_fraud(1000)
exec(code)
pipeline.fit(X, y)
```

```
from evalml.pipelines.multiclass_classification_pipeline import_
↳ MulticlassClassificationPipeline
pipeline = MulticlassClassificationPipeline(component_graph={'Imputer': ['Imputer', 'X
↳ ', 'y'], 'My Drop Null Columns Transformer': [MyDropNullColumns, 'Imputer.x', 'y'],
↳ 'DateTime Featurization Component': ['DateTime Featurization Component', 'My Drop_
↳ Null Columns Transformer.x', 'y'], 'Text Featurization Component': ['Text_
↳ Featurization Component', 'DateTime Featurization Component.x', 'y'], 'One Hot_
↳ Encoder': ['One Hot Encoder', 'Text Featurization Component.x', 'y'], 'Random_
↳ Forest Classifier': ['Random Forest Classifier', 'One Hot Encoder.x', 'y']},_
↳ parameters={'Imputer':{'categorical_impute_strategy': 'most_frequent', 'numeric_
↳ impute_strategy': 'mean', 'categorical_fill_value': None, 'numeric_fill_value':_
↳ None}, 'My Drop Null Columns Transformer':{'pct_null_threshold': 1.0}, 'DateTime_
↳ Featurization Component':{'features_to_extract': ['year', 'month', 'day_of_week',
↳ 'hour'], 'encode_as_categories': False, 'date_index': None}, 'One Hot Encoder':{'
↳ top_n': 10, 'features_to_encode': None, 'categories': None, 'drop': 'if_binary',
↳ 'handle_unknown': 'ignore', 'handle_missing': 'error'}, 'Random Forest Classifier':{'
↳ n_estimators': 100, 'max_depth': 6, 'n_jobs': -1}}, custom_name='Pipeline with_
↳ Custom Component', random_seed=20)
```

	Number of Features
Boolean	1
Categorical	6
Numeric	5

Number of training examples: 1000

Targets

False 85.90%

True 14.10%

Name: fraud, dtype: object

```
[18]: pipeline = MulticlassClassificationPipeline(component_graph={'Imputer': ['Imputer', 'X
↳ ', 'y'], 'My Drop Null Columns Transformer': [MyDropNullColumns, 'Imputer.x', 'y'],
↳ 'DateTime Featurization Component': ['DateTime Featurization Component', 'My Drop_
↳ Null Columns Transformer.x', 'y'], 'Text Featurization Component': ['Text_
↳ Featurization Component', 'DateTime Featurization Component.x', 'y'], 'One Hot_
↳ Encoder': ['One Hot Encoder', 'Text Featurization Component.x', 'y'], 'Random_
↳ Forest Classifier': ['Random Forest Classifier', 'One Hot Encoder.x', 'y']},_
↳ parameters={'Imputer':{'categorical_impute_strategy': 'most_frequent', 'numeric_
↳ impute_strategy': 'mean', 'categorical_fill_value': None, 'numeric_fill_value':_
↳ None}, 'My Drop Null Columns Transformer':{'pct_null_threshold': 1.0}, 'DateTime_
↳ Featurization Component':{'features_to_extract': ['year', 'month', 'day_of_week',
↳ 'hour'], 'encode_as_categories': False, 'date_index': None}, 'One Hot Encoder':{'
↳ top_n': 10, 'features_to_encode': None, 'categories': None, 'drop': 'if_binary',
↳ 'handle_unknown': 'ignore', 'handle_missing': 'error'}, 'Random Forest Classifier':{'
↳ n_estimators': 100, 'max_depth': 6, 'n_jobs': -1}}, custom_name='Pipeline with_
↳ Custom Component', random_seed=20)
```

## 4.5 Model Understanding

Simply examining a model’s performance metrics is not enough to select a model and promote it for use in a production setting. While developing an ML algorithm, it is important to understand how the model behaves on the data, to examine the key factors influencing its predictions and to consider where it may be deficient. Determination of what “success” may mean for an ML project depends first and foremost on the user’s domain expertise.

EvalML includes a variety of tools for understanding models, from graphing utilities to methods for explaining predictions.

\*\* Graphing methods on Jupyter Notebook and Jupyter Lab require `ipywidgets` to be installed.

\*\* If graphing on Jupyter Lab, `jupyterlab-plotly` required. To download this, make sure you have `npm` installed.

### 4.5.1 Graphing Utilities

First, let’s train a pipeline on some data.

```
[1]: import evalml
from evalml.pipelines import BinaryClassificationPipeline
X, y = evalml.demos.load_breast_cancer()

X_train, X_holdout, y_train, y_holdout = evalml.preprocessing.split_data(X, y,
    ↪problem_type='binary',
    test_size=0.
    ↪2, random_seed=0)

pipeline_binary = BinaryClassificationPipeline(['Simple Imputer', 'Random Forest_
    ↪Classifier'])
pipeline_binary.fit(X_train, y_train)
print(pipeline_binary.score(X_holdout, y_holdout, objectives=['log loss binary']))
```

Number of Features	
Numeric	30
Number of training examples:	569
Targets	
benign	62.74%
malignant	37.26%
Name: target, dtype: object	
OrderedDict([('Log Loss Binary', 0.1686746297113362)])	

### Feature Importance

We can get the importance associated with each feature of the resulting pipeline

```
[2]: pipeline_binary.feature_importance
```

	feature	importance
0	mean concave points	0.138857
1	worst perimeter	0.137780
2	worst concave points	0.117782
3	worst radius	0.100584
4	mean concavity	0.086402
5	worst area	0.072027

(continues on next page)

(continued from previous page)

6	mean perimeter	0.046500
7	worst concavity	0.043408
8	mean radius	0.037664
9	mean area	0.033683
10	radius error	0.025036
11	area error	0.019324
12	worst texture	0.014754
13	worst compactness	0.014462
14	mean texture	0.013856
15	worst smoothness	0.013710
16	worst symmetry	0.011395
17	perimeter error	0.010284
18	mean compactness	0.008162
19	mean smoothness	0.008154
20	worst fractal dimension	0.007034
21	fractal dimension error	0.005502
22	compactness error	0.004953
23	smoothness error	0.004728
24	texture error	0.004384
25	symmetry error	0.004250
26	mean fractal dimension	0.004164
27	concavity error	0.004089
28	mean symmetry	0.003997
29	concave points error	0.003076

We can also create a bar plot of the feature importances

```
[3]: pipeline_binary.graph_feature_importance()
```

Data type cannot be displayed: application/vnd.plotly.v1+json, text/html

## Permutation Importance

We can also compute and plot the permutation importance of the pipeline.

```
[4]: from evalml.model_understanding import calculate_permutation_importance
calculate_permutation_importance(pipeline_binary, X_holdout, y_holdout, 'log loss_
↳binary')
```

	feature	importance
0	worst perimeter	0.063657
1	worst area	0.045759
2	worst radius	0.041926
3	mean concave points	0.029325
4	worst concave points	0.021045
5	worst concavity	0.010105
6	worst texture	0.010044
7	mean texture	0.006178
8	mean symmetry	0.005857
9	mean area	0.004745
10	worst smoothness	0.003190
11	area error	0.003113
12	mean perimeter	0.002478

(continues on next page)

(continued from previous page)

```

13 mean fractal dimension    0.001981
14 compactness error        0.001968
15 concavity error          0.001947
16 texture error            0.000291
17 smoothness error         -0.000206
18 mean smoothness          -0.000745
19 fractal dimension error   -0.000835
20 worst compactness        -0.002392
21 mean concavity           -0.003188
22 mean compactness         -0.005377
23 radius error             -0.006229
24 mean radius              -0.006870
25 worst fractal dimension   -0.007415
26 symmetry error           -0.008175
27 perimeter error          -0.008980
28 concave points error      -0.010415
29 worst symmetry           -0.018645

```

```
[5]: from evalml.model_understanding import graph_permutation_importance
graph_permutation_importance(pipeline_binary, X_holdout, y_holdout, 'log loss binary')
```

Data type cannot be displayed: application/vnd.plotly.v1+json, text/html

## Partial Dependence Plots

We can calculate the one-way [partial dependence plots](#) for a feature.

```
[6]: from evalml.model_understanding.graphs import partial_dependence
partial_dependence(pipeline_binary, X_holdout, features='mean radius', grid_
↳ resolution=5)
```

```
[6]: feature_values  partial_dependence  class_label
0          9.69092          0.392453    malignant
1         12.40459          0.395962    malignant
2         15.11826          0.417396    malignant
3         17.83193          0.429542    malignant
4         20.54560          0.429717    malignant
```

```
[7]: from evalml.model_understanding.graphs import graph_partial_dependence
graph_partial_dependence(pipeline_binary, X_holdout, features='mean radius', grid_
↳ resolution=5)
```

Data type cannot be displayed: application/vnd.plotly.v1+json, text/html

You can also compute the partial dependence for a categorical feature. We will demonstrate this on the fraud dataset.

```
[8]: X_fraud, y_fraud = evalml.demos.load_fraud(100, verbose=False)
X_fraud.ww.init(logical_types={"provider": "Categorical", "region": "Categorical"})

fraud_pipeline = BinaryClassificationPipeline(["DateTime Featurization Component",
↳ "One Hot Encoder", "Random Forest Classifier"])
```

(continues on next page)

(continued from previous page)

```

fraud_pipeline.fit(X_fraud, y_fraud)

graph_partial_dependence(fraud_pipeline, X_fraud, features='provider')

```

Data type cannot be displayed: application/vnd.plotly.v1+json, text/html

Two-way partial dependence plots are also possible and invoke the same API.

```

[9]: partial_dependence(pipeline_binary, X_holdout, features=('worst_perimeter', 'worst_
    ↪radius'), grid_resolution=5)

```

```

[9]:      10.6876  14.404924999999999  18.12225  21.839575  25.5569  \
69.140700  0.279038      0.282898  0.435179  0.435355  0.435355
94.334275  0.304335      0.308194  0.458283  0.458458  0.458458
119.527850  0.464455      0.468314  0.612137  0.616932  0.616932
144.721425  0.483437      0.487297  0.631120  0.635915  0.635915
169.915000  0.483437      0.487297  0.631120  0.635915  0.635915

      class_label
69.140700    malignant
94.334275    malignant
119.527850    malignant
144.721425    malignant
169.915000    malignant

```

```

[10]: graph_partial_dependence(pipeline_binary, X_holdout, features=('worst_perimeter',
    ↪'worst_radius'), grid_resolution=5)

```

Data type cannot be displayed: application/vnd.plotly.v1+json, text/html

## Confusion Matrix

For binary or multiclass classification, we can view a [confusion matrix](#) of the classifier's predictions. In the DataFrame output of `confusion_matrix()`, the column header represents the predicted labels while row header represents the actual labels.

```

[11]: from evalml.model_understanding.graphs import confusion_matrix
y_pred = pipeline_binary.predict(X_holdout)
confusion_matrix(y_holdout, y_pred)

```

```

[11]:      benign  malignant
benign    0.930556  0.069444
malignant  0.023810  0.976190

```

```

[12]: from evalml.model_understanding.graphs import graph_confusion_matrix
y_pred = pipeline_binary.predict(X_holdout)
graph_confusion_matrix(y_holdout, y_pred)

```

Data type cannot be displayed: application/vnd.plotly.v1+json, text/html

## Precision-Recall Curve

For binary classification, we can view the precision-recall curve of the pipeline.

```
[13]: from evalml.model_understanding.graphs import graph_precision_recall_curve
# get the predicted probabilities associated with the "true" label
import woodwork as ww
y_encoded = y_holdout.ww.map({'benign': 0, 'malignant': 1})
y_pred_proba = pipeline_binary.predict_proba(X_holdout)["malignant"]
graph_precision_recall_curve(y_encoded, y_pred_proba)
```

Data type cannot be displayed: application/vnd.plotly.v1+json, text/html

## ROC Curve

For binary and multiclass classification, we can view the Receiver Operating Characteristic (ROC) curve of the pipeline.

```
[14]: from evalml.model_understanding.graphs import graph_roc_curve
# get the predicted probabilities associated with the "malignant" label
y_pred_proba = pipeline_binary.predict_proba(X_holdout)["malignant"]
graph_roc_curve(y_encoded, y_pred_proba)
```

Data type cannot be displayed: application/vnd.plotly.v1+json, text/html

The ROC curve can also be generated for multiclass classification problems. For multiclass problems, the graph will show a one-vs-many ROC curve for each class.

```
[15]: from evalml.pipelines import MulticlassClassificationPipeline
X_multi, y_multi = evalml.demos.load_wine()

pipeline_multi = MulticlassClassificationPipeline(['Simple Imputer', 'Random Forest_
↪Classifier'])
pipeline_multi.fit(X_multi, y_multi)

y_pred_proba = pipeline_multi.predict_proba(X_multi)
graph_roc_curve(y_multi, y_pred_proba)
```

```
      Number of Features
Numeric                13
```

```
Number of training examples: 178
```

```
Targets
```

```
class_1    39.89%
```

```
class_0    33.15%
```

```
class_2    26.97%
```

```
Name: target, dtype: object
```

Data type cannot be displayed: application/vnd.plotly.v1+json, text/html

## Binary Objective Score vs. Threshold Graph

*Some binary classification objectives* (objectives that have `score_needs_proba` set to `False`) are sensitive to a decision threshold. For those objectives, we can obtain and graph the scores for thresholds from zero to one, calculated at evenly-spaced intervals determined by `steps`.

```
[16]: from evalml.model_understanding.graphs import binary_objective_vs_threshold
      binary_objective_vs_threshold(pipeline_binary, X_holdout, y_holdout, 'f1', steps=10)
```

```
[16]: threshold    score
      0         0.0  0.538462
      1         0.1  0.811881
      2         0.2  0.891304
      3         0.3  0.901099
      4         0.4  0.931818
      5         0.5  0.931818
      6         0.6  0.941176
      7         0.7  0.951220
      8         0.8  0.936709
      9         0.9  0.923077
     10         1.0  0.000000
```

```
[17]: from evalml.model_understanding.graphs import graph_binary_objective_vs_threshold
      graph_binary_objective_vs_threshold(pipeline_binary, X_holdout, y_holdout, 'f1',
      ↪ steps=100)
```

Data type cannot be displayed: application/vnd.plotly.v1+json, text/html

## Predicted Vs Actual Values Graph for Regression Problems

We can also create a scatterplot comparing predicted vs actual values for regression problems. We can specify an `outlier_threshold` to color values differently if the absolute difference between the actual and predicted values are outside of a given threshold.

```
[18]: from evalml.model_understanding.graphs import graph_prediction_vs_actual
      from evalml.pipelines import RegressionPipeline

      X_regress, y_regress = evalml.demos.load_diabetes()
      X_train, X_test, y_train, y_test = evalml.preprocessing.split_data(X_regress, y_
      ↪ regress, problem_type='regression')

      pipeline_regress = RegressionPipeline(['One Hot Encoder', 'Linear Regressor'])
      pipeline_regress.fit(X_train, y_train)

      y_pred = pipeline_regress.predict(X_test)
      graph_prediction_vs_actual(y_test, y_pred, outlier_threshold=50)
```

```
      Number of Features
Numeric                10
```

```
Number of training examples: 442
Targets
200.0    1.36%
72.0     1.36%
```

(continues on next page)

(continued from previous page)

```

90.0      1.13%
178.0     1.13%
71.0      1.13%
...
73.0      0.23%
222.0     0.23%
86.0      0.23%
79.0      0.23%
57.0      0.23%
Name: target, Length: 214, dtype: object

```

Data type cannot be displayed: application/vnd.plotly.v1+json, text/html

Now let's train a decision tree on some data.

```

[19]: pipeline_dt = BinaryClassificationPipeline(['Simple Imputer', 'Decision Tree_
↪Classifier'])
pipeline_dt.fit(X_train, y_train)

[19]: pipeline = BinaryClassificationPipeline(component_graph={'Simple Imputer': ['Simple_
↪Imputer', 'X', 'y'], 'Decision Tree Classifier': ['Decision Tree Classifier',
↪'Simple Imputer.x', 'y']}, parameters={'Simple Imputer':{'impute_strategy': 'most_
↪frequent', 'fill_value': None}, 'Decision Tree Classifier':{'criterion': 'gini',
↪'max_features': 'auto', 'max_depth': 6, 'min_samples_split': 2, 'min_weight_
↪fraction_leaf': 0.0}}, random_seed=0)

```

## Tree Visualization

We can visualize the structure of the Decision Tree that was fit to that data, and save it if necessary.

```

[20]: from evalml.model_understanding.graphs import visualize_decision_tree

visualize_decision_tree(pipeline_dt.estimator, max_depth=2, rotate=False, filled=True,
↪ filepath=None)

[20]:

```

## 4.5.2 Explaining Predictions

We can explain why the model made certain predictions with the [explain\\_predictions](#) function. This will use the [Shapley Additive Explanations \(SHAP\)](#) algorithm to identify the top features that explain the predicted value.

This function can explain both classification and regression models - all you need to do is provide the pipeline, the input features, and a list of rows corresponding to the indices of the input features you want to explain. The function will return a table that you can print summarizing the top 3 most positive and negative contributing features to the predicted value.

In the example below, we explain the prediction for the third data point in the data set. We see that the `worst concave points` feature increased the estimated probability that the tumor is malignant by 20% while the `worst radius` feature decreased the probability the tumor is malignant by 5%.

```

[21]: from evalml.model_understanding.prediction_explanations import explain_predictions

table = explain_predictions(pipeline=pipeline_binary, input_features=X_holdout,
↪ y=None, indices_to_explain=[3],

```

(continues on next page)



(continued from previous page)

```

top_k_features=6, include_shap_values=True)
print(table)
Random Forest Classifier w/ Simple Imputer

{'Simple Imputer': {'impute_strategy': 'most_frequent', 'fill_value': None}, 'Random_
↳Forest Classifier': {'n_estimators': 100, 'max_depth': 6, 'n_jobs': -1}}

1 of 1

    SHAP Value      Feature Name      Feature Value      Contribution to Prediction
    =====
    ↳0.02          worst concavity          0.18              -
    ↳0.03          mean concavity          0.04              -
    ↳0.03          worst area             599.50            -
    ↳0.03          worst radius           14.04             -
    ↳0.05          mean concave points     0.03              -
    ↳0.05          worst perimeter        92.80             -
    ↳0.06

```

The interpretation of the table is the same for regression problems - but the SHAP value now corresponds to the change in the estimated value of the dependent variable rather than a change in probability. For multiclass classification problems, a table will be output for each possible class.

Below is an example of how you would explain three predictions with *explain\_predictions*.

```

[22]: from evalml.model_understanding.prediction_explanations import explain_predictions

report = explain_predictions(pipeline=pipeline_binary,
                             input_features=X_holdout, y=y_holdout, indices_to_
    ↳explain=[0, 4, 9], include_shap_values=True,
                             output_format='text')
print(report)
Random Forest Classifier w/ Simple Imputer

{'Simple Imputer': {'impute_strategy': 'most_frequent', 'fill_value': None}, 'Random_
    ↳Forest Classifier': {'n_estimators': 100, 'max_depth': 6, 'n_jobs': -1}}

1 of 3

    SHAP Value      Feature Name      Feature Value      Contribution to Prediction
    =====
    ↳-0.04          worst perimeter        101.20            -
    ↳-0.05          worst concave points     0.06              -
    ↳-0.05

```

(continues on next page)

(continued from previous page)

↪-0.05	mean concave points	0.01	-	↪
2 of 3				
↪SHAP Value	Feature Name	Feature Value	Contribution to Prediction	↪
↪	=====			
↪0.05	worst radius	11.94	-	↪
↪0.06	worst perimeter	80.78	-	↪
↪0.06	mean concave points	0.02	-	↪
3 of 3				
↪SHAP Value	Feature Name	Feature Value	Contribution to Prediction	↪
↪	=====			
↪-0.05	worst concave points	0.10	-	↪
↪-0.06	worst perimeter	99.21	-	↪
↪-0.08	mean concave points	0.03	-	↪

## Explaining Best and Worst Predictions

When debugging machine learning models, it is often useful to analyze the best and worst predictions the model made. The `explain_predictions_best_worst` function can help us with this.

This function will display the output of `explain_predictions` for the best 2 and worst 2 predictions. By default, the best and worst predictions are determined by the absolute error for regression problems and `cross entropy` for classification problems.

We can specify our own ranking function by passing in a function to the `metric` parameter. This function will be called on `y_true` and `y_pred`. By convention, lower scores are better.

At the top of each table, we can see the predicted probabilities, target value, error, and row index for that prediction. For a regression problem, we would see the predicted value instead of predicted probabilities.

```
[23]: from evalml.model_understanding.prediction_explanations import explain_predictions_
      ↪ best_worst

report = explain_predictions_best_worst(pipeline=pipeline_binary, input_features=X_
      ↪ holdout, y_true=y_holdout,
                                     include_shap_values=True, top_k_features=6,
      ↪ num_to_explain=2)
```

(continues on next page)

(continued from previous page)

```
print(report)
```

```
Random Forest Classifier w/ Simple Imputer
```

```
{'Simple Imputer': {'impute_strategy': 'most_frequent', 'fill_value': None}, 'Random_
↳Forest Classifier': {'n_estimators': 100, 'max_depth': 6, 'n_jobs': -1}}
```

```
Best 1 of 2
```

```
Predicted Probabilities: [benign: 1.0, malignant: 0.0]
```

```
Predicted Value: benign
```

```
Target Value: benign
```

```
Cross Entropy: 0.0
```

```
Index ID: 502
```

	Feature Name	Feature Value	Contribution to Prediction
↳SHAP Value			
↳=====			
	mean concavity	0.06	-
↳-0.03	worst area	552.00	-
↳-0.03	worst concave points	0.08	-
↳-0.05	worst radius	13.57	-
↳-0.05	mean concave points	0.03	-
↳-0.05	worst perimeter	86.67	-
↳-0.06			

```
Best 2 of 2
```

```
Predicted Probabilities: [benign: 1.0, malignant: 0.0]
```

```
Predicted Value: benign
```

```
Target Value: benign
```

```
Cross Entropy: 0.0
```

```
Index ID: 52
```

	Feature Name	Feature Value	Contribution to Prediction
↳SHAP Value			
↳=====			
	mean concavity	0.02	-
↳-0.02	worst area	527.20	-
↳-0.03	worst radius	13.10	-
↳-0.04	worst concave points	0.06	-
↳-0.04	mean concave points	0.01	-
↳-0.05	worst perimeter	83.67	-
↳-0.06			

(continues on next page)

(continued from previous page)

Worst 1 of 2

Predicted Probabilities: [benign: 0.266, malignant: 0.734]  
 Predicted Value: malignant  
 Target Value: benign  
 Cross Entropy: 1.325  
 Index ID: 363

	Feature Name	Feature Value	Contribution to Prediction	SHAP
↪Value				
↪=====				
	worst perimeter	117.20	+	0.13
	worst radius	18.13	+	0.12
	worst area	1009.00	+	0.11
	mean area	838.10	+	0.06
	mean radius	16.50	+	0.05
	worst concavity	0.17	-	-0.05

Worst 2 of 2

Predicted Probabilities: [benign: 1.0, malignant: 0.0]  
 Predicted Value: benign  
 Target Value: malignant  
 Cross Entropy: 7.987  
 Index ID: 135

	Feature Name	Feature Value	Contribution to Prediction	
↪SHAP Value				
↪=====				
	mean concavity	0.05	-	
↪-0.03				
	worst area	653.60	-	
↪-0.04				
	worst concave points	0.09	-	
↪-0.05				
	worst radius	14.49	-	
↪-0.05				
	worst perimeter	92.04	-	
↪-0.06				
	mean concave points	0.03	-	
↪-0.06				

We use a custom metric ([hinge loss](#)) for selecting the best and worst predictions. See this example:

```
import numpy as np

def hinge_loss(y_true, y_pred_proba):
    probabilities = np.clip(y_pred_proba.iloc[:, 1], 0.001, 0.999)
```

(continues on next page)

(continued from previous page)

```

y_true[y_true == 0] = -1

    return np.clip(1 - y_true * np.log(probabilities / (1 - probabilities)), a_min=0,
↪a_max=None)

report = explain_predictions_best_worst(pipeline=pipeline, input_features=X, y_true=y,
↪include_shap_values=True, num_to_explain=5,
↪metric= hinge_loss)

print(report)

```

## Changing Output Formats

Instead of getting the prediction explanations as text, you can get the report as a python dictionary or pandas dataframe. All you have to do is pass `output_format="dict"` or `output_format="dataframe"` to either `explain_prediction`, `explain_predictions`, or `explain_predictions_best_worst`.

## Single prediction as a dictionary

```

[24]: import json
single_prediction_report = explain_predictions(pipeline=pipeline_binary, input_
↪features=X_holdout, indices_to_explain=[3],
↪y=y_holdout, top_k_features=6, include_
↪shap_values=True,
↪output_format="dict")
print(json.dumps(single_prediction_report, indent=2))

```

```

{
  "explanations": [
    {
      "explanations": [
        {
          "feature_names": [
            "worst concavity",
            "mean concavity",
            "worst area",
            "worst radius",
            "mean concave points",
            "worst perimeter"
          ],
          "feature_values": [
            0.1791,
            0.038,
            599.5,
            14.04,
            0.034,
            92.8
          ],
          "qualitative_explanation": [
            "-",
            "-",
            "-",
            "-",
            "-",
            "-"
          ]
        }
      ]
    }
  ]
}

```

(continues on next page)

(continued from previous page)

```

        "_":
    ],
    "quantitative_explanation": [
        -0.023008481104309524,
        -0.02621982146725469,
        -0.033821592020020774,
        -0.04666659740586632,
        -0.0541511910494414,
        -0.05523688273171911
    ],
    "drill_down": {},
    "class_name": "malignant",
    "expected_value": 0.3711208791208791
    }
    ]
    }
    ]
    }

```

### Single prediction as a dataframe

```

[25]: single_prediction_report = explain_predictions(pipeline=pipeline_binary, input_
    ↪ features=X_holdout,
                                           indices_to_explain=[3],
                                           y=y_holdout, top_k_features=6, include_
    ↪ shap_values=True,
                                           output_format="dataframe")
single_prediction_report

```

```

[25]:
   feature_names  feature_values  qualitative_explanation  \
0    worst concavity           0.1791                  -
1    mean concavity           0.0380                  -
2    worst area             599.5000                  -
3    worst radius           14.0400                  -
4  mean concave points           0.0340                  -
5    worst perimeter          92.8000                  -

   quantitative_explanation  class_name  prediction_number
0          -0.023008    malignant              0
1          -0.026220    malignant              0
2          -0.033822    malignant              0
3          -0.046667    malignant              0
4          -0.054151    malignant              0
5          -0.055237    malignant              0

```

## Best and worst predictions as a dictionary

```
[26]: report = explain_predictions_best_worst(pipeline=pipeline_binary, input_features=X, y_
      ↪ true=y,
      num_to_explain=1, top_k_features=6,
      include_shap_values=True, output_format="dict
      ↪")
print(json.dumps(report, indent=2))
```

```
{
  "explanations": [
    {
      "rank": {
        "prefix": "best",
        "index": 1
      },
      "predicted_values": {
        "probabilities": {
          "benign": 1.0,
          "malignant": 0.0
        },
        "predicted_value": "benign",
        "target_value": "benign",
        "error_name": "Cross Entropy",
        "error_value": 0.0001970443507070075,
        "index_id": 475
      },
      "explanations": [
        {
          "feature_names": [
            "mean concavity",
            "worst area",
            "worst radius",
            "worst concave points",
            "worst perimeter",
            "mean concave points"
          ],
          "feature_values": [
            0.05835,
            605.8,
            14.09,
            0.09783,
            93.22,
            0.03078
          ],
          "qualitative_explanation": [
            "-",
            "-",
            "-",
            "-",
            "-",
            "-"
          ],
          "quantitative_explanation": [
            -0.028481050954786636,
            -0.03050522196002462,
            -0.042922079201003216,
            -0.04429366151003684,
```

(continues on next page)

(continued from previous page)

```

        -0.05486784013962313,
        -0.05639460900233733
    ],
    "drill_down": {},
    "class_name": "malignant",
    "expected_value": 0.3711208791208791
}
]
},
{
    "rank": {
        "prefix": "worst",
        "index": 1
    },
    "predicted_values": {
        "probabilities": {
            "benign": 1.0,
            "malignant": 0.0
        },
        "predicted_value": "benign",
        "target_value": "malignant",
        "error_name": "Cross Entropy",
        "error_value": 7.986911819330411,
        "index_id": 135
    },
    "explanations": [
        {
            "feature_names": [
                "mean concavity",
                "worst area",
                "worst concave points",
                "worst radius",
                "worst perimeter",
                "mean concave points"
            ],
            "feature_values": [
                0.04711,
                653.6,
                0.09331,
                14.49,
                92.04,
                0.02704
            ],
            "qualitative_explanation": [
                "-",
                "-",
                "-",
                "-",
                "-",
                "-"
            ],
            "quantitative_explanation": [
                -0.029936744551331215,
                -0.03748357654576422,
                -0.04553126236476177,
                -0.0483274199182721,
                -0.06039220265366764,

```

(continues on next page)



(continued from previous page)

```

        -0.060441902449258976
    ],
    "drill_down": {},
    "class_name": "malignant",
    "expected_value": 0.3711208791208791
}
]
}
]
}

```

### Best and worst predictions as a dataframe

```

[27]: report = explain_predictions_best_worst(pipeline=pipeline_binary, input_features=X_
↳ holdout, y_true=y_holdout,
                                           num_to_explain=1, top_k_features=6,
                                           include_shap_values=True, output_format=
↳ "dataframe")
report

```

```

[27]:
   feature_names  feature_values  qualitative_explanation  \
0      mean concavity      0.05928                    -
1      worst area      552.00000                    -
2  worst concave points      0.08411                    -
3      worst radius      13.57000                    -
4  mean concave points      0.03279                    -
5  worst perimeter      86.67000                    -
6      mean concavity      0.04711                    -
7      worst area      653.60000                    -
8  worst concave points      0.09331                    -
9      worst radius      14.49000                    -
10     worst perimeter      92.04000                    -
11  mean concave points      0.02704                    -

   quantitative_explanation  class_name  label_benign_probability  \
0          -0.029022  malignant                    1.0
1          -0.034112  malignant                    1.0
2          -0.046896  malignant                    1.0
3          -0.046928  malignant                    1.0
4          -0.052902  malignant                    1.0
5          -0.064320  malignant                    1.0
6          -0.029937  malignant                    1.0
7          -0.037484  malignant                    1.0
8          -0.045531  malignant                    1.0
9          -0.048327  malignant                    1.0
10         -0.060392  malignant                    1.0
11         -0.060442  malignant                    1.0

   label_malignant_probability  predicted_value  target_value  error_name  \
0                        0.0          benign    benign  Cross Entropy
1                        0.0          benign    benign  Cross Entropy
2                        0.0          benign    benign  Cross Entropy
3                        0.0          benign    benign  Cross Entropy
4                        0.0          benign    benign  Cross Entropy
5                        0.0          benign    benign  Cross Entropy

```

(continues on next page)

(continued from previous page)

6	0.0	benign	malignant	Cross Entropy
7	0.0	benign	malignant	Cross Entropy
8	0.0	benign	malignant	Cross Entropy
9	0.0	benign	malignant	Cross Entropy
10	0.0	benign	malignant	Cross Entropy
11	0.0	benign	malignant	Cross Entropy

	error_value	index_id	rank	prefix
0	0.000197	502	1	best
1	0.000197	502	1	best
2	0.000197	502	1	best
3	0.000197	502	1	best
4	0.000197	502	1	best
5	0.000197	502	1	best
6	7.986912	135	1	worst
7	7.986912	135	1	worst
8	7.986912	135	1	worst
9	7.986912	135	1	worst
10	7.986912	135	1	worst
11	7.986912	135	1	worst

## Force Plots

Force plots can be generated to predict single or multiple rows for binary, multiclass and regression problem types. Here's an example of predicting a single row on a binary classification dataset. The force plots show the predictive power of each of the features in making the negative ("Class: 0") prediction and the positive ("Class: 1") prediction.

```
[28]: import shap

from evalml.model_understanding.force_plots import graph_force_plot

rows_to_explain = [0] # Should be a list of integer indices of the rows to explain.

results = graph_force_plot(pipeline_binary, rows_to_explain=rows_to_explain,
                           training_data=X_holdout, y=y_holdout)

for result in results:
    for cls in result:
        print("Class:", cls)
        display(result[cls]["plot"])
```

<IPython.core.display.HTML object>

Class: malignant

<shap.plots.\_force.AdditiveForceVisualizer at 0x7fb4e729e6d0>

Here's an example of a force plot explaining multiple predictions on a multiclass problem. These plots show the force plots for each row arranged as consecutive columns that can be ordered by the dropdown above. Clicking the column indicates which row explanation is underneath.

```
[29]: rows_to_explain = [0,1,2,3,4] # Should be a list of integer indices of the rows to_
    ↪ explain.

results = graph_force_plot(pipeline_multi,
                           rows_to_explain=rows_to_explain,
```

(continues on next page)

(continued from previous page)

```

                                training_data=X_multi, y=y_multi)

for idx, result in enumerate(results):
    print("Row:", idx)
    for cls in result:
        print("Class:", cls)
        display(result[cls]["plot"])

```

<IPython.core.display.HTML object>

Row: 0  
Class: class\_0

<shap.plots.\_force.AdditiveForceVisualizer at 0x7fb4e5c45b50>

Class: class\_1

<shap.plots.\_force.AdditiveForceVisualizer at 0x7fb4e5c45cd0>

Class: class\_2

<shap.plots.\_force.AdditiveForceVisualizer at 0x7fb4e71f08b0>

Row: 1  
Class: class\_0

<shap.plots.\_force.AdditiveForceVisualizer at 0x7fb4e71f0190>

Class: class\_1

<shap.plots.\_force.AdditiveForceVisualizer at 0x7fb4e71f04c0>

Class: class\_2

<shap.plots.\_force.AdditiveForceVisualizer at 0x7fb4e71f0a60>

Row: 2  
Class: class\_0

<shap.plots.\_force.AdditiveForceVisualizer at 0x7fb4e71f0b80>

Class: class\_1

<shap.plots.\_force.AdditiveForceVisualizer at 0x7fb4e71f05b0>

Class: class\_2

<shap.plots.\_force.AdditiveForceVisualizer at 0x7fb4e7c601c0>

Row: 3  
Class: class\_0

<shap.plots.\_force.AdditiveForceVisualizer at 0x7fb4e7235f70>

Class: class\_1

<shap.plots.\_force.AdditiveForceVisualizer at 0x7fb4ced3d610>

Class: class\_2

<shap.plots.\_force.AdditiveForceVisualizer at 0x7fb4ced3dd00>

Row: 4  
Class: class\_0

<shap.plots.\_force.AdditiveForceVisualizer at 0x7fb4ced3de20>

Class: class\_1

```
<shap.plots._force.AdditiveForceVisualizer at 0x7fb4ced3ddc0>
Class: class_2
<shap.plots._force.AdditiveForceVisualizer at 0x7fb4ced28700>
```

## 4.6 Data Checks

EvalML provides data checks to help guide you in achieving the highest performing model. These utility functions help deal with problems such as overfitting, abnormal data, and missing data. These data checks can be found under `evalml/data_checks`. Below we will cover examples for each available data check in EvalML, as well as the `DefaultDataChecks` collection of data checks.

### 4.6.1 Missing Data

Missing data or rows with NaN values provide many challenges for machine learning pipelines. In the worst case, many algorithms simply will not run with missing data! EvalML pipelines contain imputation *components* to ensure that doesn't happen. Imputation works by approximating missing values with existing values. However, if a column contains a high number of missing values, a large percentage of the column would be approximated by a small percentage. This could potentially create a column without useful information for machine learning pipelines. By using `HighlyNullDataCheck`, EvalML will alert you to this potential problem by returning the columns that pass the missing values threshold.

```
[1]: import numpy as np
import pandas as pd

from evalml.data_checks import HighlyNullDataCheck

X = pd.DataFrame([[1, 2, 3],
                  [0, 4, np.nan],
                  [1, 4, np.nan],
                  [9, 4, np.nan],
                  [8, 6, np.nan]])

null_check = HighlyNullDataCheck(pct_null_col_threshold=0.8, pct_null_row_threshold=0.
↪8)
results = null_check.validate(X)

for message in results['warnings']:
    print("Warning:", message['message'])

for message in results['errors']:
    print("Error:", message['message'])

Warning: Column '2' is 80.0% or more null
```

## 4.6.2 Abnormal Data

EvalML provides a few data checks to check for abnormal data:

- NoVarianceDataCheck
- ClassImbalanceDataCheck
- TargetLeakageDataCheck
- InvalidTargetDataCheck
- IDColumnsDataCheck
- OutliersDataCheck
- HighVarianceCVDDataCheck
- MulticollinearityDataCheck
- UniquenessDataCheck
- TargetDistributionDataCheck
- DateTimeFormatDataCheck

### Zero Variance

Data with zero variance indicates that all values are identical. If a feature has zero variance, it is not likely to be a useful feature. Similarly, if the target has zero variance, there is likely something wrong. NoVarianceDataCheck checks if the target or any feature has only one unique value and alerts you to any such columns.

```
[2]: from evalml.data_checks import NoVarianceDataCheck
X = pd.DataFrame({"no var col": [0, 0, 0],
                  "good col": [0, 4, 1]})
y = pd.Series([1, 0, 1])
no_variance_data_check = NoVarianceDataCheck()
results = no_variance_data_check.validate(X, y)

for message in results['warnings']:
    print("Warning:", message['message'])

for message in results['errors']:
    print("Error:", message['message'])

Error: no var col has 1 unique value.
```

Note that you can set NaN to count as an unique value, but NoVarianceDataCheck will still return a warning if there is only one unique non-NaN value in a given column.

```
[3]: from evalml.data_checks import NoVarianceDataCheck

X = pd.DataFrame({"no var col": [0, 0, 0],
                  "no var col with nan": [1, np.nan, 1],
                  "good col": [0, 4, 1]})
y = pd.Series([1, 0, 1])

no_variance_data_check = NoVarianceDataCheck(count_nan_as_value=True)
results = no_variance_data_check.validate(X, y)

for message in results['warnings']:
```

(continues on next page)

(continued from previous page)

```
print("Warning:", message['message'])

for message in results['errors']:
    print("Error:", message['message'])
```

```
Warning: no var col with nan has two unique values including nulls. Consider encoding
↳the nulls for this column to be useful for machine learning.
Error: no var col has 1 unique value.
```

## Class Imbalance

For classification problems, the distribution of examples across each class can vary. For small variations, this is normal and expected. However, when the number of examples for each class label is disproportionately biased or skewed towards a particular class (or classes), it can be difficult for machine learning models to predict well. In addition, having a low number of examples for a given class could mean that one or more of the CV folds generated for the training data could only have few or no examples from that class. This may cause the model to only predict the majority class and ultimately resulting in a poor-performant model.

`ClassImbalanceDataCheck` checks if the target labels are imbalanced beyond a specified threshold for a certain number of CV folds. It returns `DataCheckError` messages for any classes that have less samples than double the number of CV folds specified (since that indicates the likelihood of having at little to no samples of that class in a given fold), and `DataCheckWarning` messages for any classes that fall below the set threshold percentage.

```
[4]: from evalml.data_checks import ClassImbalanceDataCheck

X = pd.DataFrame([[1, 2, 0, 1],
                  [4, 1, 9, 0],
                  [4, 4, 8, 3],
                  [9, 2, 7, 1]])
y = pd.Series([0, 1, 1, 1, 1])

class_imbalance_check = ClassImbalanceDataCheck(threshold=0.25, num_cv_folds=4)
results = class_imbalance_check.validate(X, y)

for message in results['warnings']:
    print("Warning:", message['message'])

for message in results['errors']:
    print("Error:", message['message'])
```

```
Warning: The following labels fall below 25% of the target: [0]
Warning: The following labels in the target have severe class imbalance because they
↳fall under 25% of the target and have less than 100 samples: [0]
Error: The number of instances of these targets is less than 2 * the number of cross
↳folds = 8 instances: [0, 1]
```

## Target Leakage

**Target leakage**, also known as data leakage, can occur when you train your model on a dataset that includes information that should not be available at the time of prediction. This causes the model to score suspiciously well, but perform poorly in production. `TargetLeakageDataCheck` checks for features that could potentially be “leaking” information by calculating the Pearson correlation coefficient between each feature and the target to warn users if there are features are highly correlated with the target. Currently, only numerical features are considered.

```
[5]: from evalml.data_checks import TargetLeakageDataCheck
X = pd.DataFrame({'leak': [10, 42, 31, 51, 61],
                  'x': [42, 54, 12, 64, 12],
                  'y': [12, 5, 13, 74, 24]})
y = pd.Series([10, 42, 31, 51, 40])

target_leakage_check = TargetLeakageDataCheck(pct_corr_threshold=0.8)
results = target_leakage_check.validate(X, y)

for message in results['warnings']:
    print("Warning:", message['message'])

for message in results['errors']:
    print("Error:", message['message'])
```

Warning: Column 'leak' is 80.0% or more correlated with the target  
Warning: Column 'x' is 80.0% or more correlated with the target  
Warning: Column 'y' is 80.0% or more correlated with the target

## Invalid Target Data

The `InvalidTargetDataCheck` checks if the target data contains any missing or invalid values. Specifically:

- if any of the target values are missing, a `DataCheckError` message is returned
- if the specified problem type is a binary classification problem but there is more or less than two unique values in the target, a `DataCheckError` message is returned
- if binary classification target classes are numeric values not equal to `{0, 1}`, a `DataCheckError` message is returned because it can cause unpredictable behavior when passed to pipelines

```
[6]: from evalml.data_checks import InvalidTargetDataCheck

X = pd.DataFrame({})
y = pd.Series([0, 1, None, None])

invalid_target_check = InvalidTargetDataCheck('binary', 'Log Loss Binary')
results = invalid_target_check.validate(X, y)

for message in results['warnings']:
    print("Warning:", message['message'])

for message in results['errors']:
    print("Error:", message['message'])
```

Warning: Input target and features have different lengths  
Warning: Input target and features have mismatched indices  
Error: 2 row(s) (50.0%) of target values are null

## ID Columns

ID columns in your dataset provide little to no benefit to a machine learning pipeline as the pipeline cannot extrapolate useful information from unique identifiers. Thus, `IDColumnsDataCheck` reminds you if these columns exists. In the given example, 'user\_number' and 'id' columns are both identified as potentially being unique identifiers that should be removed.

```
[7]: from evalml.data_checks import IDColumnsDataCheck

X = pd.DataFrame([[0, 53, 6325, 5], [1, 90, 6325, 10], [2, 90, 18, 20]], columns=['user_
↪number', 'cost', 'revenue', 'id'])

id_col_check = IDColumnsDataCheck(id_threshold=0.9)
results = id_col_check.validate(X, y)

for message in results['warnings']:
    print("Warning:", message['message'])

for message in results['errors']:
    print("Error:", message['message'])

Warning: Column 'id' is 90.0% or more likely to be an ID column
Warning: Column 'user_number' is 90.0% or more likely to be an ID column
```

## Multicollinearity

The `MulticollinearityDataCheck` data check is used in to detect if are any set of features that are likely to be multicollinear. Multicollinear features affect the performance of a model, but more importantly, it may greatly impact model interpretation. EvalML uses mutual information to determine collinearity.

```
[8]: from evalml.data_checks import MulticollinearityDataCheck

y = pd.Series([1, 0, 2, 3, 4])
X = pd.DataFrame({'col_1': y,
                  'col_2': y * 3,
                  'col_3': ~y,
                  'col_4': y / 2,
                  'col_5': y + 1,
                  'not_collinear': [0, 1, 0, 0, 0]})

multi_check = MulticollinearityDataCheck(threshold=0.95)
results = multi_check.validate(X, y)

for message in results['warnings']:
    print("Warning:", message['message'])

for message in results['errors']:
    print("Error:", message['message'])

Warning: Columns are likely to be correlated: [('col_1', 'col_2'), ('col_1', 'col_3'),
↪ ('col_1', 'col_4'), ('col_1', 'col_5'), ('col_2', 'col_3'), ('col_2', 'col_4'), (
↪ 'col_2', 'col_5'), ('col_3', 'col_4'), ('col_3', 'col_5'), ('col_4', 'col_5')]
```



## Uniqueness

The UniquenessDataCheck is used to detect columns with either too unique or not unique enough values. For regression type problems, the data is checked for a lower limit of uniqueness. For multiclass type problems, the data is checked for an upper limit.

```
[9]: import pandas as pd
from evalml.data_checks import UniquenessDataCheck

X = pd.DataFrame({'most_unique': [float(x) for x in range(10)], # [0,1,2,3,4,5,6,7,8,
↪9]
                  'more_unique': [x % 5 for x in range(10)], # [0,1,2,3,4,0,1,2,3,4]
                  'unique': [x % 3 for x in range(10)], # [0,1,2,0,1,2,0,1,2,0]
                  'less_unique': [x % 2 for x in range(10)], # [0,1,0,1,0,1,0,1,0,1]
                  'not_unique': [float(1) for x in range(10)]}) # [1,1,1,1,1,1,1,1,1,1,
↪1]

uniqueness_check = UniquenessDataCheck(problem_type="regression",
                                       threshold=.5)
results = uniqueness_check.validate(X, y=None)

for message in results['warnings']:
    print("Warning:", message['message'])

for message in results['errors']:
    print("Error:", message['message'])

Warning: Input columns (not_unique) for regression problem type are not unique enough.
```

## Sparsity

The SparsityDataCheck is used to identify features that contain a sparsity of values.

```
[10]: from evalml.data_checks import SparsityDataCheck

X = pd.DataFrame({'most_sparse': [float(x) for x in range(10)], # [0,1,2,3,4,5,6,7,8,
↪9]
                  'more_sparse': [x % 5 for x in range(10)], # [0,1,2,3,4,0,1,2,3,
↪4]
                  'sparse': [x % 3 for x in range(10)], # [0,1,2,0,1,2,0,1,2,
↪0]
                  'less_sparse': [x % 2 for x in range(10)], # [0,1,0,1,0,1,0,1,0,
↪1]
                  'not_sparse': [float(1) for x in range(10)]}) # [1,1,1,1,1,1,1,1,1,
↪1]

sparsity_check = SparsityDataCheck(problem_type="multiclass",
                                   threshold=.4,
                                   unique_count_threshold=3)
results = sparsity_check.validate(X, y=None)

for message in results['warnings']:
    print("Warning:", message['message'])

for message in results['errors']:
    print("Error:", message['message'])
```

```
Warning: Input columns (most_sparse) for multiclass problem type are too sparse.
Warning: Input columns (more_sparse) for multiclass problem type are too sparse.
Warning: Input columns (sparse) for multiclass problem type are too sparse.
```

## Outliers

Outliers are observations that differ significantly from other observations in the same sample. Many machine learning pipelines suffer in performance if outliers are not dropped from the training set as they are not representative of the data. `OutliersDataCheck()` uses IQR to notify you if a sample can be considered an outlier.

Below we generate a random dataset with some outliers.

```
[11]: data = np.tile(np.arange(10) * 0.01, (100, 10))
      X = pd.DataFrame(data=data)

      # generate some outliers in columns 3, 25, 55, and 72
      X.iloc[0, 3] = -10000
      X.iloc[3, 25] = 10000
      X.iloc[5, 55] = 10000
      X.iloc[10, 72] = -10000
```

We then utilize `OutliersDataCheck()` to rediscover these outliers.

```
[12]: from evalml.data_checks import OutliersDataCheck

      outliers_check = OutliersDataCheck()
      results = outliers_check.validate(X, y)

      for message in results['warnings']:
          print("Warning:", message['message'])

      for message in results['errors']:
          print("Error:", message['message'])

      Warning: Column(s) '3', '25', '55', '72' are likely to have outlier data.
```

## Target Distribution

Target data can come in a variety of distributions, such as Gaussian or Lognormal. When we work with machine learning models, we feed data into an estimator that learns from the training data provided. Sometimes the data can be significantly spread out with a long tail or outliers, which could lead to a lognormal distribution. This can cause machine learning model performance to suffer.

To help the estimators better understand the underlying relationships in the data between the features and the target, we can use the `TargetDistributionDataCheck` to identify such a distribution. If you use `AutoML.search` to try and find the best pipeline, it will automatically run this data check for you and, if a lognormal distribution is detected, will add a `LogTransformer` to your pipeline to help the model performance!

```
[13]: from scipy.stats import lognorm
      from evalml.data_checks import TargetDistributionDataCheck

      data = np.tile(np.arange(10) * 0.01, (100, 10))
      X = pd.DataFrame(data=data)
      y = pd.Series(lognorm.rvs(s=0.4, loc=1, scale=1, size=100))
```

(continues on next page)

(continued from previous page)

```
target_dist_check = TargetDistributionDataCheck()
results = target_dist_check.validate(X, y)

for message in results['warnings']:
    print("Warning:", message['message'])

for message in results['errors']:
    print("Error:", message['message'])

Warning: Target may have a lognormal distribution.
```

## Datetime Format

Datetime information is a necessary component of time series problems, but sometimes the data we deal with may contain flaws that make it impossible for time series models to work with them. For example, in order to identify a frequency in the datetime information there has to be equal interval spacing between data points i.e. January 1, 2021, January 3, 2021, January 5, 2021, ...etc which are separated by two days. If instead there are random jumps in the datetime data i.e. January 1, 2021, January 3, 2021, January 12, 2021, then a frequency can't be inferred. Another common issue with time series models are that they can't handle datetime information that isn't properly sorted. Datetime values that aren't monotonically increasing (sorted in ascending order) will encounter this issue and their frequency cannot be inferred.

To make it easy to verify that the datetime column you're working with is properly spaced and sorted, we can leverage the `DatetimeFormatDataCheck`. When initializing the data check, pass in the name of the column that contains your datetime information (or pass in "index" if it's found in either your X or y indices).

```
[14]: from evalml.data_checks import DateTimeFormatDataCheck

X = pd.DataFrame(pd.date_range("January 1, 2021", periods=8, freq='2D'), columns=[
    ↪ "dates"])
y = pd.Series([1, 2, 4, 2, 1, 2, 3, 1])

# Replaces the last entry with January 16th instead of January 15th
# so that the data is no longer evenly spaced.
X.iloc[7] = "January 16, 2021"

datetime_format_check = DateTimeFormatDataCheck(datetime_column="dates")
results = datetime_format_check.validate(X, y)

for message in results['warnings']:
    print("Warning:", message['message'])

for message in results['errors']:
    print("Error:", message['message'])

print("-----")

# Reverses the order of the index datetime values to be decreasing.
X = X[::-1]
results = datetime_format_check.validate(X, y)

for message in results['warnings']:
    print("Warning:", message['message'])

for message in results['errors']:
    print("Error:", message['message'])
```

```
Error: No frequency could be detected in dates, possibly due to uneven intervals.
-----
Error: No frequency could be detected in dates, possibly due to uneven intervals.
Error: Datetime values must be sorted in ascending order.
```

### 4.6.3 Data Check Messages

Each data check's `validate` method returns a list of `DataCheckMessage` objects indicating warnings or errors found; warnings are stored as a `DataCheckWarning` object ([API reference](#)) and errors are stored as a `DataCheckError` object ([API reference](#)). You can filter the messages returned by a data check by checking for the type of message returned. Below, `NoVarianceDataCheck` returns a list containing a `DataCheckWarning` and a `DataCheckError` message. We can determine which is which by checking the type of each message.

```
[15]: from evalml.data_checks import NoVarianceDataCheck, DataCheckError, DataCheckWarning
```

```
X = pd.DataFrame({"no var col": [0, 0, 0],
                  "no var col with nan": [1, np.nan, 1],
                  "good col": [0, 4, 1]})
y = pd.Series([1, 0, 1])

no_variance_data_check = NoVarianceDataCheck(count_nan_as_value=True)
results = no_variance_data_check.validate(X, y)
```

```
for message in results['warnings']:
    print("Warning:", message['message'])
```

```
for message in results['errors']:
    print("Error:", message['message'])
```

```
Warning: no var col with nan has two unique values including nulls. Consider encoding
↳ the nulls for this column to be useful for machine learning.
Error: no var col has 1 unique value.
```

### 4.6.4 Writing Your Own Data Check

If you would prefer to write your own data check, you can do so by extending the `DataCheck` class and implementing the `validate(self, X, y)` class method. Below, we've created a new `DataCheck`, `ZeroVarianceDataCheck`, which is similar to `NoVarianceDataCheck` defined in EvalML. The `validate(self, X, y)` method should return a dictionary with 'warnings' and 'errors' as keys mapping to list of warnings and errors, respectively.

```
[16]: from evalml.data_checks import DataCheck
```

```
class ZeroVarianceDataCheck(DataCheck):
    def validate(self, X, y):
        messages = {'warnings': [], 'errors': []}
        if not isinstance(X, pd.DataFrame):
            X = pd.DataFrame(X)
        warning_msg = "Column '{}' has zero variance"
        messages['warnings'].extend([DataCheckError(warning_msg.format(column), self.
↳ name) for column in X.columns if len(X[column].unique()) == 1])
```

## 4.6.5 Defining Collections of Data Checks

For convenience, EvalML provides a `DataChecks` class to represent a collection of data checks. We will go over `DefaultDataChecks` ([API reference](#)), a collection defined to check for some of the most common data issues.

### Default Data Checks

`DefaultDataChecks` is a collection of data checks defined to check for some of the most common data issues. They include:

- `HighlyNullDataCheck`
- `IDColumnsDataCheck`
- `TargetLeakageDataCheck`
- `InvalidTargetDataCheck`
- `TargetDistributionDataCheck` (for regression problem types)
- `ClassImbalanceDataCheck` (for classification problem types)
- `NoVarianceDataCheck`
- `DateTimeNaNDataCheck`
- `NaturalLanguageNaNDataCheck`
- `DateTimeFormatDataCheck` (for time series problem types)

## 4.6.6 Writing Your Own Collection of Data Checks

If you would prefer to create your own collection of data checks, you could either write your own data checks class by extending the `DataChecks` class and setting the `self.data_checks` attribute to the list of `DataCheck` classes or objects, or you could pass that list of data checks to the constructor of the `DataChecks` class. Below, we create two identical collections of data checks using the two different methods.

```
[17]: # Create a subclass of `DataChecks`
from evalml.data_checks import DataChecks, HighlyNullDataCheck,
↳InvalidTargetDataCheck, NoVarianceDataCheck, ClassImbalanceDataCheck,
↳TargetLeakageDataCheck
from evalml.problem_types import ProblemTypes, handle_problem_types

class MyCustomDataChecks(DataChecks):

    data_checks = [HighlyNullDataCheck, InvalidTargetDataCheck, NoVarianceDataCheck,
↳TargetLeakageDataCheck]

    def __init__(self, problem_type, objective):
        """
        A collection of basic data checks.
        Arguments:
            problem_type (str): The problem type that is being validated. Can be
↳regression, binary, or multiclass.
        """
        if handle_problem_types(problem_type) == ProblemTypes.REGRESSION:
            super().__init__(self.data_checks,
                             data_check_params={"InvalidTargetDataCheck": {"problem_
↳type": problem_type,
```

(continues on next page)

(continued from previous page)

```

        ↪": objective}))
        else:
            super().__init__(self.data_checks + [ClassImbalanceDataCheck],
                             data_check_params={"InvalidTargetDataCheck": {"problem_
        ↪type": problem_type,
                             "objective
        ↪": objective}))

custom_data_checks = MyCustomDataChecks(problem_type=ProblemTypes.REGRESSION,
        ↪objective="R2")
for data_check in custom_data_checks.data_checks:
    print(data_check.name)

HighlyNullDataCheck
InvalidTargetDataCheck
NoVarianceDataCheck
TargetLeakageDataCheck

```

```

[18]: # Pass list of data checks to the `data_checks` parameter of DataChecks
same_custom_data_checks = DataChecks(data_checks=[HighlyNullDataCheck,
        ↪InvalidTargetDataCheck, NoVarianceDataCheck, TargetLeakageDataCheck],
                                     data_check_params={"InvalidTargetDataCheck": {
        ↪"problem_type": ProblemTypes.REGRESSION,
        ↪"objective": "R2"}})
for data_check in custom_data_checks.data_checks:
    print(data_check.name)

HighlyNullDataCheck
InvalidTargetDataCheck
NoVarianceDataCheck
TargetLeakageDataCheck

```

## 4.7 Utilities

### 4.7.1 Configuring Logging

EvalML uses [the standard python logging package](#). By default, EvalML will log INFO-level logs and higher (warnings, errors and critical) to stdout, and will log everything to `evalml_debug.log` in the current working directory.

If you want to change the location of the logfile, before import, set the `EVALML_LOG_FILE` environment variable to specify a filename within an existing directory in which you have write permission. If you want to disable logging to the logfile, set `EVALML_LOG_FILE` to be empty. If the environment variable is set to an invalid location, EvalML will print a warning message to stdout and will not create a log file.

## 4.7.2 System Information

EvalML provides a command-line interface (CLI) tool prints the version of EvalML and core dependencies installed, as well as some basic system information. To use this tool, just run `evalml info` in your shell or terminal. This could be useful for debugging purposes or tracking down any version-related issues.

```
[1]: !evalml info
```

```
EvalML version: 0.30.2
EvalML installation directory: /home/docs/checkouts/readthedocs.org/user_builds/
    ↪ feature-labs-inc-evalml/envs/v0.30.2/lib/python3.8/site-packages/evalml
```

### SYSTEM INFO

```
-----
python: 3.8.6.final.0
python-bits: 64
OS: Linux
OS-release: 5.4.0-1035-aws
machine: x86_64
processor: x86_64
byteorder: little
LC_ALL: None
LANG: C.UTF-8
LOCALE: en_US.UTF-8
# of CPUs: 2
Available memory: 6.0G
```

### INSTALLED VERSIONS

```
-----
zict: 2.0.0
xgboost: 1.4.2
wrapit: 1.12.1
woodwork: 0.5.1
widgetsnbextension: 3.5.1
wheel: 0.37.0
webencodings: 0.5.1
wcwidth: 0.2.5
urllib3: 1.26.6
unidecode: 1.2.0
traitlets: 5.0.5
tqdm: 4.62.1
tornado: 6.1
toolz: 0.11.1
threadpoolctl: 2.2.0
texttable: 1.6.4
testpath: 0.5.0
terminado: 0.11.0
tenacity: 8.0.1
tblib: 1.7.0
statsmodels: 0.12.2
sphinxcontrib-websupport: 1.2.4
sphinxcontrib-serializinghtml: 1.1.5
sphinxcontrib-qthelp: 1.0.3
sphinxcontrib-jsmath: 1.0.1
sphinxcontrib-htmlhelp: 2.0.0
sphinxcontrib-devhelp: 1.0.2
sphinxcontrib-applehelp: 1.0.2
sphinx: 3.5.4
```

(continues on next page)

(continued from previous page)

```
sphinx-rtd-theme: 0.4.3
sphinx-autoapi: 1.8.3
soupsieve: 2.2.1
sortedcontainers: 2.4.0
snowballstemmer: 2.1.0
slicer: 0.0.7
sktime: 0.7.0
six: 1.16.0
shap: 0.39.0
setuptools: 57.4.0
send2trash: 1.8.0
seaborn: 0.11.2
scipy: 1.7.1
scikit-optimize: 0.8.1
scikit-learn: 0.24.2
requirements-parser: 0.2.0
requests: 2.26.0
regex: 2021.8.3
recommonmark: 0.5.0
readthedocs-sphinx-ext: 2.1.4
pyzmq: 22.2.1
pyyaml: 5.4.1
pytz: 2021.1
python-dateutil: 2.8.2
pysistent: 0.18.0
pyparsing: 2.4.7
pygments: 2.10.0
pydata-sphinx-theme: 0.6.3
pyparser: 2.20
pyaml: 21.8.3
ptyprocess: 0.7.0
psutil: 5.8.0
prompt-toolkit: 3.0.19
prometheus-client: 0.11.0
pmdarima: 1.8.0
plotly: 5.2.1
pip: 21.2.4
pillow: 8.3.1
pickleshare: 0.7.5
pexpect: 4.8.0
patsy: 0.5.1
partd: 1.2.0
parso: 0.8.2
pandocfilters: 1.4.3
pandas: 1.3.2
packaging: 21.0
numpy: 1.21.2
numba: 0.53.1
notebook: 6.4.3
nltk: 3.6.2
nlp-primitives: 1.1.0
networkx: 2.5.1
nest-asyncio: 1.5.1
nbsphinx: 0.8.7
nbformat: 5.1.3
nbconvert: 6.1.0
nbclient: 0.5.4
```

(continues on next page)



(continued from previous page)

```
msgpack: 1.0.2
mock: 1.0.1
mistune: 0.8.4
matplotlib: 3.4.3
matplotlib-inline: 0.1.2
markupsafe: 2.0.1
locket: 0.2.1
llvmlite: 0.36.0
lightgbm: 3.2.1
lazy-object-proxy: 1.6.0
kiwisolver: 1.3.1
kaleido: 0.2.1
jupyterlab-widgets: 1.0.0
jupyterlab-pygments: 0.1.2
jupyter-core: 4.7.1
jupyter-client: 6.1.12
jsonschema: 3.2.0
joblib: 1.0.1
jinja2: 3.0.1
jedi: 0.18.0
ipywidgets: 7.6.3
ipython: 7.26.0
ipython-genutils: 0.2.0
ipykernel: 6.2.0
imbalanced-learn: 0.8.0
imagesize: 1.2.0
idna: 3.2
heapdict: 1.0.1
graphviz: 0.17
future: 0.18.2
fsspec: 2021.7.0
featuretools: 0.26.1
evalml: 0.30.2
entrypoints: 0.3
docutils: 0.16
distributed: 2021.8.0
defusedxml: 0.7.1
decorator: 4.4.2
debugpy: 1.4.1
dask: 2021.8.0
cython: 0.29.17
cyclcr: 0.10.0
commonmark: 0.8.1
colorama: 0.4.4
cloudpickle: 1.6.0
click: 8.0.1
charset-normalizer: 2.0.4
cffi: 1.14.6
certifi: 2021.5.30
category-encoders: 2.2.2
catboost: 0.26.1
bleach: 4.0.0
beautifulsoup4: 4.9.3
backcall: 0.2.0
babel: 2.9.1
attrs: 21.2.0
astroid: 2.6.6
```

(continues on next page)

(continued from previous page)

```
argon2-cffi: 20.1.0
alabaster: 0.7.12
```

## 4.8 FAQ

### 4.8.1 Q: What is the difference between EvalML and other AutoML libraries?

EvalML optimizes machine learning pipelines on *custom practical objectives* instead of vague machine learning loss functions so that it will find the best pipelines for your specific needs. Furthermore, EvalML *pipelines* are able to take in all kinds of data (missing values, categorical, etc.) as long as the data are in a single table. EvalML also allows you to build your own pipelines with existing or custom components so you can have more control over the AutoML process. Moreover, EvalML also provides you with support in the form of *data checks* to ensure that you are aware of potential issues your data may cause with machine learning algorithms.

### 4.8.2 Q: How does EvalML handle missing values?

EvalML contains imputation components in its pipelines so that missing values are taken care of. EvalML optimizes over different types of imputation to search for the best possible pipeline. You can find more information about components [here](#) and in the API reference [here] (`../generated/evalml.pipelines.components.Imputer.ipynb`).

### 4.8.3 Q: How does EvalML handle categorical encoding?

EvalML provides a [one-hot-encoding component] (`../generated/evalml.pipelines.components.OneHotEncoder.ipynb`) in its pipelines for categorical variables. EvalML plans to support other encoders in the future.

### 4.8.4 Q: How does EvalML handle feature selection?

EvalML currently utilizes scikit-learn's `SelectFromModel` with a Random Forest classifier/regressor to handle feature selection. EvalML plans on supporting more feature selectors in the future. You can find more information in the API reference [here] (`../generated/evalml.pipelines.components.RFClassifierSelectFromModel.ipynb`).

### 4.8.5 Q: How is feature importance calculated?

Feature importance depends on the estimator used. Variable coefficients are used for regression-based estimators (Logistic Regression and Linear Regression) and Gini importance is used for tree-based estimators (Random Forest and XGBoost).

#### 4.8.6 Q: How does hyperparameter tuning work?

EvalML tunes hyperparameters for its pipelines through Bayesian optimization. In the future we plan to support more optimization techniques such as random search.

#### 4.8.7 Q: Can I create my own objective metric?

Yes you can! You can *create your own custom objective* so that EvalML optimizes the best model for your needs.

#### 4.8.8 Q: How does EvalML avoid overfitting?

EvalML provides *data checks* to combat overfitting. Such data checks include detecting label leakage, unstable pipelines, hold-out datasets and cross validation. EvalML defaults to using Stratified K-Fold cross-validation for classification problems and K-Fold cross-validation for regression problems but allows you to utilize your own cross-validation methods as well.

#### 4.8.9 Q: Can I create my own pipeline for EvalML?

Yes! EvalML allows you to create *custom pipelines* using modular components. This allows you to customize EvalML pipelines for your own needs or for AutoML.

#### 4.8.10 Q: Does EvalML work with X algorithm?

EvalML is constantly improving and adding new components and will allow your own algorithms to be used as components in our pipelines.



**API REFERENCE****5.1 Demo Datasets**

<i>load_breast_cancer</i>	Load breast cancer dataset. Binary classification problem.
<i>load_diabetes</i>	Load diabetes dataset. Regression problem
<i>load_fraud</i>	Load credit card fraud dataset.
<i>load_wine</i>	Load wine dataset. Multiclass problem.
<i>load_churn</i>	Load credit card fraud dataset.

**5.2 Preprocessing**

Utilities to preprocess data before using evalml.

<i>load_data</i>	Load features and target from file.
<i>drop_nan_target_rows</i>	Drops rows in X and y when row in the target y has a value of NaN.
<i>target_distribution</i>	Get the target distributions.
<i>number_of_features</i>	Get the number of features of each specific dtype in a DataFrame.
<i>split_data</i>	Splits data into train and test sets.

**5.3 Exceptions**

<i>MethodPropertyNotFoundError</i>	Exception to raise when a class is does not have an expected method or property.
<i>PipelineNotFoundError</i>	An exception raised when a particular pipeline is not found in automl search results
<i>ObjectiveNotFoundError</i>	Exception to raise when specified objective does not exist.
<i>MissingComponentError</i>	An exception raised when a component is not found in all_components()

continues on next page

Table 3 – continued from previous page

<i>ComponentNotYetFittedError</i>	An exception to be raised when predict/predict_proba/transform is called on a component without fitting first.
<i>PipelineNotYetFittedError</i>	An exception to be raised when predict/predict_proba/transform is called on a pipeline without fitting first.
<i>AutoMLSearchException</i>	Exception raised when all pipelines in an automl batch return a score of NaN for the primary objective.
<i>EnsembleMissingPipelinesError</i>	An exception raised when an ensemble is missing <i>estimators</i> (list) as a parameter.
<i>PipelineScoreError</i>	An exception raised when a pipeline errors while scoring any objective in a list of objectives.
<i>DataCheckInitError</i>	Exception raised when a data check can't initialize with the parameters given.
<i>NullsInColumnWarning</i>	Warning thrown when there are null values in the column of interest

## 5.4 AutoML

### 5.4.1 AutoML Search Interface

<i>AutoMLSearch</i>	Automated Pipeline search.
---------------------	----------------------------

### 5.4.2 AutoML Utils

<i>search</i>	Given data and configuration, run an automl search.
<i>get_default_primary_search_objective</i>	Get the default primary search objective for a problem type.
<i>make_data_splitter</i>	Given the training data and ML problem parameters, compute a data splitting method to use during AutoML search.

### 5.4.3 AutoML Algorithm Classes

<i>AutoMLAlgorithm</i>	Base class for the AutoML algorithms which power EvalML.
<i>IterativeAlgorithm</i>	An automl algorithm which first fits a base round of pipelines with default parameters, then does a round of parameter tuning on each pipeline in order of performance.

### 5.4.4 AutoML Callbacks

<i>silent_error_callback</i>	No-op.
<i>log_error_callback</i>	Logs the exception thrown as an error. Will not throw. This is the default behavior for AutoMLSearch.
<i>raise_error_callback</i>	Raises the exception thrown by the AutoMLSearch object. Also logs the exception as an error.

## 5.5 Pipelines

### 5.5.1 Pipeline Base Classes

<i>PipelineBase</i>	Machine learning pipeline made out of transformers and a estimator.
<i>ClassificationPipeline</i>	Pipeline subclass for all classification pipelines.
<i>BinaryClassificationPipeline</i>	Pipeline subclass for all binary classification pipelines.
<i>MulticlassClassificationPipeline</i>	Pipeline subclass for all multiclass classification pipelines.
<i>RegressionPipeline</i>	Pipeline subclass for all regression pipelines.
<i>TimeSeriesClassificationPipeline</i>	Pipeline base class for time series classification problems.
<i>TimeSeriesBinaryClassificationPipeline</i>	Pipeline base class for time series binary classification problems.
<i>TimeSeriesMulticlassClassificationPipeline</i>	Pipeline base class for time series multiclass classification problems.
<i>TimeSeriesRegressionPipeline</i>	Pipeline base class for time series regression problems.

### 5.5.2 Pipeline Utils

<i>make_pipeline</i>	Given input data, target data, an estimator class and the problem type,
<i>generate_pipeline_code</i>	Creates and returns a string that contains the Python imports and code required for running the EvalML pipeline.

## 5.6 Components

### 5.6.1 Component Base Classes

Components represent a step in a pipeline.

<i>ComponentBase</i>	Base class for all components.
<i>Transformer</i>	A component that may or may not need fitting that transforms data.

continues on next page

Table 10 – continued from previous page

<i>Estimator</i>	A component that fits and predicts given data.
------------------	--

## 5.6.2 Component Utils

<i>allowed_model_families</i>	List the model types allowed for a particular problem type.
<i>get_estimators</i>	Returns the estimators allowed for a particular problem type.
<i>generate_component_code</i>	Creates and returns a string that contains the Python imports and code required for running the EvalML component.

## 5.6.3 Transformers

Transformers are components that take in data as input and output transformed data.

<i>DropColumns</i>	Drops specified columns in input data.
<i>SelectColumns</i>	Selects specified columns in input data.
<i>SelectByType</i>	Selects columns by specified Woodwork logical type or semantic tag in input data.
<i>OneHotEncoder</i>	A transformer that encodes categorical features in a one-hot numeric array.
<i>TargetEncoder</i>	A transformer that encodes categorical features into target encodings.
<i>PerColumnImputer</i>	Imputes missing data according to a specified imputation strategy per column.
<i>Imputer</i>	Imputes missing data according to a specified imputation strategy.
<i>SimpleImputer</i>	Imputes missing data according to a specified imputation strategy.
<i>StandardScaler</i>	A transformer that standardizes input features by removing the mean and scaling to unit variance.
<i>RFRegressorSelectFromModel</i>	Selects top features based on importance weights using a Random Forest regressor.
<i>RFClassifierSelectFromModel</i>	Selects top features based on importance weights using a Random Forest classifier.
<i>DropNullColumns</i>	Transformer to drop features whose percentage of NaN values exceeds a specified threshold.
<i>DateTimeFeaturizer</i>	Transformer that can automatically extract features from datetime columns.
<i>TextFeaturizer</i>	Transformer that can automatically featurize text columns using featuretools' nlp_primitives.
<i>DelayedFeatureTransformer</i>	Transformer that delays input features and target variable for time series problems.
<i>DFSTransformer</i>	Featuretools DFS component that generates features for the input features.
<i>PolynomialDetrender</i>	Removes trends from time series by fitting a polynomial to the data.

continues on next page



Table 12 – continued from previous page

<i>Undersampler</i>	Initializes an undersampling transformer to downsample the majority classes in the dataset.
<i>SMOTEOversampler</i>	SMOTE Oversampler component. Works on numerical datasets only. This component is only run during training and not during predict.
<i>SMOTENCOversampler</i>	SMOTENC Oversampler component. Uses SMOTENC to generate synthetic samples. Works on a mix of numerical and categorical columns.
<i>SMOTENOVersampler</i>	SMOTEN Oversampler component. Uses SMOTEN to generate synthetic samples. Works for purely categorical datasets.

## 5.6.4 Estimators

### Classifiers

Classifiers are components that output a predicted class label.

<i>CatBoostClassifier</i>	CatBoost Classifier, a classifier that uses gradient-boosting on decision trees.
<i>ElasticNetClassifier</i>	Elastic Net Classifier. Uses Logistic Regression with elasticnet penalty as the base estimator.
<i>ExtraTreesClassifier</i>	Extra Trees Classifier.
<i>RandomForestClassifier</i>	Random Forest Classifier.
<i>LightGBMClassifier</i>	LightGBM Classifier.
<i>LogisticRegressionClassifier</i>	Logistic Regression Classifier.
<i>XGBoostClassifier</i>	XGBoost Classifier.
<i>BaselineClassifier</i>	Classifier that predicts using the specified strategy.
<i>SklearnStackedEnsembleClassifier</i>	Scikit-learn Stacked Ensemble Classifier.
<i>DecisionTreeClassifier</i>	Decision Tree Classifier.
<i>KNeighborsClassifier</i>	K-Nearest Neighbors Classifier.
<i>SVMClassifier</i>	Support Vector Machine Classifier.

### Regressors

Regressors are components that output a predicted target value.

<i>ARIMAREgressor</i>	Autoregressive Integrated Moving Average Model.
<i>CatBoostRegressor</i>	CatBoost Regressor, a regressor that uses gradient-boosting on decision trees.
<i>ElasticNetRegressor</i>	Elastic Net Regressor.
<i>LinearRegressor</i>	Linear Regressor.
<i>ExtraTreesRegressor</i>	Extra Trees Regressor.
<i>RandomForestRegressor</i>	Random Forest Regressor.
<i>XGBoostRegressor</i>	XGBoost Regressor.
<i>BaselineRegressor</i>	Baseline regressor that uses a simple strategy to make predictions.

continues on next page

Table 14 – continued from previous page

<i>TimeSeriesBaselineEstimator</i>	Time series estimator that predicts using the naive forecasting approach.
<i>SklearnStackedEnsembleRegressor</i>	Scikit-learn Stacked Ensemble Regressor.
<i>DecisionTreeRegressor</i>	Decision Tree Regressor.
<i>LightGBMRegressor</i>	LightGBM Regressor.
<i>SVMRegressor</i>	Support Vector Machine Regressor.

## 5.7 Model Understanding

### 5.7.1 Utility Methods

<i>confusion_matrix</i>	Confusion matrix for binary and multiclass classification.
<i>normalize_confusion_matrix</i>	Normalizes a confusion matrix.
<i>precision_recall_curve</i>	Given labels and binary classifier predicted probabilities, compute and return the data representing a precision-recall curve.
<i>roc_curve</i>	Given labels and classifier predicted probabilities, compute and return the data representing a Receiver Operating Characteristic (ROC) curve. Works with binary or multiclass problems.
<i>calculate_permutation_importance</i>	Calculates permutation importance for features.
<i>calculate_permutation_importance_one_column</i>	Calculates permutation importance for one column in the original dataframe.
<i>binary_objective_vs_threshold</i>	Computes objective score as a function of potential binary classification
<i>get_prediction_vs_actual_over_time_data</i>	Get the data needed for the prediction_vs_actual_over_time plot.
<i>partial_dependence</i>	Calculates one or two-way partial dependence. If a single integer or
<i>get_prediction_vs_actual_data</i>	Combines <code>y_true</code> and <code>y_pred</code> into a single dataframe and adds a column for outliers. Used in <code>graph_prediction_vs_actual()</code> .
<i>get_linear_coefficients</i>	Returns a dataframe showing the features with the greatest predictive power for a linear model.
<i>t_sne</i>	Get the transformed output after fitting X to the embedded space using t-SNE.

### 5.7.2 Graph Utility Methods

<i>graph_precision_recall_curve</i>	Generate and display a precision-recall plot.
<i>graph_roc_curve</i>	Generate and display a Receiver Operating Characteristic (ROC) plot for binary and multiclass classification problems.
<i>graph_confusion_matrix</i>	Generate and display a confusion matrix plot.

continues on next page

Table 16 – continued from previous page

<i>graph_permutation_importance</i>	Generate a bar graph of the pipeline’s permutation importance.
<i>graph_binary_objective_vs_threshold</i>	Generates a plot graphing objective score vs. decision thresholds for a fitted binary classification pipeline.
<i>graph_prediction_vs_actual</i>	Generate a scatter plot comparing the true and predicted values. Used for regression plotting
<i>graph_prediction_vs_actual_over_time</i>	Plot the target values and predictions against time on the x-axis.
<i>graph_partial_dependence</i>	Create an one-way or two-way partial dependence plot. Passing a single integer or
<i>graph_t_sne</i>	Plot high dimensional data into lower dimensional space using t-SNE .

### 5.7.3 Prediction Explanations

<i>explain_predictions</i>	Creates a report summarizing the top contributing features for each data point in the input features.
<i>explain_predictions_best_worst</i>	Creates a report summarizing the top contributing features for the best and worst points in the dataset as measured by error to true labels.

## 5.8 Objectives

### 5.8.1 Objective Base Classes

<i>ObjectiveBase</i>	Base class for all objectives.
<i>BinaryClassificationObjective</i>	Base class for all binary classification objectives.
<i>MulticlassClassificationObjective</i>	Base class for all multiclass classification objectives.
<i>RegressionObjective</i>	Base class for all regression objectives.

### 5.8.2 Domain-Specific Objectives

<i>FraudCost</i>	Score the percentage of money lost of the total transaction amount process due to fraud.
<i>LeadScoring</i>	Lead scoring.
<i>CostBenefitMatrix</i>	Score using a cost-benefit matrix. Scores quantify the benefits of a given value, so greater numeric

### 5.8.3 Classification Objectives

<i>AccuracyBinary</i>	Accuracy score for binary classification.
<i>AccuracyMulticlass</i>	Accuracy score for multiclass classification.
<i>AUC</i>	AUC score for binary classification.
<i>AUCMacro</i>	AUC score for multiclass classification using macro averaging.
<i>AUCMicro</i>	AUC score for multiclass classification using micro averaging.
<i>AUCWeighted</i>	AUC Score for multiclass classification using weighted averaging.
<i>Gini</i>	Gini coefficient for binary classification.
<i>BalancedAccuracyBinary</i>	Balanced accuracy score for binary classification.
<i>BalancedAccuracyMulticlass</i>	Balanced accuracy score for multiclass classification.
<i>F1</i>	F1 score for binary classification.
<i>F1Micro</i>	F1 score for multiclass classification using micro averaging.
<i>F1Macro</i>	F1 score for multiclass classification using macro averaging.
<i>F1Weighted</i>	F1 score for multiclass classification using weighted averaging.
<i>LogLossBinary</i>	Log Loss for binary classification.
<i>LogLossMulticlass</i>	Log Loss for multiclass classification.
<i>MCCBinary</i>	Matthews correlation coefficient for binary classification.
<i>MCCMulticlass</i>	Matthews correlation coefficient for multiclass classification.
<i>Precision</i>	Precision score for binary classification.
<i>PrecisionMicro</i>	Precision score for multiclass classification using micro averaging.
<i>PrecisionMacro</i>	Precision score for multiclass classification using macro averaging.
<i>PrecisionWeighted</i>	Precision score for multiclass classification using weighted averaging.
<i>Recall</i>	Recall score for binary classification.
<i>RecallMicro</i>	Recall score for multiclass classification using micro averaging.
<i>RecallMacro</i>	Recall score for multiclass classification using macro averaging.
<i>RecallWeighted</i>	Recall score for multiclass classification using weighted averaging.

## 5.8.4 Regression Objectives

<i>R2</i>	Coefficient of determination for regression.
<i>MAE</i>	Mean absolute error for regression.
<i>MAPE</i>	Mean absolute percentage error for time series regression. Scaled by 100 to return a percentage.
<i>MSE</i>	Mean squared error for regression.
<i>MeanSquaredLogError</i>	Mean squared log error for regression.
<i>MedianAE</i>	Median absolute error for regression.
<i>MaxError</i>	Maximum residual error for regression.
<i>ExpVariance</i>	Explained variance score for regression.
<i>RootMeanSquaredError</i>	Root mean squared error for regression.
<i>RootMeanSquaredLogError</i>	Root mean squared log error for regression.

## 5.8.5 Objective Utils

<i>get_all_objective_names</i>	Get a list of the names of all objectives.
<i>get_core_objectives</i>	Returns all core objective instances associated with the given problem type.
<i>get_core_objective_names</i>	Get a list of all valid core objectives.
<i>get_non_core_objectives</i>	Get non-core objective classes.
<i>get_objective</i>	Returns the Objective class corresponding to a given objective name.

## 5.9 Problem Types

<i>handle_problem_types</i>	Handles <i>problem_type</i> by either returning the ProblemTypes or converting from a str.
<i>detect_problem_type</i>	Determine the type of problem is being solved based on the targets (binary vs multiclass classification, regression)
<i>ProblemTypes</i>	Enum defining the supported types of machine learning problems.

## 5.10 Model Family

<i>handle_model_family</i>	Handles <i>model_family</i> by either returning the ModelFamily or converting from a string
<i>ModelFamily</i>	Enum for family of machine learning models.

## 5.11 Tuners

<i>Tuner</i>	Defines API for base Tuner classes.
<i>SKOptTuner</i>	Bayesian Optimizer.
<i>GridSearchTuner</i>	Grid Search Optimizer, which generates all of the possible points to search for using a grid.
<i>RandomSearchTuner</i>	Random Search Optimizer.

## 5.12 Data Checks

### 5.12.1 Data Check Classes

<i>DataCheck</i>	Base class for all data checks. Data checks are a set of heuristics used to determine if there are problems with input data.
<i>InvalidTargetDataCheck</i>	Checks if the target data contains missing or invalid values.
<i>HighlyNullDataCheck</i>	Checks if there are any highly-null columns and rows in the input.
<i>IDColumnsDataCheck</i>	Check if any of the features are likely to be ID columns.
<i>TargetLeakageDataCheck</i>	Check if any of the features are highly correlated with the target by using mutual information or Pearson correlation.
<i>OutliersDataCheck</i>	Checks if there are any outliers in input data by using IQR to determine score anomalies. Columns with score anomalies are considered to contain outliers.
<i>NoVarianceDataCheck</i>	Check if the target or any of the features have no variance.
<i>ClassImbalanceDataCheck</i>	Check if any of the target labels are imbalanced, or if the number of values for each target are below 2 times the number of CV folds. Use for classification problems.
<i>MulticollinearityDataCheck</i>	Check if any set features are likely to be multicollinear.
<i>DateTimeNaNDataCheck</i>	Checks each column in the input for datetime features and will issue an error if NaN values are present.
<i>NaturalLanguageNaNDataCheck</i>	Checks each column in the input for natural language features and will issue an error if NaN values are present.
<i>DateTimeFormatDataCheck</i>	Checks if the datetime column has equally spaced intervals and is monotonically increasing or decreasing in order
<i>DataChecks</i>	A collection of data checks.
<i>DefaultDataChecks</i>	A collection of basic data checks that is used by AutoML by default.

### 5.12.2 Data Check Messages

<i>DataCheckMessage</i>	Base class for a message returned by a DataCheck, tagged by name.
<i>DataCheckError</i>	DataCheckMessage subclass for errors returned by data checks.
<i>DataCheckWarning</i>	DataCheckMessage subclass for warnings returned by data checks.

### 5.12.3 Data Check Message Types

<i>DataCheckMessageType</i>	Enum for type of data check message: WARNING or ERROR.
-----------------------------	--

### 5.12.4 Data Check Message Codes

<i>DataCheckMessageCode</i>	Enum for data check message code.
-----------------------------	-----------------------------------

## 5.13 Utils

### 5.13.1 General Utils

<i>import_or_raise</i>	Attempts to import the requested library by name.
<i>convert_to_seconds</i>	Converts a string describing a length of time to its length in seconds.
<i>get_random_state</i>	Generates a <code>numpy.random.RandomState</code> instance using seed.
<i>get_random_seed</i>	Given a <code>numpy.random.RandomState</code> object, generate an int representing a seed value for another random number generator. Or, if given an int, return that int.
<i>pad_with_nans</i>	Pad the beginning <code>num_to_pad</code> rows with nans.
<i>drop_rows_with_nans</i>	Drop rows that have any NaNs in all dataframes or series.
<i>infer_feature_types</i>	Create a Woodwork structure from the given list, pandas, or numpy input, with specified types for columns.
<i>save_plot</i>	Saves fig to filepath if specified, or to a default location if not.
<i>is_all_numeric</i>	Checks if the given DataFrame contains only numeric values
<i>get_importable_subclasses</i>	Get importable subclasses of a base class. Used to list all of our

## Evalml

### Subpackages

## Automl

### Subpackages

## automl\_algorithm

### Submodules

## automl\_algorithm

### Module Contents

### Classes Summary

<i>AutoMLAlgorithm</i>	Base class for the AutoML algorithms which power EvalML.
------------------------	--

---

### Exceptions Summary

## Contents

```
class evalml.automl.automl_algorithm.automl_algorithm.AutoMLAlgorithm(allowed_pipelines=None,
                                                                    cus-
                                                                    tom_hyperparameters=None,
                                                                    max_iterations=None,
                                                                    tuner_class=None,
                                                                    ran-
                                                                    dom_seed=0)
```

Base class for the AutoML algorithms which power EvalML.

This class represents an automated machine learning (AutoML) algorithm. It encapsulates the decision-making logic behind an automl search, by both deciding which pipelines to evaluate next and by deciding what set of parameters to configure the pipeline with.

To use this interface, you must define a `next_batch` method which returns the next group of pipelines to evaluate on the training data. That method may access state and results recorded from the previous batches, although that information is not tracked in a general way in this base class. Overriding `add_result` is a convenient way to record pipeline evaluation info if necessary.

#### Parameters

- **allowed\_pipelines** (*list(class)*) – A list of PipelineBase subclasses indicating the pipelines allowed in the search. The default of None indicates all pipelines for this problem type are allowed.



- **custom\_hyperparameters** (*dict*) – Custom hyperparameter ranges specified for pipelines to iterate over.
- **max\_iterations** (*int*) – The maximum number of iterations to be evaluated.
- **tuner\_class** (*class*) – A subclass of Tuner, to be used to find parameters for each pipeline. The default of None indicates the SKOptTuner will be used.
- **random\_seed** (*int*) – Seed for the random number generator. Defaults to 0.

### Methods

<code>add_result</code>	Register results from evaluating a pipeline
<code>batch_number</code>	Returns the number of batches which have been recommended so far.
<code>next_batch</code>	Get the next batch of pipelines to evaluate
<code>pipeline_number</code>	Returns the number of pipelines which have been recommended so far.

**add\_result** (*self*, *score\_to\_minimize*, *pipeline*, *trained\_pipeline\_results*)

Register results from evaluating a pipeline

### Parameters

- **score\_to\_minimize** (*float*) – The score obtained by this pipeline on the primary objective, converted so that lower values indicate better pipelines.
- **pipeline** (*PipelineBase*) – The trained pipeline object which was used to compute the score.
- **trained\_pipeline\_results** (*dict*) – Results from training a pipeline.

**property batch\_number** (*self*)

Returns the number of batches which have been recommended so far.

**abstract next\_batch** (*self*)

Get the next batch of pipelines to evaluate

**Returns** a list of instances of PipelineBase subclasses, ready to be trained and evaluated.

**Return type** list(PipelineBase)

**property pipeline\_number** (*self*)

Returns the number of pipelines which have been recommended so far.

**exception** `evalml.automl.automl_algorithm.automl_algorithm.AutoMLAlgorithmException`

Exception raised when an error is encountered during the computation of the automl algorithm

## evalml\_algorithm

### Module Contents

### Classes Summary

<code>EvalMLAlgorithm</code>	An automl algorithm that consists of two modes: fast and long, where fast is a subset of long.
------------------------------	--

## Contents

```
class evalml.automl.automl_algorithm.evalml_algorithm.EvalMLAlgorithm(X, y,
                                                                    prob-
                                                                    lem_type,
                                                                    sam-
                                                                    pler_name,
                                                                    tuner_class=None,
                                                                    ran-
                                                                    dom_seed=0,
                                                                    pipeline_params=None,
                                                                    cus-
                                                                    tom_hyperparameters=None,
                                                                    n_jobs=-
                                                                    1,
                                                                    text_in_ensembling=None,
                                                                    top_n=3,
                                                                    num_long_explore_pipelines=
                                                                    num_long_pipelines_per_batch)
```

An automl algorithm that consists of two modes: fast and long, where fast is a subset of long.

### 1. Naive pipelines:

- a. run baseline with default preprocessing pipeline
- b. run naive linear model with default preprocessing pipeline
- c. run basic RF pipeline with default preprocessing pipeline

### 2. Naive pipelines with feature selection

- a. subsequent pipelines will use the selected features with a SelectedColumns transformer

At this point we have a single pipeline candidate for preprocessing and feature selection

### 3. Pipelines with preprocessing components:

- a. scan rest of estimators (our current batch 1).

### 4. First ensembling run

Fast mode ends here. Begin long mode.

### 6. Run top 3 estimators:

- a. Generate 50 random parameter sets. Run all 150 in one batch

### 7. Second ensembling run

### 8. Repeat these indefinitely until stopping criterion is met:

- a. For each of the previous top 3 estimators, sample 10 parameters from the tuner. Run all 30 in one batch
- b. Run ensembling

## Methods

---

<code>add_result</code>	Register results from evaluating a pipeline. In batch number 2, the selected column names from the feature selector are taken to be used in a column selector. Information regarding the best pipeline is updated here as well.
<code>batch_number</code>	Returns the number of batches which have been recommended so far.
<code>next_batch</code>	Get the next batch of pipelines to evaluate
<code>pipeline_number</code>	Returns the number of pipelines which have been recommended so far.

---

**add\_result** (*self*, *score\_to\_minimize*, *pipeline*, *trained\_pipeline\_results*)

Register results from evaluating a pipeline. In batch number 2, the selected column names from the feature selector are taken to be used in a column selector. Information regarding the best pipeline is updated here as well.

#### Parameters

- **score\_to\_minimize** (*float*) – The score obtained by this pipeline on the primary objective, converted so that lower values indicate better pipelines.
- **pipeline** (*PipelineBase*) – The trained pipeline object which was used to compute the score.
- **trained\_pipeline\_results** (*dict*) – Results from training a pipeline.

**property batch\_number** (*self*)

Returns the number of batches which have been recommended so far.

**next\_batch** (*self*)

Get the next batch of pipelines to evaluate

**Returns** a list of instances of PipelineBase subclasses, ready to be trained and evaluated.

**Return type** list(PipelineBase)

**property pipeline\_number** (*self*)

Returns the number of pipelines which have been recommended so far.

## iterative\_algorithm

### Module Contents

### Classes Summary

---

<code>IterativeAlgorithm</code>	An automl algorithm which first fits a base round of pipelines with default parameters, then does a round of parameter tuning on each pipeline in order of performance.
---------------------------------	---

---

## Contents

```
class evalml.automl.automl_algorithm.iterative_algorithm.IterativeAlgorithm(allowed_pipelines=None,
max_iterations=None,
tuner_class=None,
random_seed=0,
pipelines_per_batch=5,
n_jobs=None,

1,
number_features=None,
ensembling=False,
text_in_ensembling=False,
pipeline_params=None,
custom_hyperparameters=None,
_ensemble=None,
_tuner=None,
_max_iter=None,
_tor_family_order=None)
```

An automl algorithm which first fits a base round of pipelines with default parameters, then does a round of parameter tuning on each pipeline in order of performance.

### Parameters

- **allowed\_pipelines** (*list (class)*) – A list of PipelineBase instances indicating the pipelines allowed in the search. The default of None indicates all pipelines for this problem type are allowed.
- **max\_iterations** (*int*) – The maximum number of iterations to be evaluated.
- **tuner\_class** (*class*) – A subclass of Tuner, to be used to find parameters for each pipeline. The default of None indicates the SKOptTuner will be used.
- **random\_seed** (*int*) – Seed for the random number generator. Defaults to 0.
- **pipelines\_per\_batch** (*int*) – The number of pipelines to be evaluated in each batch, after the first batch. Defaults to 5.
- **n\_jobs** (*int or None*) – Non-negative integer describing level of parallelism used for pipelines. Defaults to None.
- **number\_features** (*int*) – The number of columns in the input features. Defaults to None.
- **ensembling** (*boolean*) – If True, runs ensembling in a separate batch after every allowed pipeline class has been iterated over. Defaults to False.
- **text\_in\_ensembling** (*boolean*) – If True and ensembling is True, then n\_jobs will be set to 1 to avoid downstream sklearn stacking issues related to nltk. Defaults to None.
- **pipeline\_params** (*dict or None*) – Pipeline-level parameters that should be passed to the proposed pipelines. Defaults to None.
- **custom\_hyperparameters** (*dict or None*) – Custom hyperparameter ranges specified for pipelines to iterate over. Defaults to None.

- **`_estimator_family_order`** (*list(ModelFamily) or None*) – specify the sort order for the first batch. Defaults to `None`, which uses `_ESTIMATOR_FAMILY_ORDER`.

### Methods

<code>add_result</code>	Register results from evaluating a pipeline
<code>batch_number</code>	Returns the number of batches which have been recommended so far.
<code>next_batch</code>	Get the next batch of pipelines to evaluate
<code>pipeline_number</code>	Returns the number of pipelines which have been recommended so far.

**`add_result`** (*self, score\_to\_minimize, pipeline, trained\_pipeline\_results*)

Register results from evaluating a pipeline

### Parameters

- **`score_to_minimize`** (*float*) – The score obtained by this pipeline on the primary objective, converted so that lower values indicate better pipelines.
- **`pipeline`** (*PipelineBase*) – The trained pipeline object which was used to compute the score.
- **`trained_pipeline_results`** (*dict*) – Results from training a pipeline.

**`property batch_number`** (*self*)

Returns the number of batches which have been recommended so far.

**`next_batch`** (*self*)

Get the next batch of pipelines to evaluate

**Returns** a list of instances of `PipelineBase` subclasses, ready to be trained and evaluated.

**Return type** `list(PipelineBase)`

**`property pipeline_number`** (*self*)

Returns the number of pipelines which have been recommended so far.

## Package Contents

### Classes Summary

<code>AutoMLAlgorithm</code>	Base class for the AutoML algorithms which power EvalML.
<code>EvalMLAlgorithm</code>	An automl algorithm that consists of two modes: fast and long, where fast is a subset of long.
<code>IterativeAlgorithm</code>	An automl algorithm which first fits a base round of pipelines with default parameters, then does a round of parameter tuning on each pipeline in order of performance.

## Exceptions Summary

### Contents

```
class evalml.automl.automl_algorithm.AutoMLAlgorithm(allowed_pipelines=None, custom_hyperparameters=None, max_iterations=None, tuner_class=None, random_seed=0)
```

Base class for the AutoML algorithms which power EvalML.

This class represents an automated machine learning (AutoML) algorithm. It encapsulates the decision-making logic behind an automl search, by both deciding which pipelines to evaluate next and by deciding what set of parameters to configure the pipeline with.

To use this interface, you must define a `next_batch` method which returns the next group of pipelines to evaluate on the training data. That method may access state and results recorded from the previous batches, although that information is not tracked in a general way in this base class. Overriding `add_result` is a convenient way to record pipeline evaluation info if necessary.

#### Parameters

- **allowed\_pipelines** (*list(class)*) – A list of PipelineBase subclasses indicating the pipelines allowed in the search. The default of None indicates all pipelines for this problem type are allowed.
- **custom\_hyperparameters** (*dict*) – Custom hyperparameter ranges specified for pipelines to iterate over.
- **max\_iterations** (*int*) – The maximum number of iterations to be evaluated.
- **tuner\_class** (*class*) – A subclass of Tuner, to be used to find parameters for each pipeline. The default of None indicates the SKOptTuner will be used.
- **random\_seed** (*int*) – Seed for the random number generator. Defaults to 0.

#### Methods

<code>add_result</code>	Register results from evaluating a pipeline
<code>batch_number</code>	Returns the number of batches which have been recommended so far.
<code>next_batch</code>	Get the next batch of pipelines to evaluate
<code>pipeline_number</code>	Returns the number of pipelines which have been recommended so far.

**add\_result** (*self, score\_to\_minimize, pipeline, trained\_pipeline\_results*)

Register results from evaluating a pipeline

#### Parameters

- **score\_to\_minimize** (*float*) – The score obtained by this pipeline on the primary objective, converted so that lower values indicate better pipelines.
- **pipeline** (*PipelineBase*) – The trained pipeline object which was used to compute the score.
- **trained\_pipeline\_results** (*dict*) – Results from training a pipeline.

**property** `batch_number` (*self*)

Returns the number of batches which have been recommended so far.

**abstract** `next_batch` (*self*)

Get the next batch of pipelines to evaluate

**Returns** a list of instances of PipelineBase subclasses, ready to be trained and evaluated.

**Return type** list(PipelineBase)

**property** `pipeline_number` (*self*)

Returns the number of pipelines which have been recommended so far.

**exception** `evalml.automl.automl_algorithm.AutoMLAlgorithmException`

Exception raised when an error is encountered during the computation of the automl algorithm

**class** `evalml.automl.automl_algorithm.EvalMLAlgorithm`(*X*, *y*, *problem\_type*, *sampler\_name*, *tuner\_class*=None, *random\_seed*=0, *pipeline\_params*=None, *custom\_hyperparameters*=None, *n\_jobs*=-1, *text\_in\_ensembling*=None, *top\_n*=3, *num\_long\_explore\_pipelines*=50, *num\_long\_pipelines\_per\_batch*=10)

An automl algorithm that consists of two modes: fast and long, where fast is a subset of long.

#### 1. Naive pipelines:

- a. run baseline with default preprocessing pipeline
- b. run naive linear model with default preprocessing pipeline
- c. run basic RF pipeline with default preprocessing pipeline

#### 2. Naive pipelines with feature selection

- a. subsequent pipelines will use the selected features with a SelectedColumns transformer

At this point we have a single pipeline candidate for preprocessing and feature selection

#### 3. Pipelines with preprocessing components:

- a. scan rest of estimators (our current batch 1).

#### 4. First ensembling run

Fast mode ends here. Begin long mode.

#### 6. Run top 3 estimators:

- a. Generate 50 random parameter sets. Run all 150 in one batch

#### 7. Second ensembling run

#### 8. Repeat these indefinitely until stopping criterion is met:

- a. For each of the previous top 3 estimators, sample 10 parameters from the tuner. Run all 30 in one batch
- b. Run ensembling

### Methods

<code>add_result</code>	Register results from evaluating a pipeline. In batch number 2, the selected column names from the feature selector are taken to be used in a column selector. Information regarding the best pipeline is updated here as well.
<code>batch_number</code>	Returns the number of batches which have been recommended so far.
<code>next_batch</code>	Get the next batch of pipelines to evaluate
<code>pipeline_number</code>	Returns the number of pipelines which have been recommended so far.

**add\_result** (*self*, *score\_to\_minimize*, *pipeline*, *trained\_pipeline\_results*)

Register results from evaluating a pipeline. In batch number 2, the selected column names from the feature selector are taken to be used in a column selector. Information regarding the best pipeline is updated here as well.

#### Parameters

- **score\_to\_minimize** (*float*) – The score obtained by this pipeline on the primary objective, converted so that lower values indicate better pipelines.
- **pipeline** (*PipelineBase*) – The trained pipeline object which was used to compute the score.
- **trained\_pipeline\_results** (*dict*) – Results from training a pipeline.

**property batch\_number** (*self*)

Returns the number of batches which have been recommended so far.

**next\_batch** (*self*)

Get the next batch of pipelines to evaluate

**Returns** a list of instances of PipelineBase subclasses, ready to be trained and evaluated.

**Return type** list(PipelineBase)

**property pipeline\_number** (*self*)

Returns the number of pipelines which have been recommended so far.

```
class evalml.automl.automl_algorithm.IterativeAlgorithm (allowed_pipelines=None,  
                                                    max_iterations=None,  
                                                    tuner_class=None,  
                                                    random_seed=0,  
                                                    pipelines_per_batch=5,  
                                                    n_jobs=-1, number_features=None,  
                                                    ensembling=False,  
                                                    text_in_ensembling=False,  
                                                    pipeline_params=None,  
                                                    custom_hyperparameters=None,  
                                                    _estimator_family_order=None)
```

An automl algorithm which first fits a base round of pipelines with default parameters, then does a round of parameter tuning on each pipeline in order of performance.

#### Parameters



- **allowed\_pipelines** (*list(class)*) – A list of PipelineBase instances indicating the pipelines allowed in the search. The default of None indicates all pipelines for this problem type are allowed.
- **max\_iterations** (*int*) – The maximum number of iterations to be evaluated.
- **tuner\_class** (*class*) – A subclass of Tuner, to be used to find parameters for each pipeline. The default of None indicates the SKOptTuner will be used.
- **random\_seed** (*int*) – Seed for the random number generator. Defaults to 0.
- **pipelines\_per\_batch** (*int*) – The number of pipelines to be evaluated in each batch, after the first batch. Defaults to 5.
- **n\_jobs** (*int or None*) – Non-negative integer describing level of parallelism used for pipelines. Defaults to None.
- **number\_features** (*int*) – The number of columns in the input features. Defaults to None.
- **ensembling** (*boolean*) – If True, runs ensembling in a separate batch after every allowed pipeline class has been iterated over. Defaults to False.
- **text\_in\_ensembling** (*boolean*) – If True and ensembling is True, then n\_jobs will be set to 1 to avoid downstream sklearn stacking issues related to nltk. Defaults to None.
- **pipeline\_params** (*dict or None*) – Pipeline-level parameters that should be passed to the proposed pipelines. Defaults to None.
- **custom\_hyperparameters** (*dict or None*) – Custom hyperparameter ranges specified for pipelines to iterate over. Defaults to None.
- **\_estimator\_family\_order** (*list(ModelFamily) or None*) – specify the sort order for the first batch. Defaults to None, which uses \_ESTIMATOR\_FAMILY\_ORDER.

## Methods

<code>add_result</code>	Register results from evaluating a pipeline
<code>batch_number</code>	Returns the number of batches which have been recommended so far.
<code>next_batch</code>	Get the next batch of pipelines to evaluate
<code>pipeline_number</code>	Returns the number of pipelines which have been recommended so far.

**add\_result** (*self, score\_to\_minimize, pipeline, trained\_pipeline\_results*)

Register results from evaluating a pipeline

## Parameters

- **score\_to\_minimize** (*float*) – The score obtained by this pipeline on the primary objective, converted so that lower values indicate better pipelines.
- **pipeline** (*PipelineBase*) – The trained pipeline object which was used to compute the score.
- **trained\_pipeline\_results** (*dict*) – Results from training a pipeline.

**property batch\_number** (*self*)

Returns the number of batches which have been recommended so far.

**next\_batch** (*self*)

Get the next batch of pipelines to evaluate

**Returns** a list of instances of PipelineBase subclasses, ready to be trained and evaluated.

**Return type** list(PipelineBase)

**property pipeline\_number** (*self*)

Returns the number of pipelines which have been recommended so far.

## engine

### Submodules

### cf\_engine

### Module Contents

### Classes Summary

<i>CFClient</i>	Custom CFClient API to match Dask's CFClient and allow context management.
<i>CFComputation</i>	A Future-like wrapper around jobs created by the CFEngine.
<i>CFEngine</i>	The concurrent.futures (CF) engine

### Contents

**class** evalml.automl.engine.cf\_engine.**CFClient** (*pool*)

Custom CFClient API to match Dask's CFClient and allow context management.

#### Methods

<i>submit</i>	Pass through to imitate Dask's Client API.
---------------	--

**submit** (*self*, \*args, \*\*kwargs)

Pass through to imitate Dask's Client API.

**class** evalml.automl.engine.cf\_engine.**CFComputation** (*future*)

A Future-like wrapper around jobs created by the CFEngine.

#### Methods

<i>cancel</i>	Cancel the current computation.
<i>done</i>	<b>returns</b> Whether the computation is done.
<i>get_result</i>	Gets the computation result.

continues on next page

Table 45 – continued from previous page

<i>is_cancelled</i>	<b>returns</b> Returns whether computation was cancelled.
<hr/>	
<b>cancel</b> ( <i>self</i> )	Cancel the current computation.
<b>Returns</b>	<b>False if the call is currently being executed or finished running</b> and cannot be cancelled. True if the call can be canceled.
<b>Return type</b>	bool
<b>done</b> ( <i>self</i> )	<b>Returns</b> Whether the computation is done.
<b>Return type</b>	bool
<b>get_result</b> ( <i>self</i> )	Gets the computation result. Will block until the computation is finished.
<b>Raises</b>	<ul style="list-style-type: none"> <li>• <b>Exception</b> – If computation fails. Returns traceback.</li> <li>• <b>cf.TimeoutError</b> – If computation takes longer than default timeout time.</li> <li>• <b>cf.CancelledError</b> – If computation was canceled before completing.</li> </ul>
<b>Returns</b>	The result of the requested job.
<b>property is_cancelled</b> ( <i>self</i> )	<b>Returns</b> Returns whether computation was cancelled.
<b>Return type</b>	bool
<b>class</b> evalml.automl.engine.cf_engine. <b>CFEngine</b> ( <i>client</i> )	The concurrent.futures (CF) engine
<b>Methods</b>	
<hr/>	
<i>setup_job_log</i>	
<i>submit_evaluation_job</i>	Send evaluation job to cluster.
<i>submit_scoring_job</i>	Send scoring job to cluster.
<i>submit_training_job</i>	Send training job to cluster.
<hr/>	
<b>static</b> <b>setup_job_log</b> ()	
<b>submit_evaluation_job</b> ( <i>self</i> , <i>automl_config</i> , <i>pipeline</i> , <i>X</i> , <i>y</i> )	→ <i>evalml.automl.engine.engine_base.EngineComputation</i>
	Send evaluation job to cluster.
<b>Parameters</b>	<ul style="list-style-type: none"> <li>• <b>automl_config</b> – structure containing data passed from AutoMLSearch instance</li> <li>• <b>pipeline</b> (<i>pipeline.PipelineBase</i>) – pipeline to evaluate</li> </ul>

- **X** (*pd.DataFrame*) – input data for modeling
- **y** (*pd.Series*) – target data for modeling

**Returns**

**an object wrapping a reference to a future-like computation** occurring in the resource pool

**Return type** *CFComputation*

**submit\_scoring\_job** (*self*, *automl\_config*, *pipeline*, *X*, *y*, *objectives*) → *evalml.automl.engine.engine\_base.EngineComputation*

Send scoring job to cluster.

**Parameters**

- **automl\_config** – structure containing data passed from AutoMLSearch instance
- **pipeline** (*pipeline.PipelineBase*) – pipeline to train
- **X** (*pd.DataFrame*) – input data for modeling
- **y** (*pd.Series*) – target data for modeling

**Returns**

**a object wrapping a reference to a future-like computation** occurring in the resource pool

**Return type** *CFComputation*

**submit\_training\_job** (*self*, *automl\_config*, *pipeline*, *X*, *y*) → *evalml.automl.engine.engine\_base.EngineComputation*

Send training job to cluster.

**Parameters**

- **automl\_config** – structure containing data passed from AutoMLSearch instance
- **pipeline** (*pipeline.PipelineBase*) – pipeline to train
- **X** (*pd.DataFrame*) – input data for modeling
- **y** (*pd.Series*) – target data for modeling

**Returns**

**an object wrapping a reference to a future-like computation** occurring in the resource pool

**Return type** *CFComputation*

## dask\_engine

### Module Contents

### Classes Summary

---

<i>DaskComputation</i>	A Future-like wrapper around jobs created by the DaskEngine.
<i>DaskEngine</i>	The dask engine

---

## Contents

**class** evalml.engine.dask\_engine.**DaskComputation** (*dask\_future*)

A Future-like wrapper around jobs created by the DaskEngine.

**Parameters** **dask\_future** (*callable*) – Computation to do.

### Methods

<i>cancel</i>	Cancel the current computation.
<i>done</i>	<b>returns</b> Whether the computation is done.
<i>get_result</i>	Gets the computation result.
<i>is_cancelled</i>	<b>returns</b> Returns whether computation was cancelled.

**cancel** (*self*)

Cancel the current computation.

**done** (*self*)

**Returns** Whether the computation is done.

**Return type** bool

**get\_result** (*self*)

Gets the computation result. Will block until the computation is finished.

**Raises Exception** – If computation fails. Returns traceback.

**property is\_cancelled** (*self*)

**Returns** Returns whether computation was cancelled.

**Return type** bool

**class** evalml.engine.dask\_engine.**DaskEngine** (*client*)

The dask engine

### Methods

<i>send_data_to_cluster</i>	Send data to the cluster.
<i>setup_job_log</i>	
<i>submit_evaluation_job</i>	Send evaluation job to cluster.
<i>submit_scoring_job</i>	Send scoring job to cluster.
<i>submit_training_job</i>	Send training job to cluster.

**send\_data\_to\_cluster** (*self*, *X*, *y*)

Send data to the cluster.

The implementation uses caching so the data is only sent once. This follows dask best practices.

### Parameters

- **X** (*pd.DataFrame*) – input data for modeling

- **y** (*pd.Series*) – target data for modeling

**Returns** the modeling data

**Return type** *dask.Future*

**static setup\_job\_log()**

**submit\_evaluation\_job** (*self*, *automl\_config*, *pipeline*, *X*, *y*) →  
*evalml.automl.engine.engine\_base.EngineComputation*

Send evaluation job to cluster.

**Parameters**

- **automl\_config** – structure containing data passed from AutoMLSearch instance
- **pipeline** (*pipeline.PipelineBase*) – pipeline to evaluate
- **X** (*pd.DataFrame*) – input data for modeling
- **y** (*pd.Series*) – target data for modeling

**Returns**

**a object wrapping a reference to a future-like computation** occurring in the dask cluster

**Return type** *DaskComputation*

**submit\_scoring\_job** (*self*, *automl\_config*, *pipeline*, *X*, *y*, *objectives*) →  
*evalml.automl.engine.engine\_base.EngineComputation*

Send scoring job to cluster.

**Parameters**

- **automl\_config** – structure containing data passed from AutoMLSearch instance
- **pipeline** (*pipeline.PipelineBase*) – pipeline to train
- **X** (*pd.DataFrame*) – input data for modeling
- **y** (*pd.Series*) – target data for modeling

**Returns**

**a object wrapping a reference to a future-like computation** occurring in the dask cluster

**Return type** *DaskComputation*

**submit\_training\_job** (*self*, *automl\_config*, *pipeline*, *X*, *y*) →  
*evalml.automl.engine.engine\_base.EngineComputation*

Send training job to cluster.

**Parameters**

- **automl\_config** – structure containing data passed from AutoMLSearch instance
- **pipeline** (*pipeline.PipelineBase*) – pipeline to train
- **X** (*pd.DataFrame*) – input data for modeling
- **y** (*pd.Series*) – target data for modeling

**Returns**

**a object wrapping a reference to a future-like computation** occurring in the dask cluster

**Return type** *DaskComputation*

## engine\_base

### Module Contents

### Classes Summary

<i>EngineBase</i>	Helper class that provides a standard way to create an ABC using
<i>EngineComputation</i>	Wrapper around the result of a (possibly asynchronous) engine computation.
<i>JobLogger</i>	Mimics the behavior of a python logging.Logger but stores all messages rather than actually logging them.

### Functions

<i>evaluate_pipeline</i>	Function submitted to the submit_evaluation_job engine method.
<i>score_pipeline</i>	Wrapper around pipeline.score method to make it easy to score pipelines with dask.
<i>train_and_score_pipeline</i>	Given a pipeline, config and data, train and score the pipeline and return the CV or TV scores
<i>train_pipeline</i>	Train a pipeline and tune the threshold if necessary.

### Contents

**class** evalml.automl.engine.engine\_base.**EngineBase**

Helper class that provides a standard way to create an ABC using inheritance.

#### Methods

<i>setup_job_log</i>	
<i>submit_evaluation_job</i>	Submit job for pipeline evaluation during AutoMLSearch.
<i>submit_scoring_job</i>	Submit job for pipeline scoring.
<i>submit_training_job</i>	Submit job for pipeline training.

**static** **setup\_job\_log()**

**abstract** **submit\_evaluation\_job** (*self*, *automl\_config*, *pipeline*, *X*, *y*)  
Submit job for pipeline evaluation during AutoMLSearch.

**abstract** **submit\_scoring\_job** (*self*, *automl\_config*, *pipeline*, *X*, *y*, *objectives*)  
Submit job for pipeline scoring.

**abstract** **submit\_training\_job** (*self*, *automl\_config*, *pipeline*, *X*, *y*)  
Submit job for pipeline training.

**class** evalml.automl.engine.engine\_base.**EngineComputation**

Wrapper around the result of a (possibly asynchronous) engine computation.

## Methods

<code>cancel</code>	Cancel the computation.
<code>done</code>	Whether the computation is done.
<code>get_result</code>	Gets the computation result.

**abstract** `cancel(self)`

Cancel the computation.

**abstract** `done(self)`

Whether the computation is done.

**abstract** `get_result(self)`

Gets the computation result. Will block until the computation is finished.

Raises Exception: If computation fails. Returns traceback.

`evalml.automl.engine.engine_base.evaluate_pipeline(pipeline, automl_config, X, y, logger)`

Function submitted to the `submit_evaluation_job` engine method.

## Parameters

- **pipeline** (*PipelineBase*) – The pipeline to score
- **automl\_config** (*AutoMLConfig*) – The AutoMLSearch object, used to access config and the error callback
- **X** (*pd.DataFrame*) – Training features
- **y** (*pd.Series*) – Training target

## Returns

First - A dict containing `cv_score_mean`, `cv_scores`, `training_time` and a `cv_data` structure with details.

Second - The pipeline class we trained and scored. Third - the job logger instance with all the recorded messages.

**Return type** tuple of three items

**class** `evalml.automl.engine.engine_base.JobLogger`

Mimics the behavior of a python logging.Logger but stores all messages rather than actually logging them.

This is used during engine jobs so that log messages are recorded after the job completes. This is desired so that all of the messages for a single job are grouped together in the log.

## Methods

<code>debug</code>	Store message at the debug level.
<code>error</code>	Store message at the error level.
<code>info</code>	Store message at the info level.
<code>warning</code>	Store message at the warning level.
<code>write_to_logger</code>	Write all the messages to the logger. First In First Out order.

**debug** (*self, msg*)

Store message at the debug level.

**error** (*self, msg*)

Store message at the error level.



**info** (*self, msg*)

Store message at the info level.

**warning** (*self, msg*)

Store message at the warning level.

**write\_to\_logger** (*self, logger*)

Write all the messages to the logger. First In First Out order.

`evalml automl engine engine_base .score_pipeline` (*pipeline, X, y, objectives, X\_schema=None, y\_schema=None*)

Wrapper around pipeline.score method to make it easy to score pipelines with dask.

Arguments: pipeline (PipelineBase): The pipeline to score. X (pd.DataFrame): Features to score on. y (pd.Series): Target used to calculate scores. X\_schema (ww.TableSchema): Schema for features. y\_schema (ww.ColumnSchema): Schema for columns.

**Returns** dict containing pipeline scores.

`evalml automl engine engine_base .train_and_score_pipeline` (*pipeline, automl\_config, full\_X\_train, full\_y\_train, logger*)

Given a pipeline, config and data, train and score the pipeline and return the CV or TV scores

#### Parameters

- **pipeline** (*PipelineBase*) – The pipeline to score
- **automl\_config** (*AutoMLSearch*) – The AutoMLSearch object, used to access config and the error callback
- **full\_X\_train** (*pd.DataFrame*) – Training features
- **full\_y\_train** (*pd.Series*) – Training target

#### Returns

First - A dict containing **cv\_score\_mean**, **cv\_scores**, **training\_time** and a **cv\_data** structure with details. Second - The pipeline class we trained and scored. Third - the job logger instance with all the recorded messages.

**Return type** tuple of three items

`evalml automl engine engine_base .train_pipeline` (*pipeline, X, y, automl\_config, schema=True*)

Train a pipeline and tune the threshold if necessary.

#### Parameters

- **pipeline** (*PipelineBase*) – Pipeline to train.
- **X** (*pd.DataFrame*) – Features to train on.
- **y** (*pd.Series*) – Target to train on.
- **automl\_config** (*AutoMLSearch*) – The AutoMLSearch object, used to access config and the error callback
- **schema** (*bool*) – Whether to use the schemas for X and y

**Returns** trained pipeline.

**Return type** pipeline (PipelineBase)

## sequential\_engine

### Module Contents

### Classes Summary

<i>SequentialComputation</i>	A Future-like api for jobs created by the SequentialEngine, an Engine that sequentially computes the submitted jobs.
<i>SequentialEngine</i>	The default engine for the AutoML search. Trains and scores pipelines locally and sequentially.

### Contents

**class** evalml.automl.engine.sequential\_engine.**SequentialComputation** (*work*,  
\*\**kwargs*)

A Future-like api for jobs created by the SequentialEngine, an Engine that sequentially computes the submitted jobs.

In order to separate the engine from the AutoMLSearch loop, we need the sequential computations to behave the same way as concurrent computations from AutoMLSearch’s point-of-view. One way to do this is by delaying the computation in the sequential engine until `get_result` is called. Since AutoMLSearch will call `get_result` only when the computation is “done”, by always returning `True` in `done()` we make sure that `get_result` is called in the order that the jobs are submitted. So the computations happen sequentially!

**Parameters** *work* (*callable*) – Computation that should be done by the engine.

#### Methods

<i>cancel</i>	Cancel the current computation.
<i>done</i>	Whether the computation is done.
<i>get_result</i>	Gets the computation result.

**cancel** (*self*)

Cancel the current computation.

**done** (*self*)

Whether the computation is done.

**get\_result** (*self*)

Gets the computation result. Will block until the computation is finished.

Raises Exception: If computation fails. Returns traceback.

**class** evalml.automl.engine.sequential\_engine.**SequentialEngine**

The default engine for the AutoML search. Trains and scores pipelines locally and sequentially.

#### Methods

<i>setup_job_log</i>	
<i>submit_evaluation_job</i>	Submit job for pipeline evaluation during AutoMLSearch.

continues on next page

Table 57 – continued from previous page

<code>submit_scoring_job</code>	Submit job for pipeline scoring.
<code>submit_training_job</code>	Submit job for pipeline training.

**static** `setup_job_log()`

**submit\_evaluation\_job** (*self*, *automl\_config*, *pipeline*, *X*, *y*)

Submit job for pipeline evaluation during AutoMLSearch.

**submit\_scoring\_job** (*self*, *automl\_config*, *pipeline*, *X*, *y*, *objectives*)

Submit job for pipeline scoring.

**submit\_training\_job** (*self*, *automl\_config*, *pipeline*, *X*, *y*)

Submit job for pipeline training.

## Package Contents

### Classes Summary

<code>CFEngine</code>	The concurrent.futures (CF) engine
<code>DaskEngine</code>	The dask engine
<code>EngineBase</code>	Helper class that provides a standard way to create an ABC using
<code>EngineComputation</code>	Wrapper around the result of a (possibly asynchronous) engine computation.
<code>SequentialEngine</code>	The default engine for the AutoML search. Trains and scores pipelines locally and sequentially.

### Functions

<code>evaluate_pipeline</code>	Function submitted to the <code>submit_evaluation_job</code> engine method.
<code>train_and_score_pipeline</code>	Given a pipeline, config and data, train and score the pipeline and return the CV or TV scores
<code>train_pipeline</code>	Train a pipeline and tune the threshold if necessary.

### Contents

**class** `evalml.automl.engine.CFEngine` (*client*)

The concurrent.futures (CF) engine

#### Methods

<code>setup_job_log</code>	
<code>submit_evaluation_job</code>	Send evaluation job to cluster.
<code>submit_scoring_job</code>	Send scoring job to cluster.
<code>submit_training_job</code>	Send training job to cluster.

```
static setup_job_log()
```

```
submit_evaluation_job(self, automl_config, pipeline, X, y) →  
evalml.automl.engine.engine_base.EngineComputation
```

Send evaluation job to cluster.

#### Parameters

- **automl\_config** – structure containing data passed from AutoMLSearch instance
- **pipeline** (*pipeline.PipelineBase*) – pipeline to evaluate
- **X** (*pd.DataFrame*) – input data for modeling
- **y** (*pd.Series*) – target data for modeling

#### Returns

**an object wrapping a reference to a future-like computation** occurring in the resource pool

**Return type** CFComputation

```
submit_scoring_job(self, automl_config, pipeline, X, y, objectives) →  
evalml.automl.engine.engine_base.EngineComputation
```

Send scoring job to cluster.

#### Parameters

- **automl\_config** – structure containing data passed from AutoMLSearch instance
- **pipeline** (*pipeline.PipelineBase*) – pipeline to train
- **X** (*pd.DataFrame*) – input data for modeling
- **y** (*pd.Series*) – target data for modeling

#### Returns

**a object wrapping a reference to a future-like computation** occurring in the resource pool

**Return type** CFComputation

```
submit_training_job(self, automl_config, pipeline, X, y) →  
evalml.automl.engine.engine_base.EngineComputation
```

Send training job to cluster.

#### Parameters

- **automl\_config** – structure containing data passed from AutoMLSearch instance
- **pipeline** (*pipeline.PipelineBase*) – pipeline to train
- **X** (*pd.DataFrame*) – input data for modeling
- **y** (*pd.Series*) – target data for modeling

#### Returns

**an object wrapping a reference to a future-like computation** occurring in the resource pool

**Return type** CFComputation

```
class evalml.automl.engine.DaskEngine(client)
```

The dask engine

#### Methods

<code>send_data_to_cluster</code>	Send data to the cluster.
<code>setup_job_log</code>	
<code>submit_evaluation_job</code>	Send evaluation job to cluster.
<code>submit_scoring_job</code>	Send scoring job to cluster.
<code>submit_training_job</code>	Send training job to cluster.

**send\_data\_to\_cluster** (*self*, *X*, *y*)

Send data to the cluster.

The implementation uses caching so the data is only sent once. This follows dask best practices.

#### Parameters

- **X** (*pd.DataFrame*) – input data for modeling
- **y** (*pd.Series*) – target data for modeling

**Returns** the modeling data

**Return type** *dask.Future*

**static setup\_job\_log** ()

**submit\_evaluation\_job** (*self*, *automl\_config*, *pipeline*, *X*, *y*) →  
*evalml.automl.engine.engine\_base.EngineComputation*

Send evaluation job to cluster.

#### Parameters

- **automl\_config** – structure containing data passed from AutoMLSearch instance
- **pipeline** (*pipeline.PipelineBase*) – pipeline to evaluate
- **X** (*pd.DataFrame*) – input data for modeling
- **y** (*pd.Series*) – target data for modeling

#### Returns

**a object wrapping a reference to a future-like computation** occurring in the dask cluster

**Return type** *DaskComputation*

**submit\_scoring\_job** (*self*, *automl\_config*, *pipeline*, *X*, *y*, *objectives*) →  
*evalml.automl.engine.engine\_base.EngineComputation*

Send scoring job to cluster.

#### Parameters

- **automl\_config** – structure containing data passed from AutoMLSearch instance
- **pipeline** (*pipeline.PipelineBase*) – pipeline to train
- **X** (*pd.DataFrame*) – input data for modeling
- **y** (*pd.Series*) – target data for modeling

#### Returns

**a object wrapping a reference to a future-like computation** occurring in the dask cluster

**Return type** *DaskComputation*

**submit\_training\_job** (*self*, *automl\_config*, *pipeline*, *X*, *y*) →  
*evalml.automl.engine.engine\_base.EngineComputation*

Send training job to cluster.

#### Parameters

- **automl\_config** – structure containing data passed from AutoMLSearch instance
- **pipeline** (*pipeline.PipelineBase*) – pipeline to train
- **X** (*pd.DataFrame*) – input data for modeling
- **y** (*pd.Series*) – target data for modeling

#### Returns

a object wrapping a reference to a future-like computation occurring in the dask cluster

**Return type** DaskComputation

**class** *evalml.automl.engine.EngineBase*

Helper class that provides a standard way to create an ABC using inheritance.

#### Methods

---

*setup\_job\_log*

---

*submit\_evaluation\_job*

Submit job for pipeline evaluation during AutoMLSearch.

---

*submit\_scoring\_job*

Submit job for pipeline scoring.

---

*submit\_training\_job*

Submit job for pipeline training.

---

**static** *setup\_job\_log*()

**abstract** *submit\_evaluation\_job* (*self*, *automl\_config*, *pipeline*, *X*, *y*)

Submit job for pipeline evaluation during AutoMLSearch.

**abstract** *submit\_scoring\_job* (*self*, *automl\_config*, *pipeline*, *X*, *y*, *objectives*)

Submit job for pipeline scoring.

**abstract** *submit\_training\_job* (*self*, *automl\_config*, *pipeline*, *X*, *y*)

Submit job for pipeline training.

**class** *evalml.automl.engine.EngineComputation*

Wrapper around the result of a (possibly asynchronous) engine computation.

#### Methods

---

*cancel*

Cancel the computation.

---

*done*

Whether the computation is done.

---

*get\_result*

Gets the computation result.

---

**abstract** *cancel* (*self*)

Cancel the computation.

**abstract** *done* (*self*)

Whether the computation is done.

**abstract** *get\_result* (*self*)

Gets the computation result. Will block until the computation is finished.

Raises Exception: If computation fails. Returns traceback.

`evalml automl engine evaluate_pipeline (pipeline, automl_config, X, y, logger)`

Function submitted to the `submit_evaluation_job` engine method.

#### Parameters

- **pipeline** (*PipelineBase*) – The pipeline to score
- **automl\_config** (*AutoMLConfig*) – The AutoMLSearch object, used to access config and the error callback
- **X** (*pd.DataFrame*) – Training features
- **y** (*pd.Series*) – Training target

#### Returns

**First** - A dict containing `cv_score_mean`, `cv_scores`, `training_time` and a `cv_data` structure with details.  
**Second** - The pipeline class we trained and scored. **Third** - the job logger instance with all the recorded messages.

**Return type** tuple of three items

**class** `evalml automl engine SequentialEngine`

The default engine for the AutoML search. Trains and scores pipelines locally and sequentially.

#### Methods

---

`setup_job_log`

---

<code>submit_evaluation_job</code>	Submit job for pipeline evaluation during AutoMLSearch.
<code>submit_scoring_job</code>	Submit job for pipeline scoring.
<code>submit_training_job</code>	Submit job for pipeline training.

---

**static** `setup_job_log()`

**submit\_evaluation\_job** (*self*, *automl\_config*, *pipeline*, *X*, *y*)

Submit job for pipeline evaluation during AutoMLSearch.

**submit\_scoring\_job** (*self*, *automl\_config*, *pipeline*, *X*, *y*, *objectives*)

Submit job for pipeline scoring.

**submit\_training\_job** (*self*, *automl\_config*, *pipeline*, *X*, *y*)

Submit job for pipeline training.

`evalml automl engine train_and_score_pipeline (pipeline, automl_config, full_X_train, full_y_train, logger)`

Given a pipeline, config and data, train and score the pipeline and return the CV or TV scores

#### Parameters

- **pipeline** (*PipelineBase*) – The pipeline to score
- **automl\_config** (*AutoMLSearch*) – The AutoMLSearch object, used to access config and the error callback
- **full\_X\_train** (*pd.DataFrame*) – Training features
- **full\_y\_train** (*pd.Series*) – Training target

#### Returns

**First** - A dict containing `cv_score_mean`, `cv_scores`, `training_time` and a `cv_data` structure with details.  
**Second** - The pipeline class we trained and scored. **Third** - the job logger instance with all

the recorded messages.

**Return type** tuple of three items

`evalml.engine.train_pipeline(pipeline, X, y, automl_config, schema=True)`

Train a pipeline and tune the threshold if necessary.

#### Parameters

- **pipeline** (*PipelineBase*) – Pipeline to train.
- **x** (*pd.DataFrame*) – Features to train on.
- **y** (*pd.Series*) – Target to train on.
- **automl\_config** (*AutoMLSearch*) – The AutoMLSearch object, used to access config and the error callback
- **schema** (*bool*) – Whether to use the schemas for X and y

**Returns** trained pipeline.

**Return type** pipeline (*PipelineBase*)

## Submodules

`automl_search`

## Module Contents

## Classes Summary

<i>AutoMLSearch</i>	Automated Pipeline search.
---------------------	----------------------------

## Functions

<i>search</i>	Given data and configuration, run an automl search.
---------------	---

## Attributes Summary

<i>logger</i>
---------------



## Contents

```
class evalml.automl.automl_search.AutoMLSearch(X_train=None, y_train=None,
                                              problem_type=None, objective='auto',
                                              max_iterations=None, max_time=None,
                                              patience=None, tolerance=None, data_splitter=None,
                                              allowed_component_graphs=None,
                                              allowed_model_families=None,
                                              start_iteration_callback=None,
                                              add_result_callback=None,
                                              error_callback=None,
                                              additional_objectives=None,
                                              alternate_thresholding_objective='F1',
                                              random_seed=0, n_jobs=-1,
                                              tuner_class=None, optimize_thresholds=True,
                                              ensembling=False, max_batches=None,
                                              problem_configuration=None,
                                              train_best_pipeline=True,
                                              pipeline_parameters=None,
                                              custom_hyperparameters=None,
                                              sampler_method='auto',
                                              sampler_balanced_ratio=0.25,
                                              _ensembling_split_size=0.2,
                                              _pipelines_per_batch=5, engine=None)
```

Automated Pipeline search.

### Parameters

- **X\_train** (*pd.DataFrame*) – The input training data of shape [n\_samples, n\_features]. Required.
- **y\_train** (*pd.Series*) – The target training data of length [n\_samples]. Required for supervised learning tasks.
- **problem\_type** (*str or ProblemTypes*) – type of supervised learning problem. See `evalml.problem_types.ProblemType.all_problem_types` for a full list.
- **objective** (*str, ObjectiveBase*) – The objective to optimize for. Used to propose and rank pipelines, but not for optimizing each pipeline during fit-time. When set to ‘auto’, chooses:
  - LogLossBinary for binary classification problems,
  - LogLossMulticlass for multiclass classification problems, and
  - R2 for regression problems.
- **max\_iterations** (*int*) – Maximum number of iterations to search. If max\_iterations and max\_time is not set, then max\_iterations will default to max\_iterations of 5.
- **max\_time** (*int, str*) – Maximum time to search for pipelines. This will not start a new pipeline search after the duration has elapsed. If it is an integer, then the time will be in seconds. For strings, time can be specified as seconds, minutes, or hours.
- **patience** (*int*) – Number of iterations without improvement to stop search early. Must be positive. If None, early stopping is disabled. Defaults to None.

- **tolerance** (*float*) – Minimum percentage difference to qualify as score improvement for early stopping. Only applicable if patience is not None. Defaults to None.
- **allowed\_component\_graphs** (*dict*) – A dictionary of lists or ComponentGraphs indicating the component graphs allowed in the search. The format should follow { “Name\_0”: [list\_of\_components], “Name\_1”: [ComponentGraph(...)] }

The default of None indicates all pipeline component graphs for this problem type are allowed. Setting this field will cause allowed\_model\_families to be ignored.

e.g. allowed\_component\_graphs = { “My\_Graph”: [“Imputer”, “One Hot Encoder”, “Random Forest Classifier”] }

- **allowed\_model\_families** (*list(str, ModelFamily)*) – The model families to search. The default of None searches over all model families. Run evalml.pipelines.components.utils.allowed\_model\_families(“binary”) to see options. Change *binary* to *multiclass* or *regression* depending on the problem type. Note that if allowed\_pipelines is provided, this parameter will be ignored.
- **data\_splitter** (*sklearn.model\_selection.BaseCrossValidator*) – Data splitting method to use. Defaults to StratifiedKFold.
- **tuner\_class** – The tuner class to use. Defaults to SKOptTuner.
- **optimize\_thresholds** (*bool*) – Whether or not to optimize the binary pipeline threshold. Defaults to True.
- **start\_iteration\_callback** (*callable*) – Function called before each pipeline training iteration. Callback function takes three positional parameters: The pipeline instance and the AutoMLSearch object.
- **add\_result\_callback** (*callable*) – Function called after each pipeline training iteration. Callback function takes three positional parameters: A dictionary containing the training results for the new pipeline, an untrained\_pipeline containing the parameters used during training, and the AutoMLSearch object.
- **error\_callback** (*callable*) – Function called when *search()* errors and raises an Exception. Callback function takes three positional parameters: the Exception raised, the traceback, and the AutoMLSearch object. Must also accepts kwargs, so AutoMLSearch is able to pass along other appropriate parameters by default. Defaults to None, which will call *log\_error\_callback*.
- **additional\_objectives** (*list*) – Custom set of objectives to score on. Will override default objectives for problem type if not empty.
- **alternate\_thresholding\_objective** (*str*) – The objective to use for thresholding binary classification pipelines if the main objective provided isn’t tuneable. Defaults to F1.
- **random\_seed** (*int*) – Seed for the random number generator. Defaults to 0.
- **n\_jobs** (*int or None*) – Non-negative integer describing level of parallelism used for pipelines. None and 1 are equivalent. If set to -1, all CPUs are used. For n\_jobs below -1, (n\_cpus + 1 + n\_jobs) are used.
- **ensembling** (*boolean*) – If True, runs ensembling in a separate batch after every allowed pipeline class has been iterated over. If the number of unique pipelines to search over per batch is one, ensembling will not run. Defaults to False.
- **max\_batches** (*int*) – The maximum number of batches of pipelines to search. Parameters max\_time, and max\_iterations have precedence over stopping the search.

- **problem\_configuration** (*dict*, *None*) – Additional parameters needed to configure the search. For example, in time series problems, values should be passed in for the `date_index`, `gap`, and `max_delay` variables.
- **train\_best\_pipeline** (*boolean*) – Whether or not to train the best pipeline before returning it. Defaults to `True`.
- **pipeline\_parameters** (*dict*) – A dict of the parameters used to initialize a pipeline with. Keys should consist of the component names and values should specify parameter values  
e.g. `pipeline_parameters = { 'Imputer' : { 'numeric_impute_strategy': 'most_frequent' } }`
- **custom\_hyperparameters** (*dict*) – A dict of the hyperparameter ranges used to iterate over during search. Keys should consist of the component names and values should specify a singular value or `skopt.Space`.  
e.g. `custom_hyperparameters = { 'Imputer' : { 'numeric_impute_strategy': Categorical(['most_frequent', 'median']) } }`
- **sampler\_method** (*str*) – The data sampling component to use in the pipelines if the problem type is classification and the target balance is smaller than the `sampler_balanced_ratio`. Either `'auto'`, which will use our preferred sampler for the data, `'Undersampler'`, `'Oversampler'`, or `None`. Defaults to `'auto'`.
- **sampler\_balanced\_ratio** (*float*) – The minority:majority class ratio that we consider balanced, so a 1:4 ratio would be equal to 0.25. If the class balance is larger than this provided value, then we will not add a sampler since the data is then considered balanced. Overrides the `sampler_ratio` of the samplers. Defaults to 0.25.
- **\_ensembling\_split\_size** (*float*) – The amount of the training data we'll set aside for training ensemble metalearners. Only used when `ensembling` is `True`. Must be between 0 and 1, exclusive. Defaults to 0.2
- **\_pipelines\_per\_batch** (*int*) – The number of pipelines to train for every batch after the first one. The first batch will train a baseline pipeline + one of each pipeline family allowed in the search.
- **engine** (*EngineBase* or *None*) – The engine instance used to evaluate pipelines. If `None`, a `SequentialEngine` will be used.

## Methods

<code>add_to_rankings</code>	Fits and evaluates a given pipeline then adds the results to the automl rankings with the requirement that automl search has been run.
<code>best_pipeline</code>	Returns a trained instance of the best pipeline and parameters found during automl search. If <code>train_best_pipeline</code> is set to <code>False</code> , returns an untrained pipeline instance.
<code>describe_pipeline</code>	Describe a pipeline
<code>full_rankings</code>	Returns a <code>pandas.DataFrame</code> with scoring results from all pipelines searched
<code>get_pipeline</code>	Given the ID of a pipeline training result, returns an untrained instance of the specified pipeline
<code>load</code>	Loads AutoML object at file path
<code>plot</code>	

continues on next page

Table 68 – continued from previous page

<i>rankings</i>	Returns a pandas.DataFrame with scoring results from the highest-scoring set of parameters used with each pipeline.
<i>results</i>	Class that allows access to a copy of the results from <i>automl_search</i> .
<i>save</i>	Saves AutoML object at file path
<i>score_pipelines</i>	Score a list of pipelines on the given holdout data.
<i>search</i>	Find the best pipeline for the data set.
<i>train_pipelines</i>	Train a list of pipelines on the training data.

**add\_to\_rankings** (*self*, *pipeline*)

Fits and evaluates a given pipeline then adds the results to the automl rankings with the requirement that automl search has been run.

**Parameters** *pipeline* (*PipelineBase*) – pipeline to train and evaluate.

**property best\_pipeline** (*self*)

Returns a trained instance of the best pipeline and parameters found during automl search. If *train\_best\_pipeline* is set to False, returns an untrained pipeline instance.

**Returns** A trained instance of the best pipeline and parameters found during automl search. If *train\_best\_pipeline* is set to False, returns an untrained pipeline instance.

**Return type** *PipelineBase*

**describe\_pipeline** (*self*, *pipeline\_id*, *return\_dict=False*)

Describe a pipeline

**Parameters**

- **pipeline\_id** (*int*) – pipeline to describe
- **return\_dict** (*bool*) – If True, return dictionary of information about pipeline. Defaults to False.

**Returns** Description of specified pipeline. Includes information such as type of pipeline components, problem, training time, cross validation, etc.

**property full\_rankings** (*self*)

Returns a pandas.DataFrame with scoring results from all pipelines searched

**get\_pipeline** (*self*, *pipeline\_id*)

Given the ID of a pipeline training result, returns an untrained instance of the specified pipeline initialized with the parameters used to train that pipeline during automl search.

**Parameters** *pipeline\_id* (*int*) – pipeline to retrieve

**Returns** untrained pipeline instance associated with the provided ID

**Return type** *PipelineBase*

**static load** (*file\_path*, *pickle\_type='cloudpickle'*)

Loads AutoML object at file path

**Parameters**

- **file\_path** (*str*) – location to find file to load
- **{ "pickle" }** (*pickle\_type*) – the pickling library to use. Currently not used since the standard pickle library can handle cloudpickles.

- **"cloudpickle"** – the pickling library to use. Currently not used since the standard pickle library can handle cloudpickles.

**Returns** AutoSearchBase object

**property plot** (*self*)

**property rankings** (*self*)

Returns a pandas.DataFrame with scoring results from the highest-scoring set of parameters used with each pipeline.

**property results** (*self*)

Class that allows access to a copy of the results from *automl\_search*.

**Returns:** dict containing *pipeline\_results*: a dict with results from each pipeline, and *search\_order*: a list describing the order the pipelines were searched.

**save** (*self*, *file\_path*, *pickle\_type*='cloudpickle', *pickle\_protocol*=cloudpickle.DEFAULT\_PROTOCOL)

Saves AutoML object at file path

**Parameters**

- **file\_path** (*str*) – location to save file
- **{"pickle"}** (*pickle\_type*) – the pickling library to use.
- **"cloudpickle"** – the pickling library to use.
- **pickle\_protocol** (*int*) – the pickle data stream format.

**Returns** None

**score\_pipelines** (*self*, *pipelines*, *X\_holdout*, *y\_holdout*, *objectives*)

Score a list of pipelines on the given holdout data.

**Parameters**

- **pipelines** (*list* (*PipelineBase*)) – List of pipelines to train.
- **X\_holdout** (*pd.DataFrame*) – Holdout features.
- **y\_holdout** (*pd.Series*) – Holdout targets for scoring.
- **objectives** (*list* (*str*), *list* (*ObjectiveBase*)) – Objectives used for scoring.

**Returns** Dictionary keyed by pipeline name that maps to a dictionary of scores. Note that the any pipelines that error out during scoring will not be included in the dictionary but the exception and stacktrace will be displayed in the log.

**Return type** Dict[str, Dict[str, float]]

**search** (*self*, *show\_iteration\_plot*=True)

Find the best pipeline for the data set.

**Parameters**

- **feature\_types** (*list*, *optional*) – list of feature types, either numerical or categorical. Categorical features will automatically be encoded
- **show\_iteration\_plot** (*boolean*, *True*) – Shows an iteration vs. score plot in Jupyter notebook. Disabled by default in non-Jupyter environments.

**train\_pipelines** (*self*, *pipelines*)

Train a list of pipelines on the training data.

This can be helpful for training pipelines once the search is complete.

**Parameters** `pipelines` (*list* (*PipelineBase*)) – List of pipelines to train.

**Returns** Dictionary keyed by pipeline name that maps to the fitted pipeline. Note that the any pipelines that error out during training will not be included in the dictionary but the exception and stacktrace will be displayed in the log.

**Return type** Dict[str, PipelineBase]

`evalml automl automl_search logger`

`evalml automl automl_search search` (*X\_train=None, y\_train=None, problem\_type=None, objective='auto', problem\_configuration=None, \*\*kwargs*)

Given data and configuration, run an automl search.

This method will run EvalML's default suite of data checks. If the data checks produce errors, the data check results will be returned before running the automl search. In that case we recommend you alter your data to address these errors and try again.

This method is provided for convenience. If you'd like more control over when each of these steps is run, consider making calls directly to the various pieces like the data checks and AutoMLSearch, instead of using this method.

#### Parameters

- **X\_train** (*pd.DataFrame*) – The input training data of shape [n\_samples, n\_features]. Required.
- **y\_train** (*pd.Series*) – The target training data of length [n\_samples]. Required for supervised learning tasks.
- **problem\_type** (*str* or *ProblemTypes*) – type of supervised learning problem. See `evalml.problem_types.ProblemType.all_problem_types` for a full list.
- **objective** (*str*, *ObjectiveBase*) – The objective to optimize for. Used to propose and rank pipelines, but not for optimizing each pipeline during fit-time. When set to 'auto', chooses:
  - LogLossBinary for binary classification problems,
  - LogLossMulticlass for multiclass classification problems, and
  - R2 for regression problems.
- **problem\_configuration** (*dict*) – Additional parameters needed to configure the search. For example,
- **time series problems** (*in*) –
- **should be passed in for the date\_index** (*values*) –
- **gap** –
- **max\_delay variables.** (*and*) –

Other keyword arguments which are provided will be passed to AutoMLSearch.

**Returns** the automl search object containing pipelines and rankings, and the results from running the data checks. If the data check results contain errors, automl search will not be run and an automl search object will not be returned.

**Return type** (*AutoMLSearch*, dict)

## callbacks

### Module Contents

### Functions

<code>log_error_callback</code>	Logs the exception thrown as an error. Will not throw. This is the default behavior for AutoMLSearch.
<code>raise_error_callback</code>	Raises the exception thrown by the AutoMLSearch object. Also logs the exception as an error.
<code>silent_error_callback</code>	No-op.

### Attributes Summary

<code>logger</code>
---------------------

### Contents

`evalml automl callbacks.log_error_callback(exception, traceback, automl, **kwargs)`

Logs the exception thrown as an error. Will not throw. This is the default behavior for AutoMLSearch.

`evalml automl callbacks.logger`

`evalml automl callbacks.raise_error_callback(exception, traceback, automl, **kwargs)`

Raises the exception thrown by the AutoMLSearch object. Also logs the exception as an error.

`evalml automl callbacks.silent_error_callback(exception, traceback, automl, **kwargs)`

No-op.

## pipeline\_search\_plots

### Module Contents

### Classes Summary

<code>PipelineSearchPlots</code>	Plots for the AutoMLSearch class.
<code>SearchIterationPlot</code>	

## Contents

**class** evalml.pipeline\_search\_plots.**PipelineSearchPlots** (*results*, *objective*)

Plots for the AutoMLSearch class.

### Methods

<i>search_iteration_plot</i>	Shows a plot of the best score at each iteration using data gathered during training.
------------------------------	---

**search\_iteration\_plot** (*self*, *interactive\_plot=False*)

Shows a plot of the best score at each iteration using data gathered during training.

**Returns** plot

**class** evalml.pipeline\_search\_plots.**SearchIterationPlot** (*results*, *objective*)

### Methods

<i>update</i>
---------------

**update** (*self*, *results*, *objective*)

## utils

## Module Contents

## Functions

<i>check_all_pipeline_names_unique</i>	Checks whether all the pipeline names are unique.
<i>get_best_sampler_for_data</i>	Returns the name of the sampler component to use for AutoMLSearch.
<i>get_default_primary_search_objective</i>	Get the default primary search objective for a problem type.
<i>get_pipelines_from_component_graphs</i>	Returns created pipelines from passed component graphs based on the specified problem type.
<i>make_data_splitter</i>	Given the training data and ML problem parameters, compute a data splitting method to use during AutoML search.
<i>tune_binary_threshold</i>	Tunes the threshold of a binary pipeline to the X and y thresholding data



## Attributes Summary

---

*AutoMLConfig*

---

## Contents

`evalml automl utils`. **AutoMLConfig**

`evalml automl utils`. **check\_all\_pipeline\_names\_unique** (*pipelines*)

Checks whether all the pipeline names are unique.

**Parameters** **pipelines** (*list (PipelineBase)*) – List of pipelines to check if all names are unique.

**Returns** None

**Raises** **ValueError** – if any pipeline names are duplicated.

`evalml automl utils`. **get\_best\_sampler\_for\_data** (*X*, *y*, *sampler\_method*, *sampler\_balanced\_ratio*)

Returns the name of the sampler component to use for AutoMLSearch.

**Parameters**

- **X** (*pd.DataFrame*) – The input feature data
- **y** (*pd.Series*) – The input target data
- **sampler\_method** (*str*) – The *sampler\_type* argument passed to AutoMLSearch
- **sampler\_balanced\_ratio** (*float*) – The ratio of min:majority targets that we would consider balanced, or should balance the classes to.

**Returns** The string name of the sampling component to use, or None if no sampler is necessary

**Return type** *str*, None

`evalml automl utils`. **get\_default\_primary\_search\_objective** (*problem\_type*)

Get the default primary search objective for a problem type.

**Parameters** **problem\_type** (*str or ProblemType*) – problem type of interest.

**Returns** primary objective instance for the problem type.

**Return type** ObjectiveBase

`evalml automl utils`. **get\_pipelines\_from\_component\_graphs** (*component\_graphs\_dict*, *problem\_type*, *parameters=None*, *random\_seed=0*)

Returns created pipelines from passed component graphs based on the specified problem type.

**Parameters**

- **component\_graphs\_dict** (*dict*) – The dict of component graphs.
- **problem\_type** (*str or ProblemType*) – The problem type for which pipelines will be created.
- **parameters** (*dict or None*) – Pipeline-level parameters that should be passed to the proposed pipelines.

- **random\_seed** (*int*) – Random seed.

**Returns** List of pipelines made from the passed component graphs.

**Return type** list

`evalml automl utils make_data_splitter` (*X*, *y*, *problem\_type*, *problem\_configuration=None*,  
*n\_splits=3*, *shuffle=True*, *random\_seed=0*)

Given the training data and ML problem parameters, compute a data splitting method to use during AutoML search.

**Parameters**

- **X** (*pd.DataFrame*) – The input training data of shape [n\_samples, n\_features].
- **y** (*pd.Series*) – The target training data of length [n\_samples].
- **problem\_type** (*ProblemType*) – The type of machine learning problem.
- **problem\_configuration** (*dict*, *None*) – Additional parameters needed to configure the search. For example, in time series problems, values should be passed in for the `date_index`, `gap`, and `max_delay` variables. Defaults to *None*.
- **n\_splits** (*int*, *None*) – The number of CV splits, if applicable. Defaults to 3.
- **shuffle** (*bool*) – Whether or not to shuffle the data before splitting, if applicable. Defaults to *True*.
- **random\_seed** (*int*) – Seed for the random number generator. Defaults to 0.

**Returns** Data splitting method.

**Return type** `sklearn.model_selection.BaseCrossValidator`

`evalml automl utils tune_binary_threshold` (*pipeline*, *objective*, *problem\_type*,  
*X\_threshold\_tuning*, *y\_threshold\_tuning*)

Tunes the threshold of a binary pipeline to the X and y thresholding data

**Parameters**

- **pipeline** (*Pipeline*) – Pipeline instance to threshold.
- **objective** (*ObjectiveBase*) – The objective we want to tune with. If not tuneable and `best_pipeline` is *True*, will use F1.
- **problem\_type** (*ProblemType*) – The problem type of the pipeline.
- **X\_threshold\_tuning** (*pd.DataFrame*) – Features to tune pipeline to.
- **y\_threshold\_tuning** (*pd.Series*) – Target data to tune pipeline to.

## Package Contents

### Classes Summary

<code>AutoMLSearch</code>	Automated Pipeline search.
<code>EngineBase</code>	Helper class that provides a standard way to create an ABC using
<code>SequentialEngine</code>	The default engine for the AutoML search. Trains and scores pipelines locally and sequentially.

## Functions

<code>get_default_primary_search_objective</code>	Get the default primary search objective for a problem type.
<code>make_data_splitter</code>	Given the training data and ML problem parameters, compute a data splitting method to use during AutoML search.
<code>search</code>	Given data and configuration, run an automl search.
<code>tune_binary_threshold</code>	Tunes the threshold of a binary pipeline to the X and y thresholding data

## Contents

```
class evalml.automl.AutoMLSearch(X_train=None, y_train=None, problem_type=None,
                                objective='auto', max_iterations=None,
                                max_time=None, patience=None, tolerance=None,
                                data_splitter=None, allowed_component_graphs=None,
                                allowed_model_families=None,
                                start_iteration_callback=None, add_result_callback=None,
                                error_callback=None, additional_objectives=None, al-
                                ternate_thresholding_objective='F1', random_seed=0,
                                n_jobs=-1, tuner_class=None, optimize_thresholds=True,
                                ensembling=False, max_batches=None, prob-
                                lem_configuration=None, train_best_pipeline=True,
                                pipeline_parameters=None, custom_hyperparameters=None,
                                sampler_method='auto', sampler_balanced_ratio=0.25,
                                _ensembling_split_size=0.2, _pipelines_per_batch=5,
                                engine=None)
```

Automated Pipeline search.

### Parameters

- **X\_train** (*pd.DataFrame*) – The input training data of shape [n\_samples, n\_features]. Required.
- **y\_train** (*pd.Series*) – The target training data of length [n\_samples]. Required for supervised learning tasks.
- **problem\_type** (*str or ProblemTypes*) – type of supervised learning problem. See `evalml.problem_types.ProblemType.all_problem_types` for a full list.
- **objective** (*str, ObjectiveBase*) – The objective to optimize for. Used to propose and rank pipelines, but not for optimizing each pipeline during fit-time. When set to 'auto', chooses:
  - LogLossBinary for binary classification problems,
  - LogLossMulticlass for multiclass classification problems, and
  - R2 for regression problems.
- **max\_iterations** (*int*) – Maximum number of iterations to search. If `max_iterations` and `max_time` is not set, then `max_iterations` will default to `max_iterations` of 5.
- **max\_time** (*int, str*) – Maximum time to search for pipelines. This will not start a new pipeline search after the duration has elapsed. If it is an integer, then the time will be in

seconds. For strings, time can be specified as seconds, minutes, or hours.

- **patience** (*int*) – Number of iterations without improvement to stop search early. Must be positive. If None, early stopping is disabled. Defaults to None.
- **tolerance** (*float*) – Minimum percentage difference to qualify as score improvement for early stopping. Only applicable if patience is not None. Defaults to None.
- **allowed\_component\_graphs** (*dict*) – A dictionary of lists or ComponentGraphs indicating the component graphs allowed in the search. The format should follow { “Name\_0”: [list\_of\_components], “Name\_1”: [ComponentGraph(...)] }

The default of None indicates all pipeline component graphs for this problem type are allowed. Setting this field will cause allowed\_model\_families to be ignored.

e.g. allowed\_component\_graphs = { “My\_Graph”: [“Imputer”, “One Hot Encoder”, “Random Forest Classifier”] }

- **allowed\_model\_families** (*list(str, ModelFamily)*) – The model families to search. The default of None searches over all model families. Run evalml.pipelines.components.utils.allowed\_model\_families(“binary”) to see options. Change *binary* to *multiclass* or *regression* depending on the problem type. Note that if allowed\_pipelines is provided, this parameter will be ignored.
- **data\_splitter** (*sklearn.model\_selection.BaseCrossValidator*) – Data splitting method to use. Defaults to StratifiedKFold.
- **tuner\_class** – The tuner class to use. Defaults to SKOptTuner.
- **optimize\_thresholds** (*bool*) – Whether or not to optimize the binary pipeline threshold. Defaults to True.
- **start\_iteration\_callback** (*callable*) – Function called before each pipeline training iteration. Callback function takes three positional parameters: The pipeline instance and the AutoMLSearch object.
- **add\_result\_callback** (*callable*) – Function called after each pipeline training iteration. Callback function takes three positional parameters: A dictionary containing the training results for the new pipeline, an untrained\_pipeline containing the parameters used during training, and the AutoMLSearch object.
- **error\_callback** (*callable*) – Function called when *search()* errors and raises an Exception. Callback function takes three positional parameters: the Exception raised, the traceback, and the AutoMLSearch object. Must also accepts kwargs, so AutoMLSearch is able to pass along other appropriate parameters by default. Defaults to None, which will call *log\_error\_callback*.
- **additional\_objectives** (*list*) – Custom set of objectives to score on. Will override default objectives for problem type if not empty.
- **alternate\_thresholding\_objective** (*str*) – The objective to use for thresholding binary classification pipelines if the main objective provided isn’t tuneable. Defaults to F1.
- **random\_seed** (*int*) – Seed for the random number generator. Defaults to 0.
- **n\_jobs** (*int or None*) – Non-negative integer describing level of parallelism used for pipelines. None and 1 are equivalent. If set to -1, all CPUs are used. For n\_jobs below -1, (n\_cpus + 1 + n\_jobs) are used.

- **ensembling** (*boolean*) – If True, runs ensembling in a separate batch after every allowed pipeline class has been iterated over. If the number of unique pipelines to search over per batch is one, ensembling will not run. Defaults to False.
- **max\_batches** (*int*) – The maximum number of batches of pipelines to search. Parameters `max_time`, and `max_iterations` have precedence over stopping the search.
- **problem\_configuration** (*dict*, *None*) – Additional parameters needed to configure the search. For example, in time series problems, values should be passed in for the `date_index`, `gap`, and `max_delay` variables.
- **train\_best\_pipeline** (*boolean*) – Whether or not to train the best pipeline before returning it. Defaults to True.
- **pipeline\_parameters** (*dict*) – A dict of the parameters used to initialize a pipeline with. Keys should consist of the component names and values should specify parameter values  
e.g. `pipeline_parameters = { 'Imputer' : { 'numeric_impute_strategy': 'most_frequent' } }`
- **custom\_hyperparameters** (*dict*) – A dict of the hyperparameter ranges used to iterate over during search. Keys should consist of the component names and values should specify a singular value or `skopt.Space`.  
e.g. `custom_hyperparameters = { 'Imputer' : { 'numeric_impute_strategy': Categorical(['most_frequent', 'median']) } }`
- **sampler\_method** (*str*) – The data sampling component to use in the pipelines if the problem type is classification and the target balance is smaller than the `sampler_balanced_ratio`. Either 'auto', which will use our preferred sampler for the data, 'Undersampler', 'Oversampler', or None. Defaults to 'auto'.
- **sampler\_balanced\_ratio** (*float*) – The minority:majority class ratio that we consider balanced, so a 1:4 ratio would be equal to 0.25. If the class balance is larger than this provided value, then we will not add a sampler since the data is then considered balanced. Overrides the `sampler_ratio` of the samplers. Defaults to 0.25.
- **ensembling\_split\_size** (*float*) – The amount of the training data we'll set aside for training ensemble metalearners. Only used when ensembling is True. Must be between 0 and 1, exclusive. Defaults to 0.2
- **pipelines\_per\_batch** (*int*) – The number of pipelines to train for every batch after the first one. The first batch will train a baseline pipeline + one of each pipeline family allowed in the search.
- **engine** (*EngineBase* or *None*) – The engine instance used to evaluate pipelines. If None, a `SequentialEngine` will be used.

## Methods

<code>add_to_rankings</code>	Fits and evaluates a given pipeline then adds the results to the automl rankings with the requirement that automl search has been run.
<code>best_pipeline</code>	Returns a trained instance of the best pipeline and parameters found during automl search. If <code>train_best_pipeline</code> is set to False, returns an untrained pipeline instance.
<code>describe_pipeline</code>	Describe a pipeline

continues on next page

Table 78 – continued from previous page

<code>full_rankings</code>	Returns a pandas.DataFrame with scoring results from all pipelines searched
<code>get_pipeline</code>	Given the ID of a pipeline training result, returns an untrained instance of the specified pipeline
<code>load</code>	Loads AutoML object at file path
<code>plot</code>	
<code>rankings</code>	Returns a pandas.DataFrame with scoring results from the highest-scoring set of parameters used with each pipeline.
<code>results</code>	Class that allows access to a copy of the results from <code>automl_search</code> .
<code>save</code>	Saves AutoML object at file path
<code>score_pipelines</code>	Score a list of pipelines on the given holdout data.
<code>search</code>	Find the best pipeline for the data set.
<code>train_pipelines</code>	Train a list of pipelines on the training data.

**add\_to\_rankings** (*self*, *pipeline*)

Fits and evaluates a given pipeline then adds the results to the automl rankings with the requirement that automl search has been run.

**Parameters** *pipeline* (*PipelineBase*) – pipeline to train and evaluate.

**property best\_pipeline** (*self*)

Returns a trained instance of the best pipeline and parameters found during automl search. If *train\_best\_pipeline* is set to False, returns an untrained pipeline instance.

**Returns** A trained instance of the best pipeline and parameters found during automl search. If *train\_best\_pipeline* is set to False, returns an untrained pipeline instance.

**Return type** *PipelineBase*

**describe\_pipeline** (*self*, *pipeline\_id*, *return\_dict=False*)

Describe a pipeline

**Parameters**

- **pipeline\_id** (*int*) – pipeline to describe
- **return\_dict** (*bool*) – If True, return dictionary of information about pipeline. Defaults to False.

**Returns** Description of specified pipeline. Includes information such as type of pipeline components, problem, training time, cross validation, etc.

**property full\_rankings** (*self*)

Returns a pandas.DataFrame with scoring results from all pipelines searched

**get\_pipeline** (*self*, *pipeline\_id*)

Given the ID of a pipeline training result, returns an untrained instance of the specified pipeline initialized with the parameters used to train that pipeline during automl search.

**Parameters** *pipeline\_id* (*int*) – pipeline to retrieve

**Returns** untrained pipeline instance associated with the provided ID

**Return type** *PipelineBase*

**static load** (*file\_path*, *pickle\_type='cloudpickle'*)

Loads AutoML object at file path

**Parameters**

- **file\_path** (*str*) – location to find file to load
- {"**pickle**" (*pickle\_type*)} – the pickling library to use. Currently not used since the standard pickle library can handle cloudpickles.
- "**cloudpickle**" – the pickling library to use. Currently not used since the standard pickle library can handle cloudpickles.

**Returns** AutoSearchBase object

**property plot** (*self*)

**property rankings** (*self*)

Returns a pandas.DataFrame with scoring results from the highest-scoring set of parameters used with each pipeline.

**property results** (*self*)

Class that allows access to a copy of the results from *automl\_search*.

**Returns:** dict containing **pipeline\_results**: a dict with results from each pipeline, and **search\_order**: a list describing the order the pipelines were searched.

**save** (*self*, *file\_path*, *pickle\_type*='cloudpickle', *pickle\_protocol*=cloudpickle.DEFAULT\_PROTOCOL)

Saves AutoML object at file path

**Parameters**

- **file\_path** (*str*) – location to save file
- {"**pickle**" (*pickle\_type*)} – the pickling library to use.
- "**cloudpickle**" – the pickling library to use.
- **pickle\_protocol** (*int*) – the pickle data stream format.

**Returns** None

**score\_pipelines** (*self*, *pipelines*, *X\_holdout*, *y\_holdout*, *objectives*)

Score a list of pipelines on the given holdout data.

**Parameters**

- **pipelines** (*list* (*PipelineBase*)) – List of pipelines to train.
- **X\_holdout** (*pd.DataFrame*) – Holdout features.
- **y\_holdout** (*pd.Series*) – Holdout targets for scoring.
- **objectives** (*list* (*str*), *list* (*ObjectiveBase*)) – Objectives used for scoring.

**Returns** Dictionary keyed by pipeline name that maps to a dictionary of scores. Note that the any pipelines that error out during scoring will not be included in the dictionary but the exception and stacktrace will be displayed in the log.

**Return type** Dict[str, Dict[str, float]]

**search** (*self*, *show\_iteration\_plot*=True)

Find the best pipeline for the data set.

**Parameters**

- **feature\_types** (*list*, *optional*) – list of feature types, either numerical or categorical. Categorical features will automatically be encoded

- **show\_iteration\_plot** (*boolean, True*) – Shows an iteration vs. score plot in Jupyter notebook. Disabled by default in non-Jupyter environments.

**train\_pipelines** (*self, pipelines*)

Train a list of pipelines on the training data.

This can be helpful for training pipelines once the search is complete.

**Parameters** **pipelines** (*list (PipelineBase)*) – List of pipelines to train.

**Returns** Dictionary keyed by pipeline name that maps to the fitted pipeline. Note that the any pipelines that error out during training will not be included in the dictionary but the exception and stacktrace will be displayed in the log.

**Return type** Dict[str, PipelineBase]

**class** evalml.automl.EngineBase

Helper class that provides a standard way to create an ABC using inheritance.

**Methods**

---

*setup\_job\_log*

---

<i>submit_evaluation_job</i>	Submit job for pipeline evaluation during AutoMLSearch.
<i>submit_scoring_job</i>	Submit job for pipeline scoring.
<i>submit_training_job</i>	Submit job for pipeline training.

---

**static** **setup\_job\_log**()

**abstract** **submit\_evaluation\_job** (*self, automl\_config, pipeline, X, y*)  
Submit job for pipeline evaluation during AutoMLSearch.

**abstract** **submit\_scoring\_job** (*self, automl\_config, pipeline, X, y, objectives*)  
Submit job for pipeline scoring.

**abstract** **submit\_training\_job** (*self, automl\_config, pipeline, X, y*)  
Submit job for pipeline training.

evalml.automl.**get\_default\_primary\_search\_objective** (*problem\_type*)

Get the default primary search objective for a problem type.

**Parameters** **problem\_type** (*str or ProblemType*) – problem type of interest.

**Returns** primary objective instance for the problem type.

**Return type** ObjectiveBase

evalml.automl.**make\_data\_splitter** (*X, y, problem\_type, problem\_configuration=None, n\_splits=3, shuffle=True, random\_seed=0*)

Given the training data and ML problem parameters, compute a data splitting method to use during AutoML search.

**Parameters**

- **X** (*pd.DataFrame*) – The input training data of shape [n\_samples, n\_features].
- **y** (*pd.Series*) – The target training data of length [n\_samples].
- **problem\_type** (*ProblemType*) – The type of machine learning problem.
- **problem\_configuration** (*dict, None*) – Additional parameters needed to configure the search. For example, in time series problems, values should be passed in for the



date\_index, gap, and max\_delay variables. Defaults to None.

- **n\_splits** (*int*, *None*) – The number of CV splits, if applicable. Defaults to 3.
- **shuffle** (*bool*) – Whether or not to shuffle the data before splitting, if applicable. Defaults to True.
- **random\_seed** (*int*) – Seed for the random number generator. Defaults to 0.

**Returns** Data splitting method.

**Return type** sklearn.model\_selection.BaseCrossValidator

`evalml automl .search (X_train=None, y_train=None, problem_type=None, objective='auto', problem_configuration=None, **kwargs)`

Given data and configuration, run an automl search.

This method will run EvalML's default suite of data checks. If the data checks produce errors, the data check results will be returned before running the automl search. In that case we recommend you alter your data to address these errors and try again.

This method is provided for convenience. If you'd like more control over when each of these steps is run, consider making calls directly to the various pieces like the data checks and AutoMLSearch, instead of using this method.

#### Parameters

- **X\_train** (*pd.DataFrame*) – The input training data of shape [n\_samples, n\_features]. Required.
- **y\_train** (*pd.Series*) – The target training data of length [n\_samples]. Required for supervised learning tasks.
- **problem\_type** (*str or ProblemTypes*) – type of supervised learning problem. See `evalml.problem_types.ProblemType.all_problem_types` for a full list.
- **objective** (*str, ObjectiveBase*) – The objective to optimize for. Used to propose and rank pipelines, but not for optimizing each pipeline during fit-time. When set to 'auto', chooses:
  - LogLossBinary for binary classification problems,
  - LogLossMulticlass for multiclass classification problems, and
  - R2 for regression problems.
- **problem\_configuration** (*dict*) – Additional parameters needed to configure the search. For example,
- **time series problems** (*in*) –
- **should be passed in for the date\_index** (*values*) –
- **gap** –
- **max\_delay variables.** (*and*) –

Other keyword arguments which are provided will be passed to AutoMLSearch.

**Returns** the automl search object containing pipelines and rankings, and the results from running the data checks. If the data check results contain errors, automl search will not be run and an automl search object will not be returned.

**Return type** (*AutoMLSearch*, dict)

**class** evalml automl SequentialEngine

The default engine for the AutoML search. Trains and scores pipelines locally and sequentially.

**Methods**

---

*setup\_job\_log*

---

*submit\_evaluation\_job*

Submit job for pipeline evaluation during AutoMLSearch.

---

*submit\_scoring\_job*

Submit job for pipeline scoring.

---

*submit\_training\_job*

Submit job for pipeline training.

---

**static** setup\_job\_log()**submit\_evaluation\_job**(self, automl\_config, pipeline, X, y)

Submit job for pipeline evaluation during AutoMLSearch.

**submit\_scoring\_job**(self, automl\_config, pipeline, X, y, objectives)

Submit job for pipeline scoring.

**submit\_training\_job**(self, automl\_config, pipeline, X, y)

Submit job for pipeline training.

evalml.automl.tune\_binary\_threshold(pipeline, objective, problem\_type, X\_threshold\_tuning, y\_threshold\_tuning)

Tunes the threshold of a binary pipeline to the X and y thresholding data

**Parameters**

- **pipeline** (*Pipeline*) – Pipeline instance to threshold.
- **objective** (*ObjectiveBase*) – The objective we want to tune with. If not tuneable and best\_pipeline is True, will use F1.
- **problem\_type** (*ProblemType*) – The problem type of the pipeline.
- **X\_threshold\_tuning** (*pd.DataFrame*) – Features to tune pipeline to.
- **y\_threshold\_tuning** (*pd.Series*) – Target data to tune pipeline to.

**Data Checks****Submodules****class** imbalance\_data\_check**Module Contents****Classes Summary**

---

*ClassImbalanceDataCheck*

Check if any of the target labels are imbalanced, or if the number of values for each target are below 2 times the number of CV folds. Use for classification problems.

---

## Contents

**class** evalml.data\_checks.class\_imbalance\_data\_check.**ClassImbalanceDataCheck** (*threshold=0.1, min\_samples=100, num\_cv\_folds=3*)

Check if any of the target labels are imbalanced, or if the number of values for each target are below 2 times the number of CV folds. Use for classification problems.

### Parameters

- **threshold** (*float*) – The minimum threshold allowed for class imbalance before a warning is raised. This threshold is calculated by comparing the number of samples in each class to the sum of samples in that class and the majority class. For example, a multi-class case with [900, 900, 100] samples per classes 0, 1, and 2, respectively, would have a 0.10 threshold for class 2 ( $100 / (900 + 100)$ ). Defaults to 0.10.
- **min\_samples** (*int*) – The minimum number of samples per accepted class. If the minority class is both below the threshold and min\_samples, then we consider this severely imbalanced. Must be greater than 0. Defaults to 100.
- **num\_cv\_folds** (*int*) – The number of cross-validation folds. Must be positive. Choose 0 to ignore this warning. Defaults to 3.

### Methods

<i>name</i>	Returns a name describing the data check.
<i>validate</i>	Checks if any target labels are imbalanced beyond a threshold for binary and multiclass problems

**name** (*cls*)

Returns a name describing the data check.

**validate** (*self, X, y*)

**Checks if any target labels are imbalanced beyond a threshold for binary and multiclass problems**

Ignores NaN values in target labels if they appear.

### Parameters

- **X** (*pd.DataFrame, np.ndarray*) – Features. Ignored.
- **y** (*pd.Series, np.ndarray*) – Target labels to check for imbalanced data.

### Returns

**Dictionary with DataCheckWarnings if imbalance in classes is less than the threshold,**  
and DataCheckErrors if the number of values for each target is below  $2 * \text{num\_cv\_folds}$ .

**Return type** dict

Example

```
>>> import pandas as pd
>>> X = pd.DataFrame()
>>> y = pd.Series([0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1])
>>> target_check = ClassImbalanceDataCheck(threshold=0.10)
>>> assert target_check.validate(X, y) == {"errors": [{"message": "The number
of instances of these targets is less than 2 * the number of cross folds =
6 instances: [0]",
"data_check_name": "ClassImbalanceDataCheck",
"level": "error",
"code": "CLASS_IMBALANCE_BELOW_FOLDS",
"details": {"target_values": [0]}}, {"warnings": [{"message": "The following labels
fall below 10% of the target: [0]",
"data_check_name": "ClassImbalanceDataCheck",
"level": "warning",
"code": "CLASS_IMBALANCE_BELOW_THRESHOLD",
"details": {"target_values": [0]}}, {"message":
"The following labels in the target have severe class imbalance because
they fall under 10% of the target and have less than 100 samples: [0]",
"data_check_name": "ClassImbalanceDataCheck",
"level": "warning",
"code": "CLASS_IMBALANCE_SEVERE",
"details": {"target_values": [0]}}, {"actions": []}]}
```

data\_check

Module Contents

Classes Summary

<i>DataCheck</i>	Base class for all data checks. Data checks are a set of heuristics used to determine if there are problems with input data.
------------------	--

Contents

**class** evalml.data\_checks.data\_check.**DataCheck**  
Base class for all data checks. Data checks are a set of heuristics used to determine if there are problems with input data.

Methods

<i>name</i>	Returns a name describing the data check.
-------------	---

continues on next page

Table 84 – continued from previous page

<i>validate</i>	Inspects and validates the input data, runs any necessary calculations or algorithms, and returns a list of warnings and errors if applicable.
-----------------	--

**name** (*cls*)

Returns a name describing the data check.

**abstract validate** (*self, X, y=None*)

Inspects and validates the input data, runs any necessary calculations or algorithms, and returns a list of warnings and errors if applicable.

#### Parameters

- **X** (*pd.DataFrame*) – The input data of shape [n\_samples, n\_features]
- **y** (*pd.Series, optional*) – The target data of length [n\_samples]

**Returns** Dictionary of DataCheckError and DataCheckWarning messages

**Return type** dict (DataCheckMessage)

## data\_check\_action

### Module Contents

### Classes Summary

<i>DataCheckAction</i>	A recommended action returned by a DataCheck.
------------------------	---

### Contents

**class** evalml.data\_checks.data\_check\_action.**DataCheckAction** (*action\_code, metadata=None*)

A recommended action returned by a DataCheck.

#### Parameters

- **action\_code** (*DataCheckActionCode*) – Action code associated with the action.
- **metadata** (*dict, optional*) – Additional useful information associated with the action. Defaults to None.

#### Methods

*to\_dict*

**to\_dict** (*self*)

## data\_check\_action\_code

### Module Contents

### Classes Summary

---

<i>DataCheckActionCode</i>	Enum for data check action code.
----------------------------	----------------------------------

---

### Contents

**class** evalml.data\_checks.data\_check\_action\_code.**DataCheckActionCode**  
Enum for data check action code.

#### Attributes

<b>DROP_COL</b>	Action code for dropping a column.
<b>DROP_ROWS</b>	Action code for dropping rows.
<b>IM- PUTE_COL</b>	Action code for imputing a column.
<b>TRANS- FORM_TARGET</b>	Action code for transforming the target data.

#### Methods

---

<i>name</i>	The name of the Enum member.
<i>value</i>	The value of the Enum member.

---

**name** (*self*)  
The name of the Enum member.

**value** (*self*)  
The value of the Enum member.

## data\_check\_message

### Module Contents

### Classes Summary

---

<i>DataCheckError</i>	DataCheckMessage subclass for errors returned by data checks.
<i>DataCheckMessage</i>	Base class for a message returned by a DataCheck, tagged by name.
<i>DataCheckWarning</i>	DataCheckMessage subclass for warnings returned by data checks.

---

## Contents

```
class evalml.data_checks.data_check_message.DataCheckError (message,  
data_check_name,  
message_code=None,  
details=None)
```

DataCheckMessage subclass for errors returned by data checks.

### Attributes

<b>message_type</b>	DataCheckMessageType.ERROR
---------------------	----------------------------

### Methods

---

*to\_dict*

---

**to\_dict** (*self*)

```
class evalml.data_checks.data_check_message.DataCheckMessage (message,  
data_check_name,  
message_code=None,  
details=None)
```

Base class for a message returned by a DataCheck, tagged by name.

### Parameters

- **message** (*str*) – Message string
- **data\_check\_name** (*str*) – Name of data check
- **message\_code** (*DataCheckMessageCode*) – Message code associated with message. Defaults to None.
- **details** (*dict*) – Additional useful information associated with the message. Defaults to None.

### Attributes

<b>message_type</b>	None
---------------------	------

### Methods

---

*to\_dict*

---

**to\_dict** (*self*)

```
class evalml.data_checks.data_check_message.DataCheckWarning (message,  
data_check_name,  
message_code=None,  
details=None)
```

DataCheckMessage subclass for warnings returned by data checks.

## Attributes

<b>message_type</b>	DataCheckMessageType.WARNING
---------------------	------------------------------

## Methods

---

`to_dict`


---

`to_dict(self)`
`data_check_message_code`

## Module Contents

## Classes Summary

---

<code>DataCheckMessageCode</code>	Enum for data check message code.
-----------------------------------	-----------------------------------

---

## Contents

**class** evalml.data\_checks.data\_check\_message\_code.**DataCheckMessageCode**  
Enum for data check message code.

## Attributes

<b>CLASS_IMBALANCE_BELOW_FOLDS</b>	Message code for when number of values for each target is below 2 * number of CV folds.
<b>CLASS_IMBALANCE_BELOW_THRESHOLD</b>	Message code for when number of classes is less than the threshold.
<b>CLASS_IMBALANCE_SEVERE</b>	Message code for when balance in classes is less than the threshold and minimum class is less than minimum number of accepted samples.
<b>DATE-TIME_HAS_NAN</b>	Message code for when input datetime columns contain NaN values.
<b>DATE-TIME_HAS_UNEVEN_INTERVALS</b>	Message code for when the datetime values have uneven intervals.
<b>DATE-TIME_INFORMATION_NOT_FOUND</b>	Message code for when datetime information can not be found or is in an unaccepted format.
<b>DATE-TIME_IS_NOT_MONOTONIC</b>	Message code for when the datetime values are not monotonically increasing.
<b>HAS_ID_COLUMNS</b>	Message code for data that has ID columns.
<b>HAS_OUTLIERS</b>	Message code for when outliers are detected.
<b>HIGH_VARIANCE</b>	Message code for when high variance is detected for cross-validation.
<b>HIGHLY_NULL_COLUMNS</b>	Message code for highly null columns.
<b>HIGHLY_NULL_ROWS</b>	Message code for highly null rows.
<b>IS_MULTICOLLINEAR</b>	Message code for when data is potentially multicollinear.
<b>MIS-MATCHED_INDICES</b>	Message code for when input target and features have mismatched indices.

continues on next page



Table 94 – continued from previous page

<b>MIS-MATCHED_INDICES_ORDER</b>	Message code for when input target and features have mismatched indices order. The two indices have the same index values, but shuffled.
<b>MIS-MATCHED_LENGTHS</b>	Message code for when input target and features have different lengths.
<b>NATURAL_LANGUAGE_HAS_NAN</b>	Message code for when input natural language columns contain NaN values.
<b>NO_VARIANCE</b>	Message code for when data has no variance (1 unique value).
<b>NO_VARIANCE_WITH_NULL</b>	Message code for when data has one unique value and NaN values.
<b>NOT_UNIQUE_ENOUGH</b>	Message code for when data does not possess enough unique values.
<b>TARGET_BINARY_NOT_TWO_UNIQUE_VALUES</b>	Message code for target data for a binary classification problem that does not have two unique values.
<b>TARGET_HAS_NULL</b>	Message code for target data that has null values.
<b>TARGET_INCOMPATIBLE_OBJECTIVE</b>	Message code for target data that has incompatible values for the specified objective.
<b>TARGET_IS_EMPTY_OR_FULLY_NULL</b>	Message code for target data that is empty or has all null values.
<b>TARGET_IS_NONE</b>	Message code for when target is None.
<b>TARGET_LEAKAGE</b>	Message code for when target leakage is detected.
<b>TARGET_LOGNORMAL_DISTRIBUTION</b>	Message code for target data with a lognormal distribution.
<b>TARGET_MULTICLASS_HIGH_UNIQUE_CLASS</b>	Message code for target data for a multi classification problem that has an abnormally large number of unique classes relative to the number of target values.
<b>TARGET_MULTICLASS_NOT_ENOUGH_CLASSES</b>	Message code for target data for a multi classification problem that does not have more than a minimum number of classes.
<b>TARGET_MULTICLASS_NOT_TWO_EXAMPLES_PER_CLASS</b>	Message code for target data for a multi classification problem that does not have two examples per class.
<b>TARGET_UNSUPPORTED_PROBLEM_TYPE</b>	Message code for target data that is being checked against an unsupported problem type.
<b>TARGET_UNSUPPORTED_TYPE</b>	Message code for target data that is of an unsupported type.
<b>TOO_SPARSE</b>	Message code for when multiclass data has values that are too sparsely populated.
<b>TOO_UNIQUE</b>	Message code for when data possesses too many unique values.

### Methods

<i>name</i>	The name of the Enum member.
<i>value</i>	The value of the Enum member.

**name** (*self*)

The name of the Enum member.

**value** (*self*)

The value of the Enum member.

## data\_check\_message\_type

### Module Contents

### Classes Summary

---

<i>DataCheckMessageType</i>	Enum for type of data check message: WARNING or ERROR.
-----------------------------	--

---

### Contents

**class** evalml.data\_checks.data\_check\_message\_type.**DataCheckMessageType**  
Enum for type of data check message: WARNING or ERROR.

#### Attributes

<b>ERROR</b>	Error message returned by a data check.
<b>WARNING</b>	Warning message returned by a data check.

#### Methods

---

<i>name</i>	The name of the Enum member.
<i>value</i>	The value of the Enum member.

---

**name** (*self*)  
The name of the Enum member.

**value** (*self*)  
The value of the Enum member.

## data\_checks

### Module Contents

### Classes Summary

---

<i>DataChecks</i>	A collection of data checks.
-------------------	------------------------------

---

## Contents

**class** evalml.data\_checks.data\_checks.**DataChecks** (*data\_checks=None*,  
*data\_check\_params=None*)

A collection of data checks.

### Methods

<i>validate</i>	Inspects and validates the input data against data checks and returns a list of warnings and errors if applicable.
-----------------	--

**validate** (*self, X, y=None*)

Inspects and validates the input data against data checks and returns a list of warnings and errors if applicable.

### Parameters

- **X** (*pd.DataFrame, np.ndarray*) – The input data of shape [n\_samples, n\_features]
- **y** (*pd.Series, np.ndarray*) – The target data of length [n\_samples]

**Returns** Dictionary containing DataCheckMessage objects

**Return type** dict

## datetime\_format\_data\_check

## Module Contents

## Classes Summary

<i>DateTimeFormatDataCheck</i>	Checks if the datetime column has equally spaced intervals and is monotonically increasing or decreasing in order
--------------------------------	---

## Contents

**class** evalml.data\_checks.datetime\_format\_data\_check.**DateTimeFormatDataCheck** (*datetime\_column='index'*)

Checks if the datetime column has equally spaced intervals and is monotonically increasing or decreasing in order to be supported by time series estimators.

**Parameters** **datetime\_column** (*str, int*) – The name of the datetime column. If the datetime values are in the index, then pass “index”.

### Methods

<i>name</i>	Returns a name describing the data check.
<i>validate</i>	Checks if the target data has equal intervals and is sorted.

**name** (*cls*)

Returns a name describing the data check.



## Contents

**class** evalml.data\_checks.datetime\_nan\_data\_check.DateTimeNaNDataCheck

Checks each column in the input for datetime features and will issue an error if NaN values are present.

### Methods

<i>name</i>	Returns a name describing the data check.
<i>validate</i>	Checks if any datetime columns contain NaN values.

**name** (*cls*)

Returns a name describing the data check.

**validate** (*self, X, y=None*)

Checks if any datetime columns contain NaN values.

### Parameters

- **X** (*pd.DataFrame, np.ndarray*) – Features.
- **y** (*pd.Series, np.ndarray*) – Ignored. Defaults to None.

**Returns** dict with a DataCheckError if NaN values are present in datetime columns.

**Return type** dict

## Example

```
>>> import pandas as pd
>>> import woodwork as ww
>>> import numpy as np
>>> dates = np.arange(np.datetime64('2017-01-01'), np.datetime64('2017-01-08
↳'))
>>> dates[0] = np.datetime64('NaT')
>>> df = pd.DataFrame(dates, columns=['index'])
>>> df.ww.init()
>>> dt_nan_check = DateTimeNaNDataCheck()
>>> assert dt_nan_check.validate(df) == {"warnings": [],
...                                     "actions": [],
...                                     "errors": [
↳[DataCheckError(message='Input datetime column(s) (index) contains NaN
↳values. Please impute NaN values or drop these rows or columns.',
...                                                         data_
↳check_name=DateTimeNaNDataCheck.name,
...                                                         ],
↳message_code=DataCheckMessageCode.DATETIME_HAS_NAN,
...                                                         ],
↳details={"columns": 'index'}).to_dict()]}
```

evalml.data\_checks.datetime\_nan\_data\_check.error\_contains\_nan = Input datetime column(s) (

## default\_data\_checks

### Module Contents

### Classes Summary

---

<i>DefaultDataChecks</i>	A collection of basic data checks that is used by AutoML by default.
--------------------------	--

---

### Contents

**class** evalml.data\_checks.default\_data\_checks.**DefaultDataChecks** (*problem\_type*,  
*objective*,  
*n\_splits*=3,  
*date-*  
*time\_column*=None)

A collection of basic data checks that is used by AutoML by default. Includes:

- *HighlyNullDataCheck*
- *HighlyNullRowsDataCheck*
- *IDColumnsDataCheck*
- *TargetLeakageDataCheck*
- *InvalidTargetDataCheck*
- *NoVarianceDataCheck*
- *ClassImbalanceDataCheck* (for classification problem types)
- *DateTimeNaNDataCheck*
- *NaturalLanguageNaNDataCheck*
- *TargetDistributionDataCheck* (for regression problem types)
- *DateTimeFormatDataCheck* (for time series problem types)

#### Parameters

- **problem\_type** (*str*) – The problem type that is being validated. Can be regression, binary, or multiclass.
- **objective** (*str* or *ObjectiveBase*) – Name or instance of the objective class.
- **n\_splits** (*int*) – The number of splits as determined by the data splitter being used. Defaults to 3.
- **datetime\_column** (*str*) – The name of the column containing datetime information to be used for time series problems.
- **to "index" indicating that the datetime information is in the index of X or y. (Default)** –

#### Methods

<i>validate</i>	Inspects and validates the input data against data checks and returns a list of warnings and errors if applicable.
-----------------	--

**validate** (*self*, *X*, *y=None*)

Inspects and validates the input data against data checks and returns a list of warnings and errors if applicable.

#### Parameters

- **X** (*pd.DataFrame*, *np.ndarray*) – The input data of shape [n\_samples, n\_features]
- **y** (*pd.Series*, *np.ndarray*) – The target data of length [n\_samples]

**Returns** Dictionary containing DataCheckMessage objects

**Return type** dict

## highly\_null\_data\_check

### Module Contents

### Classes Summary

<i>HighlyNullDataCheck</i>	Checks if there are any highly-null columns and rows in the input.
----------------------------	--

### Contents

**class** evalml.data\_checks.highly\_null\_data\_check.**HighlyNullDataCheck** (*pct\_null\_col\_threshold=0.95*, *pct\_null\_row\_threshold=0.95*)

Checks if there are any highly-null columns and rows in the input.

#### Parameters

- **pct\_null\_col\_threshold** (*float*) – If the percentage of NaN values in an input feature exceeds this amount, that column will be considered highly-null. Defaults to 0.95.
- **pct\_null\_row\_threshold** (*float*) – If the percentage of NaN values in an input row exceeds this amount, that row will be considered highly-null. Defaults to 0.95.

#### Methods

<i>name</i>	Returns a name describing the data check.
<i>validate</i>	Checks if there are any highly-null columns or rows in the input.

**name** (*cls*)

Returns a name describing the data check.

**validate** (*self*, *X*, *y=None*)

Checks if there are any highly-null columns or rows in the input.

#### Parameters

- **x** (`pd.DataFrame`, `np.ndarray`) – Features.
- **y** (`pd.Series`, `np.ndarray`) – Ignored.

**Returns** dict with a `DataCheckWarning` if there are any highly-null columns or rows.

**Return type** dict

### Example

```
>>> import pandas as pd
>>> class SeriesWrap():
...     def __init__(self, series):
...         self.series = series
...
...     def __eq__(self, series_2):
...         return all(self.series.eq(series_2.series))
...
>>> df = pd.DataFrame({
...     'lots_of_null': [None, None, None, None, 5],
...     'no_null': [1, 2, 3, 4, 5]
... })
>>> null_check = HighlyNullDataCheck(pct_null_col_threshold=0.50, pct_null_
↳ row_threshold=0.50)
>>> validation_results = null_check.validate(df)
>>> validation_results['warnings'][0]['details']['pct_null_cols'] =
↳ SeriesWrap(validation_results['warnings'][0]['details']['pct_null_cols'])
>>> highly_null_rows = SeriesWrap(pd.Series([0.5, 0.5, 0.5, 0.5]))
>>> assert validation_results == {"errors": [],
↳
↳     "warnings": [{"message": "4 out of 5 rows are more than 50.0%
↳ null",
↳     "data_
↳ check_name": "HighlyNullDataCheck",
↳     "level": "warning",
↳     "code": "HIGHLY_NULL_ROWS",
↳     "details": {"pct_null_cols": highly_null_
↳ rows}},
↳     {"message": "Column 'lots_of_null' is 50.0% or more null",
↳     "data_check_name": "HighlyNullDataCheck",
↳     "level": "warning",
↳     "code":
↳ "HIGHLY_NULL_COLS",
↳     "details": {"column": "lots_of_null", "pct_null_rows": 0.8}}],
↳     "actions": [{"code": "DROP_ROWS", "metadata":
↳ {"rows": [0, 1, 2, 3]}},
↳     {"code": "DROP_COL",
↳     "metadata": {"column": "lots_of_null"}}}]}
```



id\_columns\_data\_check

Module Contents

Classes Summary

<i>IDColumnsDataCheck</i>	Check if any of the features are likely to be ID columns.
---------------------------	---

Contents

**class** evalml.data\_checks.id\_columns\_data\_check.IDColumnsDataCheck (*id\_threshold=1.0*)  
Check if any of the features are likely to be ID columns.

**Parameters** *id\_threshold* (*float*) – The probability threshold to be considered an ID column.  
Defaults to 1.0.

Methods

<i>name</i>	Returns a name describing the data check.
<i>validate</i>	Check if any of the features are likely to be ID columns. Currently performs these simple checks:

**name** (*cls*)  
Returns a name describing the data check.

**validate** (*self, X, y=None*)  
Check if any of the features are likely to be ID columns. Currently performs these simple checks:

- column name is “id”
- column name ends in “\_id”
- column contains all unique values (and is categorical / integer type)

**Parameters** *X* (*pd.DataFrame, np.ndarray*) – The input features to check

**Returns** A dictionary of features with column name or index and their probability of being ID columns

**Return type** dict

Example

```
>>> import pandas as pd
>>> df = pd.DataFrame({
...     'df_id': [0, 1, 2, 3, 4],
...     'x': [10, 42, 31, 51, 61],
...     'y': [42, 54, 12, 64, 12]
... })
>>> id_col_check = IDColumnsDataCheck()
>>> assert id_col_check.validate(df) == {"errors": [],
    "warnings": [{"message": "Column 'df_id' is
100.0% or more likely to be an ID column",
    "data_check_name": "IDColumnsDataCheck",
    "warning",
    "code": "HAS_ID_COLUMN",
    "details": {"column": "df_id"}},
    "actions": [{"code": "DROP_COL",
    "metadata": {"column": "df_id"}
    ]}}}
```

(continued from previous page)

## invalid\_targets\_data\_check

### Module Contents

### Classes Summary

<i>InvalidTargetDataCheck</i>	Checks if the target data contains missing or invalid values.
-------------------------------	---

### Contents

**class** evalml.data\_checks.invalid\_targets\_data\_check.**InvalidTargetDataCheck** (*problem\_type*, *objective*, *n\_unique=100*)

Checks if the target data contains missing or invalid values.

#### Parameters

- **problem\_type** (*str* or *ProblemTypes*) – The specific problem type to data check for. e.g. ‘binary’, ‘multiclass’, ‘regression’, ‘time series regression’
- **objective** (*str* or *ObjectiveBase*) – Name or instance of the objective class.
- **n\_unique** (*int*) – Number of unique target values to store when problem type is binary and target incorrectly has more than 2 unique values. Non-negative integer. If None, stores all unique values. Defaults to 100.

#### Attributes

<b>multi-class_continuous_threshold</b>	0.05
---	------

#### Methods

<i>name</i>	Returns a name describing the data check.
<i>validate</i>	Checks if the target data contains missing or invalid values.

#### **name** (*cls*)

Returns a name describing the data check.

#### **validate** (*self*, *X*, *y*)

Checks if the target data contains missing or invalid values.

#### Parameters

- **X** (*pd.DataFrame*, *np.ndarray*) – Features. Ignored.

- **y** (*pd.Series*, *np.ndarray*) – Target data to check for invalid values.

**Returns** List with *DataCheckErrors* if any invalid values are found in the target data.

**Return type** dict (*DataCheckError*)

### Example

```
>>> import pandas as pd
>>> X = pd.DataFrame({"col": [1, 2, 3, 1]})
>>> y = pd.Series([0, 1, None, None])
>>> target_check = InvalidTargetDataCheck('binary', 'Log Loss Binary')
>>> assert target_check.validate(X, y) == {"errors": [{"message": "2 row(s) (50.0%) of target values are null",
    "data_check_name": "InvalidTargetDataCheck",
    "level": "error",
    "code": "TARGET_HAS_NULL",
    "details": {"num_null_rows": 2, "pct_null_rows": 50}}],
    "warnings": [],
    "actions": [{"code": 'IMPUTE_COL',
    "metadata": {'column': None, 'impute_strategy': 'most_frequent', 'is_target': True}}]}
```

## multicollinearity\_data\_check

### Module Contents

### Classes Summary

<i>MulticollinearityDataCheck</i>	Check if any set features are likely to be multicollinear.
-----------------------------------	--

### Contents

**class** evalml.data\_checks.multicollinearity\_data\_check.**MulticollinearityDataCheck** (*threshold=0.9*)

Check if any set features are likely to be multicollinear.

**Parameters** **threshold** (*float*) – The threshold to be considered. Defaults to 0.9.

#### Methods

<i>name</i>	Returns a name describing the data check.
<i>validate</i>	Check if any set of features are likely to be multicollinear.

**name** (*cls*)

Returns a name describing the data check.

**validate** (*self*, *X*, *y=None*)

Check if any set of features are likely to be multicollinear.

**Parameters** **X** (*pd.DataFrame*, *np.ndarray*) – The input features to check

**Returns** dict with a DataCheckWarning if there are any potentially multicollinear columns.

**Return type** dict

## natural\_language\_nan\_data\_check

### Module Contents

### Classes Summary

---

<i>NaturalLanguageNaNDataCheck</i>	Checks each column in the input for natural language features and will issue an error if NaN values are present.
------------------------------------	--

---

### Attributes Summary

---

<i>error_contains_nan</i>
---------------------------

---

### Contents

```
evalml.data_checks.natural_language_nan_data_check.error_contains_nan = Input natural language
```

**class** evalml.data\_checks.natural\_language\_nan\_data\_check.**NaturalLanguageNaNDataCheck**  
Checks each column in the input for natural language features and will issue an error if NaN values are present.

#### Methods

---

<i>name</i>	Returns a name describing the data check.
<i>validate</i>	Checks if any natural language columns contain NaN values.

---

**name** (*cls*)

Returns a name describing the data check.

**validate** (*self*, *X*, *y=None*)

Checks if any natural language columns contain NaN values.

#### Parameters

- **X** (*pd.DataFrame*, *np.ndarray*) – Features.
- **y** (*pd.Series*, *np.ndarray*) – Ignored. Defaults to None.

**Returns** dict with a DataCheckError if NaN values are present in natural language columns.

**Return type** dict

## Example

```

>>> import pandas as pd
>>> import woodwork as ww
>>> import numpy as np
>>> data = pd.DataFrame()
>>> data['A'] = [None, "string_that_is_long_enough_for_natural_language"]
>>> data['B'] = ['string_that_is_long_enough_for_natural_language', 'string_
↳that_is_long_enough_for_natural_language']
>>> data['C'] = np.random.randint(0, 3, size=len(data))
>>> data.ww.init(logical_types={'A': 'NaturalLanguage', 'B': 'NaturalLanguage
↳'})
>>> nl_nan_check = NaturalLanguageNaNDataCheck()
>>> assert nl_nan_check.validate(data) == {
...     "warnings": [],
...     "actions": [],
...     "errors": [DataCheckError(message='Input natural language_
↳column(s) (A) contains NaN values. Please impute NaN values or drop these_
↳rows or columns.',
...                                     data_check_name=NaturalLanguageNaNDataCheck.name,
...                                     message_code=DataCheckMessageCode.NATURAL_LANGUAGE_
↳HAS_NAN,
...                                     details={"columns": 'A'}).to_dict()]
...     }

```

## no\_variance\_data\_check

### Module Contents

### Classes Summary

<i>NoVarianceDataCheck</i>	Check if the target or any of the features have no variance.
----------------------------	--

### Attributes Summary

<i>logger</i>
---------------

### Contents

evalml.data\_checks.no\_variance\_data\_check.logger

**class** evalml.data\_checks.no\_variance\_data\_check.NoVarianceDataCheck (*count\_nan\_as\_value=False*)  
 Check if the target or any of the features have no variance.

**Parameters** *count\_nan\_as\_value* (*bool*) – If True, missing values will be counted as their own unique value. Additionally, if true, will return a DataCheckWarning instead of an error if the feature has mostly missing data and only one unique value. Defaults to False.

### Methods

<i>name</i>	Returns a name describing the data check.
<i>validate</i>	Check if the target or any of the features have no variance (1 unique value).

**name** (*cls*)

Returns a name describing the data check.

**validate** (*self*, *X*, *y*)

Check if the target or any of the features have no variance (1 unique value).

**Parameters**

- **X** (*pd.DataFrame*, *np.ndarray*) – The input features.
- **y** (*pd.Series*, *np.ndarray*) – The target data.

**Returns** dict of warnings/errors corresponding to features or target with no variance.

**Return type** dict

## outliers\_data\_check

### Module Contents

### Classes Summary

<i>OutliersDataCheck</i>	Checks if there are any outliers in input data by using IQR to determine score anomalies. Columns with score anomalies are considered to contain outliers.
--------------------------	--

### Contents

**class** evalml.data\_checks.outliers\_data\_check.**OutliersDataCheck**

Checks if there are any outliers in input data by using IQR to determine score anomalies. Columns with score anomalies are considered to contain outliers.

**Methods**

<i>name</i>	Returns a name describing the data check.
<i>validate</i>	Checks if there are any outliers in a dataframe by using IQR to determine column anomalies. Column with anomalies are considered to contain outliers.

**name** (*cls*)

Returns a name describing the data check.

**validate** (*self*, *X*, *y=None*)

Checks if there are any outliers in a dataframe by using IQR to determine column anomalies. Column with anomalies are considered to contain outliers.

**Parameters**

- **X** (*pd.DataFrame*, *np.ndarray*) – Features

- `y` (`pd.Series`, `np.ndarray`) – Ignored.

**Returns** A dictionary with warnings if any columns have outliers.

**Return type** dict

### Example

```
>>> import pandas as pd
>>> df = pd.DataFrame({
...     'x': [1, 2, 3, 4, 5],
...     'y': [6, 7, 8, 9, 10],
...     'z': [-1, -2, -3, -1201, -4]
... })
>>> outliers_check = OutliersDataCheck()
>>> assert outliers_check.validate(df) == {"warnings": [{"message":
↳ "Column(s) 'z' are likely to have outlier data.",
↳                                     "data_check_name":
↳ "OutliersDataCheck",
↳                                     "level": "warning",
↳                                     "code": "HAS_OUTLIERS",
↳                                     "details": {"columns": ["z"]}]},
↳                                     "errors": [],
↳                                     "actions": []}]}
```

## sparsity\_data\_check

### Module Contents

### Classes Summary

<code>SparsityDataCheck</code>	Checks if there are any columns with sparsely populated values in the input.
--------------------------------	--

### Attributes Summary

<code>warning_too_unique</code>
---------------------------------

### Contents

**class** evalml.data\_checks.sparsity\_data\_check.**SparsityDataCheck** (*problem\_type*, *threshold*, *unique\_count\_threshold=10*)

Checks if there are any columns with sparsely populated values in the input.

#### Parameters

- **problem\_type** (*str* or *ProblemTypes*) – The specific problem type to data check for. ‘multiclass’ or ‘time series multiclass’ is the only accepted problem type.

- **threshold** (*float*) – The threshold value, or percentage of each column’s unique values, below which, a column exhibits sparsity. Should be between 0 and 1.
- **unique\_count\_threshold** (*int*) – The minimum number of times a unique value has to be present in a column to not be considered “sparse.” Defaults to 10.

### Methods

<i>name</i>	Returns a name describing the data check.
<i>sparsity_score</i>	This function calculates a sparsity score for the given value counts by calculating the percentage of
<i>validate</i>	Calculates what percentage of each column’s unique values exceed the count threshold and compare

#### **name** (*cls*)

Returns a name describing the data check.

#### **static sparsity\_score** (*col*, *count\_threshold=10*)

This function calculates a sparsity score for the given value counts by calculating the percentage of unique values that exceed the count\_threshold.

#### Parameters

- **col** (*pd.Series*) – Feature values.
- **count\_threshold** (*int*) – The number of instances below which a value is considered sparse. Default is 10.

**Returns** Sparsity score, or the percentage of the unique values that exceed count\_threshold.

**Return type** (float)

#### **validate** (*self*, *X*, *y=None*)

Calculates what percentage of each column’s unique values exceed the count threshold and compare that percentage to the sparsity threshold stored in the class instance.

#### Parameters

- **X** (*pd.DataFrame*, *np.ndarray*) – Features.
- **y** (*pd.Series*, *np.ndarray*) – Ignored.

**Returns** dict with a DataCheckWarning if there are any sparse columns.

**Return type** dict

### Example

```
>>> import pandas as pd
>>> df = pd.DataFrame({
...     'sparse': [float(x) for x in range(100)],
...     'not_sparse': [float(1) for x in range(100)]
... })
>>> sparsity_check = SparsityDataCheck(problem_type="multiclass", threshold=0.
↳5, unique_count_threshold=10)
>>> assert sparsity_check.validate(df) == {"errors": [],
↳                                     "warnings": [{"message": "Input columns
↳(sparse) for multiclass problem type are too sparse.",
↳                                     "data_check_name": "SparsityDataCheck
↳                                     "level":
↳"warning",
↳": "TOO_SPARSE",
↳"details": {"column": "sparse", "sparsity_score": 0.0}}],
↳                                     "actions": [{"code": "SPARSE_COL",
↳                                     "metadata": {
↳"column": "sparse"}}]}
```

(continues on next page)



(continued from previous page)

```
evalml.data_checks.sparsity_data_check.warning_too_unique = Input columns ({} ) for {} prob
```

target\_distribution\_data\_check

Module Contents

Classes Summary

<i>TargetDistributionDataCheck</i>	Checks if the target data contains certain distributions that may need to be transformed prior training to
------------------------------------	--

Contents

**class** evalml.data\_checks.target\_distribution\_data\_check.TargetDistributionDataCheck  
Checks if the target data contains certain distributions that may need to be transformed prior training to improve model performance.

Methods

<i>name</i>	Returns a name describing the data check.
<i>validate</i>	Checks if the target data has a certain distribution.

**name** (*cls*)  
Returns a name describing the data check.

**validate** (*self*, *X*, *y*)  
Checks if the target data has a certain distribution.

Parameters

- **X** (*pd.DataFrame*, *np.ndarray*) – Features. Ignored.
- **y** (*pd.Series*, *np.ndarray*) – Target data to check for underlying distributions.

**Returns** List with DataCheckErrors if certain distributions are found in the target data.

**Return type** dict (DataCheckError)

Example

```
>>> from scipy.stats import lognorm
>>> X = None
>>> y = [0.946, 0.972, 1.154, 0.954, 0.969, 1.222, 1.038, 0.999, 0.973, 0.897]
>>> target_check = TargetDistributionDataCheck()
>>> assert target_check.validate(X, y) == {"errors": [],
↳                                     "warnings": [{"message": "Target may have",
↳ a lognormal distribution.",
↳                                     "data_check_name": "TargetDistributionDataCheck",
↳                                     "level":
↳ "warning",
↳ "code": "TARGET_LOGNORMAL_DISTRIBUTION",
↳                                     "details": {"shapiro-statistic/pvalue":
↳ "0.84/0.045"}]},
↳ {"actions": [{"code": 'TRANSFORM_TARGET', 'metadata': {'column': None,
↳ 'transformation_strategy': 'lognormal', 'is_target': True}}]}
```

(continues on next page)

## target\_leakage\_data\_check

### Module Contents

### Classes Summary

<i>TargetLeakageDataCheck</i>	Check if any of the features are highly correlated with the target by using mutual information or Pearson correlation.
-------------------------------	--

### Contents

**class** evalml.data\_checks.target\_leakage\_data\_check.**TargetLeakageDataCheck** (*pct\_corr\_threshold=0.95*, *method='mutual'*)

Check if any of the features are highly correlated with the target by using mutual information or Pearson correlation.

If *method='mutual'*, this data check uses mutual information and supports all target and feature types. Otherwise, if *method='pearson'*, it uses Pearson correlation and only supports binary with numeric and boolean dtypes. Pearson correlation returns a value in [-1, 1], while mutual information returns a value in [0, 1].

#### Parameters

- **pct\_corr\_threshold** (*float*) – The correlation threshold to be considered leakage. Defaults to 0.95.
- **method** (*string*) – The method to determine correlation. Use ‘mutual’ for mutual information, otherwise ‘pearson’ for Pearson correlation. Defaults to ‘mutual’.

#### Methods

<i>name</i>	Returns a name describing the data check.
<i>validate</i>	Check if any of the features are highly correlated with the target by using mutual information or Pearson correlation.

**name** (*cls*)

Returns a name describing the data check.

**validate** (*self, X, y*)

Check if any of the features are highly correlated with the target by using mutual information or Pearson correlation.

If *method='mutual'*, supports all target and feature types. Otherwise, if *method='pearson'* only supports binary with numeric and boolean dtypes. Pearson correlation returns a value in [-1, 1], while mutual information returns a value in [0, 1].

#### Parameters

- **X** (*pd.DataFrame, np.ndarray*) – The input features to check
- **y** (*pd.Series, np.ndarray*) – The target data

**Returns** dict with a DataCheckWarning if target leakage is detected.

**Return type** dict (DataCheckWarning)

### Example

```
>>> import pandas as pd
>>> X = pd.DataFrame({
...     'leak': [10, 42, 31, 51, 61],
...     'x': [42, 54, 12, 64, 12],
...     'y': [13, 5, 13, 74, 24],
... })
>>> y = pd.Series([10, 42, 31, 51, 40])
>>> target_leakage_check = TargetLeakageDataCheck(pct_corr_threshold=0.95)
>>> assert target_leakage_check.validate(X, y) == {"warnings": [{"message":
↪ "Column 'leak' is 95.0% or more correlated with the target",
↪                                     "data_check_
↪ name": "TargetLeakageDataCheck",
↪                                     "level": "warning",
↪                                     "code": "TARGET_LEAKAGE
↪ ",
↪     "details": {"column": "leak"}},
↪     "errors": [],
↪     "actions": [{"code": "DROP_COL",
↪                                     "metadata": {"column":
↪     "leak"}}]}}
```

## uniqueness\_data\_check

### Module Contents

### Classes Summary

<i>UniquenessDataCheck</i>	Checks if there are any columns in the input that are either too unique for classification problems
----------------------------	---

### Attributes Summary

<i>warning_not_unique_enough</i>
<i>warning_too_unique</i>

## Contents

**class** evalml.data\_checks.uniqueness\_data\_check.**UniquenessDataCheck** (*problem\_type*,  
*thresh-*  
*old=0.5*)

Checks if there are any columns in the input that are either too unique for classification problems or not unique enough for regression problems.

### Parameters

- **problem\_type** (*str* or *ProblemTypes*) – The specific problem type to data check for. e.g. ‘binary’, ‘multiclass’, ‘regression’, ‘time series regression’
- **threshold** (*float*) – The threshold to set as an upper bound on uniqueness for classification type problems or lower bound on for regression type problems. Defaults to 0.50.

### Methods

<i>name</i>	Returns a name describing the data check.
<i>uniqueness_score</i>	This function calculates a uniqueness score for the provided field. NaN values are
<i>validate</i>	Checks if there are any columns in the input that are too unique in the case of classification

**name** (*cls*)

Returns a name describing the data check.

**static uniqueness\_score** (*col*)

This function calculates a uniqueness score for the provided field. NaN values are not considered as unique values in the calculation.

Based on the Herfindahl–Hirschman Index.

**Parameters** *col* (*pd.Series*) – Feature values.

**Returns** Uniqueness score.

**Return type** (float)

**validate** (*self*, *X*, *y=None*)

Checks if there are any columns in the input that are too unique in the case of classification problems or not unique enough in the case of regression problems.

### Parameters

- **X** (*pd.DataFrame*, *np.ndarray*) – Features.
- **y** (*pd.Series*, *np.ndarray*) – Ignored. Defaults to None.

### Returns

**dict with a DataCheckWarning if there are any too unique or not unique enough columns.**

**Return type** dict

## Example

```
>>> import pandas as pd
>>> df = pd.DataFrame({
...     'regression_unique_enough': [float(x) for x in range(100)],
...     'regression_not_unique_enough': [float(1) for x in range(100)]
... })
>>> uniqueness_check = UniquenessDataCheck(problem_type="regression",
↳ threshold=0.8)
>>> assert uniqueness_check.validate(df) == {"errors": [],
↳                                     "warnings": [{"message": "Input_
↳ columns (regression_not_unique_enough) for regression problem type are not_
↳ unique enough.",
↳                                     "data_check_name": "UniquenessDataCheck",
↳                                     "level": "warning",
↳                                     "code": "NOT_UNIQUE_ENOUGH",
↳                                     "details": {
↳ "column": "regression_not_unique_enough", 'uniqueness_score': 0.0}},
↳                                     "actions": [{"code":
↳ "DROP_COL",
↳                                     "metadata": {"column": "regression_not_unique_enough"}}]}
```

`evalml.data_checks.uniqueness_data_check.warning_not_unique_enough = Input columns ({} for`

`evalml.data_checks.uniqueness_data_check.warning_too_unique = Input columns ({} for {} pro`

## utils

### Module Contents

### Classes Summary

<i>EmptyDataChecks</i>	A collection of data checks.
------------------------	------------------------------

## Contents

**class** `evalml.data_checks.utils.EmptyDataChecks` (*data\_checks=None*)

A collection of data checks.

### Methods

<i>validate</i>	Inspects and validates the input data against data checks and returns a list of warnings and errors if applicable.
-----------------	--

**validate** (*self, X, y=None*)

Inspects and validates the input data against data checks and returns a list of warnings and errors if applicable.

### Parameters

- **X** (`pd.DataFrame`, `np.ndarray`) – The input data of shape `[n_samples, n_features]`

- **y** (*pd.Series*, *np.ndarray*) – The target data of length [n\_samples]

**Returns** Dictionary containing DataCheckMessage objects

**Return type** dict

## Package Contents

## Classes Summary

<i>ClassImbalanceDataCheck</i>	Check if any of the target labels are imbalanced, or if the number of values for each target are below 2 times the number of CV folds. Use for classification problems.
<i>DataCheck</i>	Base class for all data checks. Data checks are a set of heuristics used to determine if there are problems with input data.
<i>DataCheckAction</i>	A recommended action returned by a DataCheck.
<i>DataCheckActionCode</i>	Enum for data check action code.
<i>DataCheckError</i>	DataCheckMessage subclass for errors returned by data checks.
<i>DataCheckMessage</i>	Base class for a message returned by a DataCheck, tagged by name.
<i>DataCheckMessageCode</i>	Enum for data check message code.
<i>DataCheckMessageType</i>	Enum for type of data check message: WARNING or ERROR.
<i>DataChecks</i>	A collection of data checks.
<i>DataCheckWarning</i>	DataCheckMessage subclass for warnings returned by data checks.
<i>DateTimeFormatDataCheck</i>	Checks if the datetime column has equally spaced intervals and is monotonically increasing or decreasing in order
<i>DateTimeNaNDataCheck</i>	Checks each column in the input for datetime features and will issue an error if NaN values are present.
<i>DefaultDataChecks</i>	A collection of basic data checks that is used by AutoML by default.
<i>EmptyDataChecks</i>	A collection of data checks.
<i>HighlyNullDataCheck</i>	Checks if there are any highly-null columns and rows in the input.
<i>IDColumnsDataCheck</i>	Check if any of the features are likely to be ID columns.
<i>InvalidTargetDataCheck</i>	Checks if the target data contains missing or invalid values.
<i>MulticollinearityDataCheck</i>	Check if any set features are likely to be multicollinear.
<i>NaturalLanguageNaNDataCheck</i>	Checks each column in the input for natural language features and will issue an error if NaN values are present.
<i>NoVarianceDataCheck</i>	Check if the target or any of the features have no variance.
<i>OutliersDataCheck</i>	Checks if there are any outliers in input data by using IQR to determine score anomalies. Columns with score anomalies are considered to contain outliers.

continues on next page

Table 135 – continued from previous page

<i>SparsityDataCheck</i>	Checks if there are any columns with sparsely populated values in the input.
<i>TargetDistributionDataCheck</i>	Checks if the target data contains certain distributions that may need to be transformed prior training to
<i>TargetLeakageDataCheck</i>	Check if any of the features are highly correlated with the target by using mutual information or Pearson correlation.
<i>UniquenessDataCheck</i>	Checks if there are any columns in the input that are either too unique for classification problems

## Contents

**class** evalml.data\_checks.**ClassImbalanceDataCheck** (*threshold=0.1*, *min\_samples=100*, *num\_cv\_folds=3*)

Check if any of the target labels are imbalanced, or if the number of values for each target are below 2 times the number of CV folds. Use for classification problems.

### Parameters

- **threshold** (*float*) – The minimum threshold allowed for class imbalance before a warning is raised. This threshold is calculated by comparing the number of samples in each class to the sum of samples in that class and the majority class. For example, a multi-class case with [900, 900, 100] samples per classes 0, 1, and 2, respectively, would have a 0.10 threshold for class 2 ( $100 / (900 + 100)$ ). Defaults to 0.10.
- **min\_samples** (*int*) – The minimum number of samples per accepted class. If the minority class is both below the threshold and min\_samples, then we consider this severely imbalanced. Must be greater than 0. Defaults to 100.
- **num\_cv\_folds** (*int*) – The number of cross-validation folds. Must be positive. Choose 0 to ignore this warning. Defaults to 3.

### Methods

<i>name</i>	Returns a name describing the data check.
<i>validate</i>	Checks if any target labels are imbalanced beyond a threshold for binary and multiclass problems

**name** (*cls*)

Returns a name describing the data check.

**validate** (*self*, *X*, *y*)

**Checks if any target labels are imbalanced beyond a threshold for binary and multiclass problems**

Ignores NaN values in target labels if they appear.

### Parameters

- **X** (*pd.DataFrame*, *np.ndarray*) – Features. Ignored.
- **y** (*pd.Series*, *np.ndarray*) – Target labels to check for imbalanced data.

### Returns

**Dictionary with DataCheckWarnings if imbalance in classes is less than the threshold,**  
and DataCheckErrors if the number of values for each target is below  $2 * \text{num\_cv\_folds}$ .

Return type dict

### Example

```
>>> import pandas as pd
>>> X = pd.DataFrame()
>>> y = pd.Series([0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1])
>>> target_check = ClassImbalanceDataCheck(threshold=0.10)
>>> assert target_check.validate(X, y) == {"errors": [{"message": "The number
↳ of instances of these targets is less than 2 * the number of cross folds =
↳ 6 instances: [0]",
↳ "data_check_name": "ClassImbalanceDataCheck",
↳ "level": "error",
↳ "code": "CLASS_
↳ IMBALANCE_BELOW_FOLDS",
↳ "details": {"target_values": [0]}},
↳ "warnings": [{"message": "The following labels
↳ fall below 10% of the target: [0]",
↳ "data_check_name": "ClassImbalanceDataCheck",
↳ "level":
↳ "warning",
↳ "code": "CLASS_IMBALANCE_BELOW_THRESHOLD",
↳ "details": {"target_values": [0]}},
↳ {"message":
↳ "The following labels in the target have severe class imbalance because
↳ they fall under 10% of the target and have less than 100 samples: [0]",
↳ "data_check_
↳ name": "ClassImbalanceDataCheck",
↳ "level": "warning",
↳ "code": "CLASS_IMBALANCE_SEVERE",
↳ "details": {
↳ "target_values": [0]}},
↳ "actions": []}
```

**class** evalml.data\_checks.DataCheck

Base class for all data checks. Data checks are a set of heuristics used to determine if there are problems with input data.

#### Methods

<code>name</code>	Returns a name describing the data check.
<code>validate</code>	Inspects and validates the input data, runs any necessary calculations or algorithms, and returns a list of warnings and errors if applicable.

**name** (*cls*)

Returns a name describing the data check.

**abstract validate** (*self*, *X*, *y=None*)

Inspects and validates the input data, runs any necessary calculations or algorithms, and returns a list of warnings and errors if applicable.

#### Parameters

- **X** (*pd.DataFrame*) – The input data of shape [n\_samples, n\_features]
- **y** (*pd.Series, optional*) – The target data of length [n\_samples]



**Returns** Dictionary of DataCheckError and DataCheckWarning messages

**Return type** dict (*DataCheckMessage*)

**class** evalml.data\_checks.DataCheckAction (*action\_code, metadata=None*)

A recommended action returned by a DataCheck.

#### Parameters

- **action\_code** (*DataCheckActionCode*) – Action code associated with the action.
- **metadata** (*dict, optional*) – Additional useful information associated with the action. Defaults to None.

#### Methods

---

*to\_dict*

---

**to\_dict** (*self*)

**class** evalml.data\_checks.DataCheckActionCode

Enum for data check action code.

#### Attributes

<b>DROP_COL</b>	Action code for dropping a column.
<b>DROP_ROWS</b>	Action code for dropping rows.
<b>IM- PUTE_COL</b>	Action code for imputing a column.
<b>TRANS- FORM_TARGET</b>	Action code for transforming the target data.

#### Methods

---

<i>name</i>	The name of the Enum member.
<i>value</i>	The value of the Enum member.

---

**name** (*self*)

The name of the Enum member.

**value** (*self*)

The value of the Enum member.

**class** evalml.data\_checks.DataCheckError (*message, data\_check\_name, message\_code=None, details=None*)

DataCheckMessage subclass for errors returned by data checks.

#### Attributes

<b>mes- sage_type</b>	DataCheckMessageType.ERROR
---------------------------	----------------------------

#### Methods

---

`to_dict`


---

`to_dict (self)`

**class** evalml.data\_checks.**DataCheckMessage** (*message*, *data\_check\_name*, *message\_code=None*, *details=None*)

Base class for a message returned by a DataCheck, tagged by name.

#### Parameters

- **message** (*str*) – Message string
- **data\_check\_name** (*str*) – Name of data check
- **message\_code** (`DataCheckMessageCode`) – Message code associated with message. Defaults to None.
- **details** (*dict*) – Additional useful information associated with the message. Defaults to None.

#### Attributes

<b>message_type</b>	None
---------------------	------

#### Methods

---

`to_dict`


---

`to_dict (self)`

**class** evalml.data\_checks.**DataCheckMessageCode**

Enum for data check message code.

#### Attributes

<b>CLASS_IMBALANCE_BELOW_FOLDS</b>	Message code for when number of values for each target is below 2 * number of CV folds.
<b>CLASS_IMBALANCE_BELOW_THRESHOLD</b>	Message code for when number of classes is less than the threshold.
<b>CLASS_IMBALANCE_SEVERE</b>	Message code for when balance in classes is less than the threshold and minimum class is less than minimum number of accepted samples.
<b>DATE-TIME_HAS_NAN</b>	Message code for when input datetime columns contain NaN values.
<b>DATE-TIME_HAS_UNEVEN_INTERVALS</b>	Message code for when the datetime values have uneven intervals.
<b>DATE-TIME_INFORMATION_NOT_FOUND</b>	Message code for when datetime information can not be found or is in an unaccepted format.
<b>DATE-TIME_IS_NOT_MONOTONIC</b>	Message code for when the datetime values are not monotonically increasing.
<b>HAS_ID_COLUMNS</b>	Message code for data that has ID columns.
<b>HAS_OUTLIERS</b>	Message code for when outliers are detected.
<b>HIGH_VARIANCE</b>	Message code for when high variance is detected for cross-validation.
<b>HIGHLY_NULL_COLUMNS</b>	Message code for highly null columns.

continues on next page

Table 142 – continued from previous page

<b>HIGHLY_NULL_ROWS</b>	Message code for highly null rows.
<b>IS_MULTICOLLINEAR</b>	Message code for when data is potentially multicollinear.
<b>MIS-MATCHED_INDICES</b>	Message code for when input target and features have mismatched indices.
<b>MIS-MATCHED_INDICES_ORDER</b>	Message code for when input target and features have mismatched indices order. The two have the same index values, but shuffled.
<b>MIS-MATCHED_LENGTHS</b>	Message code for when input target and features have different lengths.
<b>NATURAL_LANGUAGE_HAS_NAN</b>	Message code for when input natural language columns contain NaN values.
<b>NO_VARIANCE</b>	Message code for when data has no variance (1 unique value).
<b>NO_VARIANCE_WITH_NULL</b>	Message code for when data has one unique value and NaN values.
<b>NOT_UNIQUE_ENOUGH</b>	Message code for when data does not possess enough unique values.
<b>TARGET_BINARY_NOT_TWO_UNIQUE_VALUES</b>	Message code for target data for a binary classification problem that does not have two unique values.
<b>TARGET_HAS_NULL</b>	Message code for target data that has null values.
<b>TARGET_INCOMPATIBLE_OBJECTIVE</b>	Message code for target data that has incompatible values for the specified objective
<b>TARGET_IS_EMPTY_OR_FULLY_NULL</b>	Message code for target data that is empty or has all null values.
<b>TARGET_IS_NONE</b>	Message code for when target is None.
<b>TARGET_LEAKAGE</b>	Message code for when target leakage is detected.
<b>TARGET_LOGNORMAL_DISTRIBUTION</b>	Message code for target data with a lognormal distribution.
<b>TARGET_MULTICLASS_HIGH_UNIQUE_CLASS</b>	Message code for target data for a multi classification problem that has an abnormally large number of target values.
<b>TARGET_MULTICLASS_NOT_ENOUGH_CLASSES</b>	Message code for target data for a multi classification problem that does not have more than the minimum number of classes.
<b>TARGET_MULTICLASS_NOT_TWO_EXAMPLES_PER_CLASS</b>	Message code for target data for a multi classification problem that does not have two examples per class.
<b>TARGET_UNSUPPORTED_PROBLEM_TYPE</b>	Message code for target data that is being checked against an unsupported problem type.
<b>TARGET_UNSUPPORTED_TYPE</b>	Message code for target data that is of an unsupported type.
<b>TOO_SPARSE</b>	Message code for when multiclass data has values that are too sparsely populated.
<b>TOO_UNIQUE</b>	Message code for when data possesses too many unique values.

### Methods

<i>name</i>	The name of the Enum member.
<i>value</i>	The value of the Enum member.

**name** (*self*)  
The name of the Enum member.

**value** (*self*)  
The value of the Enum member.

```
class evalml.data_checks.DataCheckMessageType
```

Enum for type of data check message: WARNING or ERROR.

#### Attributes

<b>ERROR</b>	Error message returned by a data check.
<b>WARNING</b>	Warning message returned by a data check.

#### Methods

<i>name</i>	The name of the Enum member.
<i>value</i>	The value of the Enum member.

**name** (*self*)

The name of the Enum member.

**value** (*self*)

The value of the Enum member.

**class** evalml.data\_checks.**DataChecks** (*data\_checks=None, data\_check\_params=None*)

A collection of data checks.

#### Methods

<i>validate</i>	Inspects and validates the input data against data checks and returns a list of warnings and errors if applicable.
-----------------	--

**validate** (*self, X, y=None*)

Inspects and validates the input data against data checks and returns a list of warnings and errors if applicable.

#### Parameters

- **X** (*pd.DataFrame, np.ndarray*) – The input data of shape [n\_samples, n\_features]
- **y** (*pd.Series, np.ndarray*) – The target data of length [n\_samples]

**Returns** Dictionary containing DataCheckMessage objects

**Return type** dict

**class** evalml.data\_checks.**DataCheckWarning** (*message, data\_check\_name, message\_code=None, details=None*)

DataCheckMessage subclass for warnings returned by data checks.

#### Attributes

<b>message_type</b>	DataCheckMessageType.WARNING
---------------------	------------------------------

#### Methods

<i>to_dict</i>
----------------

**to\_dict** (*self*)

**class** evalml.data\_checks.DateTimeFormatDataCheck (datetime\_column='index')

Checks if the datetime column has equally spaced intervals and is monotonically increasing or decreasing in order to be supported by time series estimators.

**Parameters** **datetime\_column** (*str*, *int*) – The name of the datetime column. If the datetime values are in the index, then pass “index”.

#### Methods

<i>name</i>	Returns a name describing the data check.
<i>validate</i>	Checks if the target data has equal intervals and is sorted.

**name** (*cls*)

Returns a name describing the data check.

**validate** (*self*, *X*, *y*)

Checks if the target data has equal intervals and is sorted.

#### Parameters

- **X** (*pd.DataFrame*, *np.ndarray*) – Features.
- **y** (*pd.Series*, *np.ndarray*) – Target data.

**Returns** List with DataCheckErrors if unequal intervals are found in the datetime column.

**Return type** dict (*DataCheckError*)

#### Example

```
>>> from pandas as pd
>>> X = pd.DataFrame(pd.date_range("January 1, 2021", periods=8), columns=[
↳ "dates"])
>>> y = pd.Series([1, 2, 4, 2, 1, 2, 3, 1])
>>> X.iloc[7] = "January 9, 2021"
>>> datetime_format_check = DateTimeFormatDataCheck()
>>> assert datetime_format_check.validate(X, y) == {"errors": [{"message":
↳ "No frequency could be detected in dates, possibly due to uneven intervals.
↳ ",
↳ "data_
↳ check_name": "EqualIntervalDataCheck",
↳ "level": "error",
↳ "code": "DATETIME_HAS_UNEVEN_
↳ INTERVALS",
↳ "details": {}}, {"message": "No frequency could be detected in dates, possibly due to uneven intervals.
↳ ",
↳ "data_
↳ check_name": "EqualIntervalDataCheck",
↳ "level": "error",
↳ "code": "DATETIME_HAS_UNEVEN_
↳ INTERVALS",
↳ "details": {}}, {"message": "No frequency could be detected in dates, possibly due to uneven intervals.
↳ ",
↳ "data_
↳ check_name": "EqualIntervalDataCheck",
↳ "level": "error",
↳ "code": "DATETIME_HAS_UNEVEN_
↳ INTERVALS",
↳ "details": {}}], "warnings": [], "actions": []}
```

**class** evalml.data\_checks.DateTimeNaNDataCheck

Checks each column in the input for datetime features and will issue an error if NaN values are present.

#### Methods

<i>name</i>	Returns a name describing the data check.
<i>validate</i>	Checks if any datetime columns contain NaN values.

**name** (*cls*)

Returns a name describing the data check.

**validate** (*self*, *X*, *y=None*)

Checks if any datetime columns contain NaN values.

#### Parameters

- **X** (*pd.DataFrame*, *np.ndarray*) – Features.
- **y** (*pd.Series*, *np.ndarray*) – Ignored. Defaults to None.

**Returns** dict with a `DataCheckError` if NaN values are present in datetime columns.

**Return type** dict

#### Example

```
>>> import pandas as pd
>>> import woodwork as ww
>>> import numpy as np
>>> dates = np.arange(np.datetime64('2017-01-01'), np.datetime64('2017-01-08
↳'))
>>> dates[0] = np.datetime64('NaT')
>>> df = pd.DataFrame(dates, columns=['index'])
>>> df.ww.init()
>>> dt_nan_check = DateTimeNaNDataCheck()
>>> assert dt_nan_check.validate(df) == {"warnings": [],
...                                     "actions": [],
...                                     "errors": [
↳[DataCheckError(message='Input datetime column(s) (index) contains NaN
↳values. Please impute NaN values or drop these rows or columns.',
...                                     data_
↳check_name=DateTimeNaNDataCheck.name,
...                                     ]
↳message_code=DataCheckMessageCode.DATETIME_HAS_NAN,
...                                     ]
↳details={"columns": 'index'}).to_dict()]}
```

**class** evalml.data\_checks.**DefaultDataChecks** (*problem\_type*, *objective*, *n\_splits=3*, *date-time\_column=None*)

A collection of basic data checks that is used by AutoML by default. Includes:

- *HighlyNullDataCheck*
- *HighlyNullRowsDataCheck*
- *IDColumnsDataCheck*
- *TargetLeakageDataCheck*
- *InvalidTargetDataCheck*
- *NoVarianceDataCheck*
- *ClassImbalanceDataCheck* (for classification problem types)
- *DateTimeNaNDataCheck*
- *NaturalLanguageNaNDataCheck*
- *TargetDistributionDataCheck* (for regression problem types)
- *DateTimeFormatDataCheck* (for time series problem types)

**Parameters**

- **problem\_type** (*str*) – The problem type that is being validated. Can be regression, binary, or multiclass.
- **objective** (*str* or *ObjectiveBase*) – Name or instance of the objective class.
- **n\_splits** (*int*) – The number of splits as determined by the data splitter being used. Defaults to 3.
- **datetime\_column** (*str*) – The name of the column containing datetime information to be used for time series problems.
- **to "index" indicating that the datetime information is in the index of X or y. (Default) –**

**Methods**


---

<i>validate</i>	Inspects and validates the input data against data checks and returns a list of warnings and errors if applicable.
-----------------	--

---

**validate** (*self*, *X*, *y=None*)

Inspects and validates the input data against data checks and returns a list of warnings and errors if applicable.

**Parameters**

- **X** (*pd.DataFrame*, *np.ndarray*) – The input data of shape [n\_samples, n\_features]
- **y** (*pd.Series*, *np.ndarray*) – The target data of length [n\_samples]

**Returns** Dictionary containing DataCheckMessage objects

**Return type** dict

**class** evalml.data\_checks.**EmptyDataChecks** (*data\_checks=None*)

A collection of data checks.

**Methods**


---

<i>validate</i>	Inspects and validates the input data against data checks and returns a list of warnings and errors if applicable.
-----------------	--

---

**validate** (*self*, *X*, *y=None*)

Inspects and validates the input data against data checks and returns a list of warnings and errors if applicable.

**Parameters**

- **X** (*pd.DataFrame*, *np.ndarray*) – The input data of shape [n\_samples, n\_features]
- **y** (*pd.Series*, *np.ndarray*) – The target data of length [n\_samples]

**Returns** Dictionary containing DataCheckMessage objects

**Return type** dict

**class** evalml.data\_checks.**HighlyNullDataCheck** (*pct\_null\_col\_threshold=0.95*,  
*pct\_null\_row\_threshold=0.95*)

Checks if there are any highly-null columns and rows in the input.

### Parameters

- **pct\_null\_col\_threshold** (*float*) – If the percentage of NaN values in an input feature exceeds this amount, that column will be considered highly-null. Defaults to 0.95.
- **pct\_null\_row\_threshold** (*float*) – If the percentage of NaN values in an input row exceeds this amount, that row will be considered highly-null. Defaults to 0.95.

### Methods

<i>name</i>	Returns a name describing the data check.
<i>validate</i>	Checks if there are any highly-null columns or rows in the input.

#### **name** (*cls*)

Returns a name describing the data check.

#### **validate** (*self*, *X*, *y=None*)

Checks if there are any highly-null columns or rows in the input.

### Parameters

- **X** (*pd.DataFrame*, *np.ndarray*) – Features.
- **y** (*pd.Series*, *np.ndarray*) – Ignored.

**Returns** dict with a DataCheckWarning if there are any highly-null columns or rows.

**Return type** dict

### Example

```
>>> import pandas as pd
>>> class SeriesWrap():
...     def __init__(self, series):
...         self.series = series
...
...     def __eq__(self, series_2):
...         return all(self.series.eq(series_2.series))
...
>>> df = pd.DataFrame({
...     'lots_of_null': [None, None, None, None, 5],
...     'no_null': [1, 2, 3, 4, 5]
... })
>>> null_check = HighlyNullDataCheck(pct_null_col_threshold=0.50, pct_null_
↪row_threshold=0.50)
>>> validation_results = null_check.validate(df)
>>> validation_results['warnings'][0]['details']['pct_null_cols'] =
↪SeriesWrap(validation_results['warnings'][0]['details']['pct_null_cols'])
>>> highly_null_rows = SeriesWrap(pd.Series([0.5, 0.5, 0.5, 0.5]))
>>> assert validation_results== {"errors": [],
↪    "warnings": [{"message": "4 out of 5 rows are more than 50.0%
↪null",
↪check_name": "HighlyNullDataCheck",
↪    "level": "warning",
↪    "code": "HIGHLY_NULL_ROWS",
↪    "details": {"pct_null_cols": highly_null_
↪rows}},
↪    "": "Column 'lots_of_null' is 50.0% or more null",
↪    "data_check_name": "HighlyNullDataCheck",
↪    "level": "warning",
↪    "code": "HIGHLY_NULL_COLS",
↪    "details": {"column": "lots_of_null", "pct_null_rows": 0.8}}],
↪    "actions": [{"code": "DROP_ROWS", "metadata": {"rows": [0, 1, 2, 3]}}],
```

(continues on next page)



(continued from previous page)

---

**class** evalml.data\_checks.IDColumnsDataCheck (*id\_threshold=1.0*)

Check if any of the features are likely to be ID columns.

**Parameters** *id\_threshold* (*float*) – The probability threshold to be considered an ID column.  
Defaults to 1.0.

### Methods

<i>name</i>	Returns a name describing the data check.
<i>validate</i>	Check if any of the features are likely to be ID columns. Currently performs these simple checks:

**name** (*cls*)

Returns a name describing the data check.

**validate** (*self, X, y=None*)

Check if any of the features are likely to be ID columns. Currently performs these simple checks:

- column name is “id”
- column name ends in “\_id”
- column contains all unique values (and is categorical / integer type)

**Parameters** *X* (*pd.DataFrame, np.ndarray*) – The input features to check

**Returns** A dictionary of features with column name or index and their probability of being ID columns

**Return type** dict

### Example

```
>>> import pandas as pd
>>> df = pd.DataFrame({
...     'df_id': [0, 1, 2, 3, 4],
...     'x': [10, 42, 31, 51, 61],
...     'y': [42, 54, 12, 64, 12]
... })
>>> id_col_check = IDColumnsDataCheck()
>>> assert id_col_check.validate(df) == {"errors": [],
↳                                     "warnings": [{"message": "Column 'df_id' is
↳ 100.0% or more likely to be an ID column",
↳                                     "data_check_name": "IDColumnsDataCheck",
↳                                     "level":
↳ "warning",
↳ "code": "HAS_ID_COLUMN",
↳         "details": {"column": "df_id"}},
↳         "actions": [{"code": "DROP_COL",
↳                     "metadata": {"column": "df_id
↳ "}]}}
```

**class** evalml.data\_checks.InvalidTargetDataCheck (*problem\_type, objective,*  
*n\_unique=100*)

Checks if the target data contains missing or invalid values.

**Parameters**

- **problem\_type** (*str* or *ProblemTypes*) – The specific problem type to data check for. e.g. ‘binary’, ‘multiclass’, ‘regression’, ‘time series regression’
- **objective** (*str* or *ObjectiveBase*) – Name or instance of the objective class.
- **n\_unique** (*int*) – Number of unique target values to store when problem type is binary and target incorrectly has more than 2 unique values. Non-negative integer. If None, stores all unique values. Defaults to 100.

**Attributes**

<b>multi-class_continuous_threshold</b>	0.05
---	------

**Methods**

<i>name</i>	Returns a name describing the data check.
<i>validate</i>	Checks if the target data contains missing or invalid values.

**name** (*cls*)

Returns a name describing the data check.

**validate** (*self*, *X*, *y*)

Checks if the target data contains missing or invalid values.

**Parameters**

- **X** (*pd.DataFrame*, *np.ndarray*) – Features. Ignored.
- **y** (*pd.Series*, *np.ndarray*) – Target data to check for invalid values.

**Returns** List with DataCheckErrors if any invalid values are found in the target data.

**Return type** dict (*DataCheckError*)

**Example**

```
>>> import pandas as pd
>>> X = pd.DataFrame({"col": [1, 2, 3, 1]})
>>> y = pd.Series([0, 1, None, None])
>>> target_check = InvalidTargetDataCheck('binary', 'Log Loss Binary')
>>> assert target_check.validate(X, y) == {"errors": [{"message": "2 row(s) (50.0%) of target values are null",
↳                                     "data_check_name": "InvalidTargetDataCheck",
↳                                     "level": "error",
↳                                     "code": "TARGET_HAS_NULL",
↳                                     "details": {"num_null_rows": 2, "pct_null_rows": 50}}],
↳                                     "warnings": [],
↳                                     "actions": [{"code": 'IMPUTE_COL',
↳ 'metadata': {'column': None, 'impute_strategy': 'most_frequent', 'is_target': True}}]}
```

**class** evalml.data\_checks.MulticollinearityDataCheck (*threshold=0.9*)

Check if any set features are likely to be multicollinear.

**Parameters** `threshold` (*float*) – The threshold to be considered. Defaults to 0.9.

#### Methods

<code>name</code>	Returns a name describing the data check.
<code>validate</code>	Check if any set of features are likely to be multicollinear.

**name** (*cls*)

Returns a name describing the data check.

**validate** (*self*, *X*, *y=None*)

Check if any set of features are likely to be multicollinear.

**Parameters** *X* (*pd.DataFrame*, *np.ndarray*) – The input features to check

**Returns** dict with a DataCheckWarning if there are any potentially multicollinear columns.

**Return type** dict

**class** evalml.data\_checks.NaturalLanguageNaNDataCheck

Checks each column in the input for natural language features and will issue an error if NaN values are present.

#### Methods

<code>name</code>	Returns a name describing the data check.
<code>validate</code>	Checks if any natural language columns contain NaN values.

**name** (*cls*)

Returns a name describing the data check.

**validate** (*self*, *X*, *y=None*)

Checks if any natural language columns contain NaN values.

#### Parameters

- *X* (*pd.DataFrame*, *np.ndarray*) – Features.
- *y* (*pd.Series*, *np.ndarray*) – Ignored. Defaults to None.

**Returns** dict with a DataCheckError if NaN values are present in natural language columns.

**Return type** dict

#### Example

```
>>> import pandas as pd
>>> import woodwork as ww
>>> import numpy as np
>>> data = pd.DataFrame()
>>> data['A'] = [None, "string_that_is_long_enough_for_natural_language"]
>>> data['B'] = ['string_that_is_long_enough_for_natural_language', 'string_
↳ that_is_long_enough_for_natural_language']
>>> data['C'] = np.random.randint(0, 3, size=len(data))
>>> data.ww.init(logical_types={'A': 'NaturalLanguage', 'B': 'NaturalLanguage
↳ '})
>>> nl_nan_check = NaturalLanguageNaNDataCheck()
```

(continues on next page)

(continued from previous page)

```

>>> assert nl_nan_check.validate(data) == {
...     "warnings": [],
...     "actions": [],
...     "errors": [DataCheckError(message='Input natural language_
↪column(s) (A) contains NaN values. Please impute NaN values or drop these_
↪rows or columns.',
...                               data_check_name=NaturalLanguageNaNDataCheck.name,
...                               message_code=DataCheckMessageCode.NATURAL_LANGUAGE_
↪HAS_NAN,
...                               details={"columns": 'A'}).to_dict()]
...     }

```

**class** evalml.data\_checks.NoVarianceDataCheck (count\_nan\_as\_value=False)

Check if the target or any of the features have no variance.

**Parameters** **count\_nan\_as\_value** (*bool*) – If True, missing values will be counted as their own unique value. Additionally, if true, will return a DataCheckWarning instead of an error if the feature has mostly missing data and only one unique value. Defaults to False.

### Methods

<i>name</i>	Returns a name describing the data check.
<i>validate</i>	Check if the target or any of the features have no variance (1 unique value).

**name** (*cls*)

Returns a name describing the data check.

**validate** (*self*, *X*, *y*)

Check if the target or any of the features have no variance (1 unique value).

### Parameters

- **X** (*pd.DataFrame*, *np.ndarray*) – The input features.
- **y** (*pd.Series*, *np.ndarray*) – The target data.

**Returns** dict of warnings/errors corresponding to features or target with no variance.

**Return type** dict

**class** evalml.data\_checks.OutliersDataCheck

Checks if there are any outliers in input data by using IQR to determine score anomalies. Columns with score anomalies are considered to contain outliers.

### Methods

<i>name</i>	Returns a name describing the data check.
<i>validate</i>	Checks if there are any outliers in a dataframe by using IQR to determine column anomalies. Column with anomalies are considered to contain outliers.

**name** (*cls*)

Returns a name describing the data check.

**validate** (*self*, *X*, *y=None*)

Checks if there are any outliers in a dataframe by using IQR to determine column anomalies. Column with anomalies are considered to contain outliers.

**Parameters**

- **X** (*pd.DataFrame*, *np.ndarray*) – Features
- **y** (*pd.Series*, *np.ndarray*) – Ignored.

**Returns** A dictionary with warnings if any columns have outliers.

**Return type** dict

**Example**

```
>>> import pandas as pd
>>> df = pd.DataFrame({
...     'x': [1, 2, 3, 4, 5],
...     'y': [6, 7, 8, 9, 10],
...     'z': [-1, -2, -3, -1201, -4]
... })
>>> outliers_check = OutliersDataCheck()
>>> assert outliers_check.validate(df) == {"warnings": [{"message":
↳ "Column(s) 'z' are likely to have outlier data.",
↳                                     "data_check_name":
↳ "OutliersDataCheck",
↳                                     "level": "warning",
↳                                     "code": "HAS_OUTLIERS",
↳                                     "details": {"columns": ["z"]}}],
↳                                     "errors": [],
↳                                     "actions": []}
```

**class** evalml.data\_checks.**SparsityDataCheck** (*problem\_type*, *threshold*,  
*unique\_count\_threshold=10*)

Checks if there are any columns with sparsely populated values in the input.

**Parameters**

- **problem\_type** (*str* or *ProblemTypes*) – The specific problem type to data check for. ‘multiclass’ or ‘time series multiclass’ is the only accepted problem type.
- **threshold** (*float*) – The threshold value, or percentage of each column’s unique values, below which, a column exhibits sparsity. Should be between 0 and 1.
- **unique\_count\_threshold** (*int*) – The minimum number of times a unique value has to be present in a column to not be considered “sparse.” Defaults to 10.

**Methods**

<i>name</i>	Returns a name describing the data check.
<i>sparsity_score</i>	This function calculates a sparsity score for the given value counts by calculating the percentage of
<i>validate</i>	Calculates what percentage of each column’s unique values exceed the count threshold and compare

**name** (*cls*)

Returns a name describing the data check.

**static sparsity\_score** (*col*, *count\_threshold=10*)

This function calculates a sparsity score for the given value counts by calculating the percentage of unique values that exceed the count\_threshold.

**Parameters**

- **col** (*pd.Series*) – Feature values.
- **count\_threshold** (*int*) – The number of instances below which a value is considered sparse. Default is 10.

**Returns** Sparsity score, or the percentage of the unique values that exceed count\_threshold.

**Return type** (float)

**validate** (*self, X, y=None*)

Calculates what percentage of each column's unique values exceed the count threshold and compare that percentage to the sparsity threshold stored in the class instance.

**Parameters**

- **X** (*pd.DataFrame, np.ndarray*) – Features.
- **y** (*pd.Series, np.ndarray*) – Ignored.

**Returns** dict with a DataCheckWarning if there are any sparse columns.

**Return type** dict

**Example**

```
>>> import pandas as pd
>>> df = pd.DataFrame({
...     'sparse': [float(x) for x in range(100)],
...     'not_sparse': [float(1) for x in range(100)]
... })
>>> sparsity_check = SparsityDataCheck(problem_type="multiclass", threshold=0.
↳5, unique_count_threshold=10)
>>> assert sparsity_check.validate(df) == {"errors": [],
↳                                     "warnings": [{"message": "Input columns_
↳(sparse) for multiclass problem type are too sparse.",
↳                                     "data_check_name": "SparsityDataCheck
↳                                     "level":
↳ "warning",
↳                                     "code
↳ ": "TOO_SPARSE",
↳ "details": {"column": "sparse", 'sparsity_score': 0.0}}},
↳                                     "actions": [{"code": "DROP_COL",
↳                                     "metadata": {
↳ "column": "sparse"}}]}
```

**class** evalml.data\_checks.TargetDistributionDataCheck

Checks if the target data contains certain distributions that may need to be transformed prior training to improve model performance.

**Methods**

<i>name</i>	Returns a name describing the data check.
<i>validate</i>	Checks if the target data has a certain distribution.

**name** (*cls*)

Returns a name describing the data check.

**validate** (*self, X, y*)

Checks if the target data has a certain distribution.

**Parameters**

- **X** (*pd.DataFrame*, *np.ndarray*) – Features. Ignored.
- **y** (*pd.Series*, *np.ndarray*) – Target data to check for underlying distributions.

**Returns** List with *DataCheckErrors* if certain distributions are found in the target data.

**Return type** dict (*DataCheckError*)

**Example**

```
>>> from scipy.stats import lognorm
>>> X = None
>>> y = [0.946, 0.972, 1.154, 0.954, 0.969, 1.222, 1.038, 0.999, 0.973, 0.897]
>>> target_check = TargetDistributionDataCheck()
>>> assert target_check.validate(X, y) == {"errors": [],
    "warnings": [{"message": "Target may have
    a lognormal distribution.",
    "data_check_name": "TargetDistributionDataCheck",
    "level":
    "warning",
    "code": "TARGET_LOGNORMAL_DISTRIBUTION",
    "details": {"shapiro-statistic/pvalue":
    '0.84/0.045'}}]},
    "actions": [{"code": 'TRANSFORM_TARGET', 'metadata': {'column': None,
    'transformation_strategy': 'lognormal', 'is_target': True}]}
```

**class** evalml.data\_checks.**TargetLeakageDataCheck** (*pct\_corr\_threshold=0.95*,  
*method='mutual'*)

Check if any of the features are highly correlated with the target by using mutual information or Pearson correlation.

If *method='mutual'*, this data check uses mutual information and supports all target and feature types. Otherwise, if *method='pearson'*, it uses Pearson correlation and only supports binary with numeric and boolean dtypes. Pearson correlation returns a value in [-1, 1], while mutual information returns a value in [0, 1].

**Parameters**

- **pct\_corr\_threshold** (*float*) – The correlation threshold to be considered leakage. Defaults to 0.95.
- **method** (*string*) – The method to determine correlation. Use ‘mutual’ for mutual information, otherwise ‘pearson’ for Pearson correlation. Defaults to ‘mutual’.

**Methods**

<i>name</i>	Returns a name describing the data check.
<i>validate</i>	Check if any of the features are highly correlated with the target by using mutual information or Pearson correlation.

**name** (*cls*)

Returns a name describing the data check.

**validate** (*self*, *X*, *y*)

Check if any of the features are highly correlated with the target by using mutual information or Pearson correlation.

If *method*='mutual', supports all target and feature types. Otherwise, if *method*='pearson' only supports binary with numeric and boolean dtypes. Pearson correlation returns a value in [-1, 1], while mutual information returns a value in [0, 1].

#### Parameters

- **X** (*pd.DataFrame*, *np.ndarray*) – The input features to check
- **y** (*pd.Series*, *np.ndarray*) – The target data

**Returns** dict with a *DataCheckWarning* if target leakage is detected.

**Return type** dict (*DataCheckWarning*)

#### Example

```
>>> import pandas as pd
>>> X = pd.DataFrame({
...     'leak': [10, 42, 31, 51, 61],
...     'x': [42, 54, 12, 64, 12],
...     'y': [13, 5, 13, 74, 24],
... })
>>> y = pd.Series([10, 42, 31, 51, 40])
>>> target_leakage_check = TargetLeakageDataCheck(pct_corr_threshold=0.95)
>>> assert target_leakage_check.validate(X, y) == {"warnings": [{"message":
↳ "Column 'leak' is 95.0% or more correlated with the target",
↳                                     "data_check_
↳ name": "TargetLeakageDataCheck",
↳                                     "level": "warning",
↳                                     "code": "TARGET_LEAKAGE
↳ ",
↳     "details": {"column": "leak"}},
↳     "errors": [],
↳     "actions": [{"code": "DROP_COL",
↳                                     "metadata": {"column":
↳     "leak"}}]}
```

**class** evalml.data\_checks.UniquenessDataCheck (*problem\_type*, *threshold*=0.5)

Checks if there are any columns in the input that are either too unique for classification problems or not unique enough for regression problems.

#### Parameters

- **problem\_type** (*str* or *ProblemTypes*) – The specific problem type to data check for. e.g. 'binary', 'multiclass', 'regression', 'time series regression'
- **threshold** (*float*) – The threshold to set as an upper bound on uniqueness for classification type problems or lower bound on for regression type problems. Defaults to 0.50.

#### Methods

<i>name</i>	Returns a name describing the data check.
<i>uniqueness_score</i>	This function calculates a uniqueness score for the provided field. NaN values are
<i>validate</i>	Checks if there are any columns in the input that are too unique in the case of classification

**name** (*cls*)



Returns a name describing the data check.

**static uniqueness\_score** (*col*)

This function calculates a uniqueness score for the provided field. NaN values are not considered as unique values in the calculation.

Based on the Herfindahl–Hirschman Index.

**Parameters** *col* (*pd.Series*) – Feature values.

**Returns** Uniqueness score.

**Return type** (float)

**validate** (*self, X, y=None*)

Checks if there are any columns in the input that are too unique in the case of classification problems or not unique enough in the case of regression problems.

**Parameters**

- *X* (*pd.DataFrame*, *np.ndarray*) – Features.
- *y* (*pd.Series*, *np.ndarray*) – Ignored. Defaults to None.

**Returns**

**dict with a DataCheckWarning if there are any too unique or not unique enough columns.**

**Return type** dict

## Example

```
>>> import pandas as pd
>>> df = pd.DataFrame({
...     'regression_unique_enough': [float(x) for x in range(100)],
...     'regression_not_unique_enough': [float(1) for x in range(100)]
... })
>>> uniqueness_check = UniquenessDataCheck(problem_type="regression",
↳ threshold=0.8)
>>> assert uniqueness_check.validate(df) == {"errors": [],
↳                                     "warnings": [{"message": "Input
↳ columns (regression_not_unique_enough) for regression problem type are not
↳ unique enough.",
↳     "data_check_name": "UniquenessDataCheck",
↳     "level": "warning",
↳     "code": "NOT_UNIQUE_ENOUGH",
↳     "details": {
↳ "column": "regression_not_unique_enough", 'uniqueness_score': 0.0}}},
↳     "actions": [{"code":
↳ "DROP_COL",
↳     "metadata": {"column": "regression_not_unique_enough"}}]}
```

## Demos

### Submodules

#### breast\_cancer

### Module Contents

#### Functions

---

<code>load_breast_cancer</code>	Load breast cancer dataset. Binary classification problem.
---------------------------------	--

---

### Contents

`evalml.demos.breast_cancer.load_breast_cancer()`

Load breast cancer dataset. Binary classification problem.

**Returns** X and y

**Return type** (pd.DataFrame, pd.Series)

#### churn

### Module Contents

#### Functions

---

<code>load_churn</code>	Load credit card fraud dataset.
-------------------------	---------------------------------

---

### Contents

`evalml.demos.churn.load_churn(n_rows=None, verbose=True)`

**Load credit card fraud dataset.** The fraud dataset can be used for binary classification problems.

#### Parameters

- **n\_rows** (*int*) – Number of rows from the dataset to return
- **verbose** (*bool*) – Whether to print information about features and labels

**Returns** X and y

**Return type** (pd.DataFrame, pd.Series)

## diabetes

### Module Contents

### Functions

---

<code>load_diabetes</code>	Load diabetes dataset. Regression problem
----------------------------	---

---

### Contents

```
evalml.demos.diabetes.load_diabetes()
```

Load diabetes dataset. Regression problem

**Returns** X and y

**Return type** (pd.DataFrame, pd.Series)

## fraud

### Module Contents

### Functions

---

<code>load_fraud</code>	Load credit card fraud dataset.
-------------------------	---------------------------------

---

### Contents

```
evalml.demos.fraud.load_fraud(n_rows=None, verbose=True)
```

**Load credit card fraud dataset.** The fraud dataset can be used for binary classification problems.

#### Parameters

- **n\_rows** (*int*) – Number of rows from the dataset to return
- **verbose** (*bool*) – Whether to print information about features and labels

**Returns** X and y

**Return type** (pd.DataFrame, pd.Series)

## wine

### Module Contents

#### Functions

---

<code>load_wine</code>	Load wine dataset. Multiclass problem.
------------------------	--

---

#### Contents

`evalml.demos.wine.load_wine()`  
Load wine dataset. Multiclass problem.

**Returns** X and y

**Return type** (pd.DataFrame, pd.Series)

### Package Contents

#### Functions

---

<code>load_breast_cancer</code>	Load breast cancer dataset. Binary classification problem.
<code>load_churn</code>	Load credit card fraud dataset.
<code>load_diabetes</code>	Load diabetes dataset. Regression problem
<code>load_fraud</code>	Load credit card fraud dataset.
<code>load_wine</code>	Load wine dataset. Multiclass problem.

---

#### Contents

`evalml.demos.load_breast_cancer()`  
Load breast cancer dataset. Binary classification problem.

**Returns** X and y

**Return type** (pd.DataFrame, pd.Series)

`evalml.demos.load_churn(n_rows=None, verbose=True)`

**Load credit card fraud dataset.** The fraud dataset can be used for binary classification problems.

##### Parameters

- **n\_rows** (*int*) – Number of rows from the dataset to return
- **verbose** (*bool*) – Whether to print information about features and labels

**Returns** X and y

**Return type** (pd.DataFrame, pd.Series)

```
evalml.demos.load_diabetes()
```

Load diabetes dataset. Regression problem

**Returns** X and y

**Return type** (pd.DataFrame, pd.Series)

```
evalml.demos.load_fraud(n_rows=None, verbose=True)
```

**Load credit card fraud dataset.** The fraud dataset can be used for binary classification problems.

**Parameters**

- **n\_rows** (*int*) – Number of rows from the dataset to return
- **verbose** (*bool*) – Whether to print information about features and labels

**Returns** X and y

**Return type** (pd.DataFrame, pd.Series)

```
evalml.demos.load_wine()
```

Load wine dataset. Multiclass problem.

**Returns** X and y

**Return type** (pd.DataFrame, pd.Series)

## Exceptions

## Submodules

## exceptions

## Module Contents

## Classes Summary

---

*PartialDependenceErrorCode*

Enum identifying the type of error encountered in partial dependence.

---

## Exceptions Summary

## Contents

**exception** evalml.exceptions.exceptions.**AutoMLSearchException**

Exception raised when all pipelines in an automl batch return a score of NaN for the primary objective.

**exception** evalml.exceptions.exceptions.**ComponentNotYetFittedError**

An exception to be raised when predict/predict\_proba/transform is called on a component without fitting first.

**exception** evalml.exceptions.exceptions.**DataCheckInitError**

Exception raised when a data check can't initialize with the parameters given.

**exception** evalml.exceptions.exceptions.**EnsembleMissingPipelinesError**

An exception raised when an ensemble is missing *estimators* (list) as a parameter.

**exception** evalml.exceptions.exceptions.**MethodPropertyNotFoundError**

Exception to raise when a class is does not have an expected method or property.

**exception** evalml.exceptions.exceptions.**MissingComponentError**

An exception raised when a component is not found in all\_components()

**exception** evalml.exceptions.exceptions.**NoPositiveLabelException**

Exception when a particular classification label for the 'positive' class cannot be found in the column index or unique values

**exception** evalml.exceptions.exceptions.**NullsInColumnWarning**

Warning thrown when there are null values in the column of interest

**exception** evalml.exceptions.exceptions.**ObjectiveCreationError**

Exception when get\_objective tries to instantiate an objective and required args are not provided.

**exception** evalml.exceptions.exceptions.**ObjectiveNotFoundError**

Exception to raise when specified objective does not exist.

**exception** evalml.exceptions.exceptions.**ParameterNotUsedWarning** (*components*)

Warning thrown when a pipeline parameter isn't used in a defined pipeline's component graph during initialization.

**exception** evalml.exceptions.exceptions.**PartialDependenceError** (*message, code*)

Exception raised for all errors that partial dependence can raise.

**class** evalml.exceptions.exceptions.**PartialDependenceErrorCode**

Enum identifying the type of error encountered in partial dependence.

### Attributes

<b>ALL_OTHER_ERRORS</b>	errors
<b>COMPUTED_PERCENTILES_TOO_CLOSE</b>	computed_percentiles_too_close
<b>FEATURE_IS_ALL_NANS</b>	feature_is_all_nans
<b>FEATURE_IS_MOSTLY_ONE_VALUE</b>	feature_is_mostly_one_value
<b>FEATURES_ARGUMENT_INCORRECT_TYPES</b>	features_argument_incorrect_types
<b>ICE_PLOT_REQUESTED_FOR_TWO_WAY_PLOT</b>	ice_dependence_plot
<b>INVALID_CLASS_LABEL</b>	invalid_class_label_requested_for_plot
<b>INVALID_FEATURE_TYPE</b>	invalid_feature_type
<b>PIPELINE_IS_BASELINE</b>	baseline
<b>TOO_MANY_FEATURES</b>	features
<b>TWO_WAY_REQUESTED_FOR_DATES</b>	dates
<b>UNFITTED_PIPELINE</b>	unfitted_pipeline

## Methods

<i>name</i>	The name of the Enum member.
<i>value</i>	The value of the Enum member.

**name** (*self*)

The name of the Enum member.

**value** (*self*)

The value of the Enum member.

**exception** evalml.exceptions.exceptions.**PipelineNotFoundError**

An exception raised when a particular pipeline is not found in automl search results

**exception** evalml.exceptions.exceptions.**PipelineNotYetFittedError**

An exception to be raised when predict/predict\_proba/transform is called on a pipeline without fitting first.

**exception** evalml.exceptions.exceptions.**PipelineScoreError** (*exceptions*,  
*scored\_successfully*)

An exception raised when a pipeline errors while scoring any objective in a list of objectives.

## Parameters

- **exceptions** (*dict*) – A dictionary mapping an objective name (str) to a tuple of the form (exception, traceback). All of the objectives that errored will be stored here.
- **scored\_successfully** (*dict*) – A dictionary mapping an objective name (str) to a score value. All of the objectives that did not error will be stored here.

## Package Contents

### Classes Summary

---

<code>PartialDependenceErrorCode</code>	Enum identifying the type of error encountered in partial dependence.
---	---

---

### Exceptions Summary

---

#### Contents

- exception** `evalml.exceptions.AutoMLSearchException`  
Exception raised when all pipelines in an automl batch return a score of NaN for the primary objective.
- exception** `evalml.exceptions.ComponentNotYetFittedError`  
An exception to be raised when `predict/predict_proba/transform` is called on a component without fitting first.
- exception** `evalml.exceptions.DataCheckInitError`  
Exception raised when a data check can't initialize with the parameters given.
- exception** `evalml.exceptions.EnsembleMissingPipelinesError`  
An exception raised when an ensemble is missing *estimators* (list) as a parameter.
- exception** `evalml.exceptions.MethodPropertyNotFoundError`  
Exception to raise when a class does not have an expected method or property.
- exception** `evalml.exceptions.MissingComponentError`  
An exception raised when a component is not found in `all_components()`
- exception** `evalml.exceptions.NoPositiveLabelException`  
Exception when a particular classification label for the 'positive' class cannot be found in the column index or unique values
- exception** `evalml.exceptions.NullsInColumnWarning`  
Warning thrown when there are null values in the column of interest
- exception** `evalml.exceptions.ObjectiveCreationError`  
Exception when `get_objective` tries to instantiate an objective and required args are not provided.
- exception** `evalml.exceptions.ObjectiveNotFoundError`  
Exception to raise when specified objective does not exist.
- exception** `evalml.exceptions.ParameterNotUsedWarning` (*components*)  
Warning thrown when a pipeline parameter isn't used in a defined pipeline's component graph during initialization.
- exception** `evalml.exceptions.PartialDependenceError` (*message, code*)  
Exception raised for all errors that partial dependence can raise.
- class** `evalml.exceptions.PartialDependenceErrorCode`  
Enum identifying the type of error encountered in partial dependence.

#### Attributes



<b>ALL_OTHER_ERRORS</b>	errors
<b>COMPUTED_PERCENTILES_TOO_CLOSE</b>	computed_percentiles_too_close
<b>FEATURE_IS_ALL_NANS</b>	feature_is_all_nans
<b>FEATURE_IS_MOSTLY_ONE_VALUE</b>	feature_is_mostly_one_value
<b>FEATURES_ARGUMENT_INCORRECT_TYPES</b>	features_argument_incorrect_types
<b>ICE_PLOT_REQUESTED_FOR_TWO_WAY_PLOT</b>	ice_dependence_plot
<b>INVALID_CLASS_LABEL</b>	invalid_class_label_requested_for_plot
<b>INVALID_FEATURE_TYPE</b>	invalid_feature_type
<b>PIPELINE_IS_BASELINE</b>	baseline
<b>TOO_MANY_FEATURES</b>	features
<b>TWO_WAY_REQUESTED_FOR_DATES</b>	dates
<b>UNFITTED_PIPELINE</b>	unfitted_pipeline

## Methods

<i>name</i>	The name of the Enum member.
<i>value</i>	The value of the Enum member.

**name** (*self*)

The name of the Enum member.

**value** (*self*)

The value of the Enum member.

**exception** evalml.exceptions.PipelineNotFoundError

An exception raised when a particular pipeline is not found in automl search results

**exception** evalml.exceptions.PipelineNotYetFittedError

An exception to be raised when predict/predict\_proba/transform is called on a pipeline without fitting first.

**exception** evalml.exceptions.PipelineScoreError (*exceptions, scored\_successfully*)

An exception raised when a pipeline errors while scoring any objective in a list of objectives.

## Parameters

- **exceptions** (*dict*) – A dictionary mapping an objective name (str) to a tuple of the form (exception, traceback). All of the objectives that errored will be stored here.
- **scored\_successfully** (*dict*) – A dictionary mapping an objective name (str) to a score value. All of the objectives that did not error will be stored here.

## Model Family

### Submodules

#### model\_family

### Module Contents

### Classes Summary

---

<code>ModelFamily</code>	Enum for family of machine learning models.
--------------------------	---

---

### Contents

**class** evalml.model\_family.model\_family.**ModelFamily**

Enum for family of machine learning models.

#### Attributes

<b>ARIMA</b>	ARIMA model family.
<b>BASELINE</b>	Baseline model family.
<b>CAT-BOOST</b>	CatBoost model family.
<b>DECISION_TREE</b>	Decision Tree model family.
<b>ENSEMBLE</b>	Ensemble model family.
<b>EXTRA_TREES</b>	Extra Trees model family.
<b>K_NEIGHBORS</b>	K Nearest Neighbors model family.
<b>LIGHTGBM</b>	LightGBM model family.
<b>LINEAR_MODEL</b>	Linear model family.
<b>NONE</b>	None
<b>PROPHET</b>	Prophet model family.
<b>RANDOM_FOREST</b>	Random Forest model family.
<b>SVM</b>	SVM model family.
<b>XGBOOST</b>	XGBoost model family.

#### Methods

---

<code>is_tree_estimator</code>	Checks whether the estimator's model family uses trees.
<code>name</code>	The name of the Enum member.
<code>value</code>	The value of the Enum member.

---

`is_tree_estimator(self)`

Checks whether the estimator’s model family uses trees.

**name** (*self*)

The name of the Enum member.

**value** (*self*)

The value of the Enum member.

## utils

## Module Contents

## Functions

---

*handle\_model\_family*

Handles `model_family` by either returning the `ModelFamily` or converting from a string

---

## Contents

`evalml.model_family.utils.handle_model_family(model_family)`

Handles `model_family` by either returning the `ModelFamily` or converting from a string

**Parameters** `model_family` (*str* or `ModelFamily`) – Model type that needs to be handled

**Returns** `ModelFamily`

## Package Contents

## Classes Summary

---

*ModelFamily*

Enum for family of machine learning models.

---

## Functions

---

*handle\_model\_family*

Handles `model_family` by either returning the `ModelFamily` or converting from a string

---

## Contents

`evalml.model_family.handle_model_family(model_family)`

Handles `model_family` by either returning the `ModelFamily` or converting from a string

**Parameters** `model_family` (*str* or `ModelFamily`) – Model type that needs to be handled

**Returns** `ModelFamily`

**class** `evalml.model_family.ModelFamily`

Enum for family of machine learning models.

### Attributes

<b>ARIMA</b>	ARIMA model family.
<b>BASELINE</b>	Baseline model family.
<b>CAT-BOOST</b>	CatBoost model family.
<b>DECISION_TREE</b>	Decision Tree model family.
<b>ENSEMBLE</b>	Ensemble model family.
<b>EXTRA_TREES</b>	Extra Trees model family.
<b>K_NEIGHBORS</b>	K Nearest Neighbors model family.
<b>LIGHTGBM</b>	LightGBM model family.
<b>LINEAR_MODEL</b>	Linear model family.
<b>NONE</b>	None
<b>PROPHET</b>	Prophet model family.
<b>RANDOM_FOREST</b>	Random Forest model family.
<b>SVM</b>	SVM model family.
<b>XGBOOST</b>	XGBoost model family.

### Methods

<code>is_tree_estimator</code>	Checks whether the estimator's model family uses trees.
<code>name</code>	The name of the Enum member.
<code>value</code>	The value of the Enum member.

**is\_tree\_estimator** (*self*)  
Checks whether the estimator's model family uses trees.

**name** (*self*)  
The name of the Enum member.

**value** (*self*)  
The value of the Enum member.

## Model Understanding

### Subpackages

### prediction\_explanations

### Submodules

### explainers

## Module Contents

### Classes Summary

<code>ExplainPredictionsStage</code>	Generic enumeration.
--------------------------------------	----------------------

### Functions

<code>abs_error</code>	Computes the absolute error per data point for regression problems.
<code>cross_entropy</code>	Computes Cross Entropy Loss per data point for classification problems.
<code>explain_predictions</code>	Creates a report summarizing the top contributing features for each data point in the input features.
<code>explain_predictions_best_worst</code>	Creates a report summarizing the top contributing features for the best and worst points in the dataset as measured by error to true labels.

### Attributes Summary

<code>DEFAULT_METRICS</code>
------------------------------

## Contents

`evalml.model_understanding.prediction_explanations.explainers.abs_error(y_true, y_pred)`

Computes the absolute error per data point for regression problems.

#### Parameters

- **y\_true** (`pd.Series`) – True labels.
- **y\_pred** (`pd.Series`) – Predicted values.

**Returns** `np.ndarray`

`evalml.model_understanding.prediction_explanations.explainers.cross_entropy(y_true, y_pred_proba)`

Computes Cross Entropy Loss per data point for classification problems.

#### Parameters

- **y\_true** (`pd.Series`) – True labels encoded as ints.
- **y\_pred\_proba** (`pd.DataFrame`) – Predicted probabilities. One column per class.

**Returns** `np.ndarray`

`evalml.model_understanding.prediction_explanations.explainers.DEFAULT_METRICS`

```
evalml.model_understanding.prediction_explanations.explainers.explain_predictions(pipeline,
in-
put_features=
y,
in-
dices_to_explain=
top_k_features=
in-
clude_shap_values=
in-
clude_expected_value=
out-
put_format=)
```

Creates a report summarizing the top contributing features for each data point in the input features.

XGBoost and Stacked Ensemble models, as well as CatBoost multiclass classifiers, are not currently supported.

#### Parameters

- **pipeline** (*PipelineBase*) – Fitted pipeline whose predictions we want to explain with SHAP.
- **input\_features** (*pd.DataFrame*) – Dataframe of input data to evaluate the pipeline on.
- **y** (*pd.Series*) – Labels for the input data.
- **indices\_to\_explain** (*list(int)*) – List of integer indices to explain.
- **top\_k\_features** (*int*) – How many of the highest/lowest contributing feature to include in the table for each data point. Default is 3.
- **include\_shap\_values** (*bool*) – Whether SHAP values should be included in the table. Default is False.
- **include\_expected\_value** (*bool*) – Whether the expected value should be included in the table. Default is False.
- **output\_format** (*str*) – Either “text”, “dict”, or “dataframe”. Default is “text”.

#### Returns

**str, dict, or pd.DataFrame - A report explaining the top contributing features to each prediction for each row of**  
The report will include the feature names, prediction contribution, and SHAP Value (optional).

#### Raises

- **ValueError** – if input\_features is empty.
- **ValueError** – if an output\_format outside of “text”, “dict” or “dataframe” is provided.
- **ValueError** – if the requested index falls outside the input\_feature’s boundaries.

```
evalml.model_understanding.prediction_explanations.explainers.explain_predictions_best_worst
```

Creates a report summarizing the top contributing features for the best and worst points in the dataset as measured by error to true labels.

XGBoost and Stacked Ensemble models, as well as CatBoost multiclass classifiers, are not currently supported.

#### Parameters

- **pipeline** (*PipelineBase*) – Fitted pipeline whose predictions we want to explain with SHAP.
- **input\_features** (*pd.DataFrame*) – Input data to evaluate the pipeline on.
- **y\_true** (*pd.Series*) – True labels for the input data.
- **num\_to\_explain** (*int*) – How many of the best, worst, random data points to explain.
- **top\_k\_features** (*int*) – How many of the highest/lowest contributing feature to include in the table for each data point.
- **include\_shap\_values** (*bool*) – Whether SHAP values should be included in the table. Default is False.
- **metric** (*callable*) – The metric used to identify the best and worst points in the dataset. Function must accept the true labels and predicted value or probabilities as the only arguments and lower values must be better. By default, this will be the absolute error for regression problems and cross entropy loss for classification problems.
- **output\_format** (*str*) – Either “text” or “dict”. Default is “text”.
- **callback** (*callable*) – Function to be called with incremental updates. Has the following parameters: - progress\_stage: stage of computation - time\_elapsed: total time in seconds that has elapsed since start of call

#### Returns

**str, dict, or pd.DataFrame** - A report explaining the top contributing features for the best/worst predictions in the dataset. For each of the best/worst rows of input\_features, the predicted values, true labels, metric value, feature names, prediction contribution, and SHAP Value (optional) will be listed.

#### Raises

- **ValueError** – if input\_features does not have more than twice the requested features to explain.
- **ValueError** – if y\_true and input\_features have mismatched lengths.
- **ValueError** – if an output\_format outside of “text”, “dict” or “dataframe” is provided.

**class** evalml.model\_understanding.prediction\_explanations.explainers.**ExplainPredictionsStage**  
Generic enumeration.

Derive from this class to define new enumerations.

#### Attributes

<b>COM- PUTE_FEATURE_STAGE</b>	compute_feature_stage
<b>COM- PUTE_SHAP_VALUES_STAGE</b>	compute_shap_value_stage
<b>DONE</b>	done
<b>PRE- DICT_STAGE</b>	predict_stage
<b>PREPRO- CESS- ING_STAGE</b>	preprocessing_stage

#### Methods

<i>name</i>	The name of the Enum member.
<i>value</i>	The value of the Enum member.

**name** (*self*)  
The name of the Enum member.

**value** (*self*)  
The value of the Enum member.

## Package Contents

### Functions

<i>explain_predictions</i>	Creates a report summarizing the top contributing features for each data point in the input features.
<i>explain_predictions_best_worst</i>	Creates a report summarizing the top contributing features for the best and worst points in the dataset as measured by error to true labels.



## Contents

```
evalml.model_understanding.prediction_explanations.explain_predictions(pipeline,
                                                                    in-
                                                                    put_features,
                                                                    y,
                                                                    in-
                                                                    dices_to_explain,
                                                                    top_k_features=3,
                                                                    in-
                                                                    clude_shap_values=False,
                                                                    in-
                                                                    clude_expected_value=False,
                                                                    out-
                                                                    put_format='text')
```

Creates a report summarizing the top contributing features for each data point in the input features.

XGBoost and Stacked Ensemble models, as well as CatBoost multiclass classifiers, are not currently supported.

### Parameters

- **pipeline** (*PipelineBase*) – Fitted pipeline whose predictions we want to explain with SHAP.
- **input\_features** (*pd.DataFrame*) – Dataframe of input data to evaluate the pipeline on.
- **y** (*pd.Series*) – Labels for the input data.
- **indices\_to\_explain** (*list(int)*) – List of integer indices to explain.
- **top\_k\_features** (*int*) – How many of the highest/lowest contributing feature to include in the table for each data point. Default is 3.
- **include\_shap\_values** (*bool*) – Whether SHAP values should be included in the table. Default is False.
- **include\_expected\_value** (*bool*) – Whether the expected value should be included in the table. Default is False.
- **output\_format** (*str*) – Either “text”, “dict”, or “dataframe”. Default is “text”.

### Returns

**str, dict, or pd.DataFrame** - A report explaining the top contributing features to each prediction for each row of the input data. The report will include the feature names, prediction contribution, and SHAP Value (optional).

### Raises

- **ValueError** – if input\_features is empty.
- **ValueError** – if an output\_format outside of “text”, “dict” or “dataframe is provided.
- **ValueError** – if the requested index falls outside the input\_feature’s boundaries.

`evalml.model_understanding.prediction_explanations.explain_predictions_best_worst` (*pipeline*, *input\_features*, *y\_true*, *num\_to\_explain*, *top\_k\_features*, *include\_shap\_values*, *metric=None*, *output\_format="text"*, *callback=None*)

Creates a report summarizing the top contributing features for the best and worst points in the dataset as measured by error to true labels.

XGBoost and Stacked Ensemble models, as well as CatBoost multiclass classifiers, are not currently supported.

#### Parameters

- **pipeline** (*PipelineBase*) – Fitted pipeline whose predictions we want to explain with SHAP.
- **input\_features** (*pd.DataFrame*) – Input data to evaluate the pipeline on.
- **y\_true** (*pd.Series*) – True labels for the input data.
- **num\_to\_explain** (*int*) – How many of the best, worst, random data points to explain.
- **top\_k\_features** (*int*) – How many of the highest/lowest contributing feature to include in the table for each data point.
- **include\_shap\_values** (*bool*) – Whether SHAP values should be included in the table. Default is False.
- **metric** (*callable*) – The metric used to identify the best and worst points in the dataset. Function must accept the true labels and predicted value or probabilities as the only arguments and lower values must be better. By default, this will be the absolute error for regression problems and cross entropy loss for classification problems.
- **output\_format** (*str*) – Either “text” or “dict”. Default is “text”.
- **callback** (*callable*) – Function to be called with incremental updates. Has the following parameters: - *progress\_stage*: stage of computation - *time\_elapsed*: total time in seconds that has elapsed since start of call

#### Returns

**str, dict, or pd.DataFrame** - A report explaining the top contributing features for the best/worst predictions in the dataset. For each of the best/worst rows of *input\_features*, the predicted values, true labels, metric value, feature names, prediction contribution, and SHAP Value (optional) will be listed.

#### Raises

- **ValueError** – if *input\_features* does not have more than twice the requested features to explain.
- **ValueError** – if *y\_true* and *input\_features* have mismatched lengths.
- **ValueError** – if an *output\_format* outside of “text”, “dict” or “dataframe” is provided.

## Submodules

### force\_plots

## Module Contents

## Functions

<code>force_plot</code>	Function to generate the data required to build a force plot.
<code>graph_force_plot</code>	Function to generate force plots for the desired rows of the training data.

## Contents

`evalml.model_understanding.force_plots.force_plot` (*pipeline*, *rows\_to\_explain*, *training\_data*, *y*)

Function to generate the data required to build a force plot.

### Parameters

- **pipeline** (*PipelineBase*) – The pipeline to generate the force plot for.
- **rows\_to\_explain** (*list(int)*) – A list of the indices of the *training\_data* to explain.
- **training\_data** (*pandas.DataFrame*) – The data used to train the pipeline.
- **y** (*pandas.Series*) – The target data.

### Returns

**list of dictionaries where each dict contains force plot data. Each dictionary entry** represents the explanations for a single row.

#### For single row binary force plots:

```
[{'malignant': {'expected_value': 0.37, 'feature_names': ['worst concave points',
                                                         'worst perimeter', 'worst radius'], 'shap_values': [0.09, 0.09, 0.08], 'plot': AdditiveForceVisualizer}]
```

#### For two row binary force plots:

```
[{'malignant': {'expected_value': 0.37, 'feature_names': ['worst concave points',
                                                         'worst perimeter', 'worst radius'], 'shap_values': [0.09, 0.09, 0.08], 'plot': AdditiveForceVisualizer},
 {'malignant': {'expected_value': 0.29, 'feature_names': ['worst concave points',
                                                         'worst perimeter', 'worst radius'], 'shap_values': [0.05, 0.03, 0.02], 'plot': AdditiveForceVisualizer}]
```

**Return type** `list(dict())`

### Raises

- **TypeError** – if *rows\_to\_explain* is not a list.
- **TypeError** – if all values in *rows\_to\_explain* aren't integers.

`evalml.model_understanding.force_plots.graph_force_plot` (*pipeline*, *rows\_to\_explain*, *training\_data*, *y*, *matplotlib=False*)

Function to generate force plots for the desired rows of the training data.

#### Parameters

- **pipeline** (*PipelineBase*) – The pipeline to generate the force plot for.
- **rows\_to\_explain** (*list(int)*) – A list of the indices indicating which of the rows of the *training\_data* to explain.
- **training\_data** (*pandas.DataFrame*) – The data used to train the pipeline.
- **y** (*pandas.Series*) – The target data for the pipeline.
- **matplotlib** (*bool*) – flag to display the force plot using matplotlib (outside of jupyter) Defaults to False.

#### Returns

The same as `force_plot()`, but with an additional key in each dictionary for the plot.

**Return type** `list(dict(shap.AdditiveForceVisualizer))`

## graphs

### Module Contents

#### Functions

<code>binary_objective_vs_threshold</code>	Computes objective score as a function of potential binary classification
<code>confusion_matrix</code>	Confusion matrix for binary and multiclass classification.
<code>decision_tree_data_from_estimator</code>	Return data for a fitted tree in a restructured format
<code>decision_tree_data_from_pipeline</code>	Return data for a fitted pipeline with in a restructured format
<code>get_linear_coefficients</code>	Returns a dataframe showing the features with the greatest predictive power for a linear model.
<code>get_prediction_vs_actual_data</code>	Combines <i>y_true</i> and <i>y_pred</i> into a single dataframe and adds a column for outliers. Used in <code>graph_prediction_vs_actual()</code> .
<code>get_prediction_vs_actual_over_time_data</code>	Get the data needed for the <code>prediction_vs_actual_over_time</code> plot.
<code>graph_binary_objective_vs_threshold</code>	Generates a plot graphing objective score vs. decision thresholds for a fitted binary classification pipeline.
<code>graph_confusion_matrix</code>	Generate and display a confusion matrix plot.
<code>graph_partial_dependence</code>	Create an one-way or two-way partial dependence plot. Passing a single integer or
<code>graph_permutation_importance</code>	Generate a bar graph of the pipeline's permutation importance.
<code>graph_precision_recall_curve</code>	Generate and display a precision-recall plot.
<code>graph_prediction_vs_actual</code>	Generate a scatter plot comparing the true and predicted values. Used for regression plotting

continues on next page

Table 186 – continued from previous page

<code>graph_prediction_vs_actual_over_time</code>	Plot the target values and predictions against time on the x-axis.
<code>graph_roc_curve</code>	Generate and display a Receiver Operating Characteristic (ROC) plot for binary and multiclass classification problems.
<code>graph_t_sne</code>	Plot high dimensional data into lower dimensional space using t-SNE .
<code>normalize_confusion_matrix</code>	Normalizes a confusion matrix.
<code>partial_dependence</code>	Calculates one or two-way partial dependence. If a single integer or
<code>precision_recall_curve</code>	Given labels and binary classifier predicted probabilities, compute and return the data representing a precision-recall curve.
<code>roc_curve</code>	Given labels and classifier predicted probabilities, compute and return the data representing a Receiver Operating Characteristic (ROC) curve. Works with binary or multiclass problems.
<code>t_sne</code>	Get the transformed output after fitting X to the embedded space using t-SNE.
<code>visualize_decision_tree</code>	Generate an image visualizing the decision tree

## Contents

`evalml.model_understanding.graphs.binary_objective_vs_threshold`(*pipeline*, *X*,  
*y*, *objective*,  
*steps=100*)

**Computes objective score as a function of potential binary classification** decision thresholds for a fitted binary classification pipeline.

### Parameters

- **pipeline** (*BinaryClassificationPipeline obj*) – Fitted binary classification pipeline
- **X** (*pd.DataFrame*) – The input data used to compute objective score
- **y** (*pd.Series*) – The target labels
- **objective** (*ObjectiveBase obj, str*) – Objective used to score
- **steps** (*int*) – Number of intervals to divide and calculate objective score at

**Returns** DataFrame with thresholds and the corresponding objective score calculated at each threshold

**Return type** `pd.DataFrame`

`evalml.model_understanding.graphs.confusion_matrix`(*y\_true*, *y\_predicted*, *normalize\_method='true'*)

Confusion matrix for binary and multiclass classification.

### Parameters

- **y\_true** (*pd.Series or np.ndarray*) – True binary labels.
- **y\_pred** (*pd.Series or np.ndarray*) – Predictions from a binary classifier.

- **normalize\_method** (`{'true', 'pred', 'all', None}`) – Normalization method to use, if not None. Supported options are: ‘true’ to normalize by row, ‘pred’ to normalize by column, or ‘all’ to normalize by all values. Defaults to ‘true’.

**Returns** Confusion matrix. The column header represents the predicted labels while row header represents the actual labels.

**Return type** `pd.DataFrame`

`evalml.model_understanding.graphs.decision_tree_data_from_estimator(estimator)`

Return data for a fitted tree in a restructured format

**Parameters** **estimator** (*ComponentBase*) – A fitted DecisionTree-based estimator.

**Returns** An OrderedDict of OrderedDicts describing a tree structure

**Return type** `OrderedDict`

`evalml.model_understanding.graphs.decision_tree_data_from_pipeline(pipeline_)`

Return data for a fitted pipeline with in a restructured format

**Parameters** **pipeline** (*PipelineBase*) – A pipeline with a DecisionTree-based estimator.

**Returns** An OrderedDict of OrderedDicts describing a tree structure

**Return type** `OrderedDict`

`evalml.model_understanding.graphs.get_linear_coefficients(estimator, features=None)`

Returns a dataframe showing the features with the greatest predictive power for a linear model.

**Parameters**

- **estimator** (*Estimator*) – Fitted linear model family estimator.
- **features** (*list[str]*) – List of feature names associated with the underlying data.

**Returns** Displaying the features by importance.

**Return type** `pd.DataFrame`

`evalml.model_understanding.graphs.get_prediction_vs_actual_data(y_true, y_pred, outlier_threshold=None)`

Combines `y_true` and `y_pred` into a single dataframe and adds a column for outliers. Used in `graph_prediction_vs_actual()`.

**Parameters**

- **y\_true** (*pd.Series, or np.ndarray*) – The real target values of the data
- **y\_pred** (*pd.Series, or np.ndarray*) – The predicted values outputted by the regression model.
- **outlier\_threshold** (*int, float*) – A positive threshold for what is considered an outlier value. This value is compared to the absolute difference between each value of `y_true` and `y_pred`. Values within this threshold will be blue, otherwise they will be yellow. Defaults to None

**Returns**

- *prediction*: Predicted values from regression model.
- *actual*: Real target values.
- *outlier*: Colors indicating which values are in the threshold for what is considered an outlier value.

**Return type** `pd.DataFrame` with the following columns

```
evalml.model_understanding.graphs.get_prediction_vs_actual_over_time_data(pipeline,
                                                                           X,
                                                                           y,
                                                                           dates)
```

Get the data needed for the `prediction_vs_actual_over_time` plot.

#### Parameters

- **pipeline** (*TimeSeriesRegressionPipeline*) – Fitted time series regression pipeline.
- **x** (*pd.DataFrame*) – Features used to generate new predictions.
- **y** (*pd.Series*) – Target values to compare predictions against.
- **dates** (*pd.Series*) – Dates corresponding to target values and predictions.

**Returns** `pd.DataFrame`

```
evalml.model_understanding.graphs.graph_binary_objective_vs_threshold(pipeline,
                                                                      X, y,
                                                                      ob-
                                                                      jec-
                                                                      tive,
                                                                      steps=100)
```

Generates a plot graphing objective score vs. decision thresholds for a fitted binary classification pipeline.

#### Parameters

- **pipeline** (*PipelineBase* or *subclass*) – Fitted pipeline
- **x** (*pd.DataFrame*) – The input data used to score and compute scores
- **y** (*pd.Series*) – The target labels
- **objective** (*ObjectiveBase obj*, *str*) – Objective used to score, shown on the y-axis of the graph
- **steps** (*int*) – Number of intervals to divide and calculate objective score at

**Returns** `plotly.Figure` representing the objective score vs. threshold graph generated

```
evalml.model_understanding.graphs.graph_confusion_matrix(y_true, y_pred, nor-
                                                         malize_method='true',
                                                         title_addition=None)
```

Generate and display a confusion matrix plot.

If `normalize_method` is set, hover text will show raw count, otherwise hover text will show count normalized with method 'true'.

#### Parameters

- **y\_true** (*pd.Series* or *np.ndarray*) – True binary labels.
- **y\_pred** (*pd.Series* or *np.ndarray*) – Predictions from a binary classifier.
- **normalize\_method** (*{'true', 'pred', 'all', None}*) – Normalization method to use, if not None. Supported options are: 'true' to normalize by row, 'pred' to normalize by column, or 'all' to normalize by all values. Defaults to 'true'.
- **title\_addition** (*str* or *None*) – if not None, append to plot title. Defaults to None.

**Returns** `plotly.Figure` representing the confusion matrix plot generated

```
evalml.model_understanding.graphs.graph_partial_dependence(pipeline, X, features,  
                                                           class_label=None,  
                                                           grid_resolution=100,  
                                                           kind='average')
```

Create an one-way or two-way partial dependence plot. Passing a single integer or string as features will create a one-way partial dependence plot with the feature values plotted against the partial dependence. Passing features a tuple of int/strings will create a two-way partial dependence plot with a contour of feature[0] in the y-axis, feature[1] in the x-axis and the partial dependence in the z-axis.

#### Parameters

- **pipeline** (*PipelineBase* or *subclass*) – Fitted pipeline
- **X** (*pd.DataFrame*, *np.ndarray*) – The input data used to generate a grid of values for feature where partial dependence will be calculated at
- **features** (*int*, *string*, *tuple[int or string]*) – The target feature for which to create the partial dependence plot for. If features is an int, it must be the index of the feature to use. If features is a string, it must be a valid column name in X. If features is a tuple of strings, it must contain valid column int/names in X.
- **class\_label** (*string*, *optional*) – Name of class to plot for multiclass problems. If None, will plot the partial dependence for each class. This argument does not change behavior for regression or binary classification pipelines. For binary classification, the partial dependence for the positive label will always be displayed. Defaults to None.
- **grid\_resolution** (*int*) – Number of samples of feature(s) for partial dependence plot
- **{ 'average' }** (*kind*) – Type of partial dependence to plot. ‘average’ creates a regular partial dependence (PD) graph, ‘individual’ creates an individual conditional expectation (ICE) plot, and ‘both’ creates a single-figure PD and ICE plot. ICE plots can only be shown for one-way partial dependence plots.
- **'individual'** – Type of partial dependence to plot. ‘average’ creates a regular partial dependence (PD) graph, ‘individual’ creates an individual conditional expectation (ICE) plot, and ‘both’ creates a single-figure PD and ICE plot. ICE plots can only be shown for one-way partial dependence plots.
- **'both'** – Type of partial dependence to plot. ‘average’ creates a regular partial dependence (PD) graph, ‘individual’ creates an individual conditional expectation (ICE) plot, and ‘both’ creates a single-figure PD and ICE plot. ICE plots can only be shown for one-way partial dependence plots.

**Returns** figure object containing the partial dependence data for plotting

**Return type** `plotly.graph_objects.Figure`

#### Raises

- **PartialDependenceError** – if a graph is requested for a class name that isn’t present in the pipeline.
- **PartialDependenceError** – if an ICE plot is requested for a two-way partial dependence.

```
evalml.model_understanding.graphs.graph_permutation_importance(pipeline, X,  
                                                                y, objective,  
                                                                importance,  
                                                                importance_threshold=0)
```

Generate a bar graph of the pipeline’s permutation importance.

#### Parameters



- **pipeline** (*PipelineBase* or *subclass*) – Fitted pipeline
- **x** (*pd.DataFrame*) – The input data used to score and compute permutation importance
- **y** (*pd.Series*) – The target data
- **objective** (*str*, *ObjectiveBase*) – Objective to score on
- **importance\_threshold** (*float*, *optional*) – If provided, graph features with a permutation importance whose absolute value is larger than `importance_threshold`. Defaults to zero.

**Returns** `plotly.Figure`, a bar graph showing features and their respective permutation importance.

```
evalml.model_understanding.graphs.graph_precision_recall_curve(y_true,
                                                                y_pred_proba,
                                                                title_addition=None)
```

Generate and display a precision-recall plot.

#### Parameters

- **y\_true** (*pd.Series* or *np.ndarray*) – True binary labels.
- **y\_pred\_proba** (*pd.Series* or *np.ndarray*) – Predictions from a binary classifier, before thresholding has been applied. Note this should be the predicted probability for the “true” label.
- **title\_addition** (*str* or *None*) – If not *None*, append to plot title. Default *None*.

**Returns** `plotly.Figure` representing the precision-recall plot generated

```
evalml.model_understanding.graphs.graph_prediction_vs_actual(y_true, y_pred, outlier_threshold=None)
```

Generate a scatter plot comparing the true and predicted values. Used for regression plotting

#### Parameters

- **y\_true** (*pd.Series*) – The real target values of the data
- **y\_pred** (*pd.Series*) – The predicted values outputted by the regression model.
- **outlier\_threshold** (*int*, *float*) – A positive threshold for what is considered an outlier value. This value is compared to the absolute difference between each value of `y_true` and `y_pred`. Values within this threshold will be blue, otherwise they will be yellow. Defaults to *None*

**Returns** `plotly.Figure` representing the predicted vs. actual values graph

```
evalml.model_understanding.graphs.graph_prediction_vs_actual_over_time(pipeline,
                                                                         X, y,
                                                                         dates)
```

Plot the target values and predictions against time on the x-axis.

#### Parameters

- **pipeline** (*TimeSeriesRegressionPipeline*) – Fitted time series regression pipeline.
- **x** (*pd.DataFrame*) – Features used to generate new predictions.
- **y** (*pd.Series*) – Target values to compare predictions against.
- **dates** (*pd.Series*) – Dates corresponding to target values and predictions.

**Returns** Showing the prediction vs actual over time.

**Return type** `plotly.Figure`

```
evalml.model_understanding.graphs.graph_roc_curve(y_true, y_pred_proba, cus-  
tom_class_names=None, ti-  
tle_addition=None)
```

Generate and display a Receiver Operating Characteristic (ROC) plot for binary and multiclass classification problems.

**Parameters**

- **y\_true** (`pd.Series` or `np.ndarray`) – True labels.
- **y\_pred\_proba** (`pd.Series` or `np.ndarray`) – Predictions from a classifier, before thresholding has been applied. Note this should be a one dimensional array with the predicted probability for the “true” label in the binary case.
- **custom\_class\_labels** (`list` or `None`) – If not `None`, custom labels for classes. Default `None`.
- **title\_addition** (`str` or `None`) – if not `None`, append to plot title. Default `None`.

**Returns** `plotly.Figure` representing the ROC plot generated

```
evalml.model_understanding.graphs.graph_t_sne(X, n_components=2, perplexity=30.0,  
learning_rate=200.0, metric='euclidean',  
marker_line_width=2, marker_size=7,  
**kwargs)
```

Plot high dimensional data into lower dimensional space using t-SNE .

**Parameters**

- **X** (`np.ndarray`, `pd.DataFrame`) – Data to be transformed. Must be numeric.
- **n\_components** (`int`, *optional*) – Dimension of the embedded space.
- **perplexity** (`float`, *optional*) – Related to the number of nearest neighbors that is used in other manifold learning
- **Larger datasets usually require a larger perplexity. Consider selecting a value between 5 and 50.** (*algorithms.*)  
–
- **learning\_rate** (`float`, *optional*) – Usually in the range `[10.0, 1000.0]`. If the cost function gets stuck in a bad
- **minimum** (*local*) –
- **the learning rate may help.** (*increasing*) –
- **metric** (`str`, *optional*) – The metric to use when calculating distance between instances in a feature array.
- **marker\_line\_width** (`int`, *optional*) – Determines the line width of the marker boundary.
- **marker\_size** (`int`, *optional*) – Determines the size of the marker.

**Returns** `plotly.Figure` representing the transformed data

```
evalml.model_understanding.graphs.normalize_confusion_matrix(conf_mat, normal-  
ize_method='true')
```

Normalizes a confusion matrix.

**Parameters**

- **conf\_mat** (`pd.DataFrame` or `np.ndarray`) – Confusion matrix to normalize.

- **normalize\_method** (`{'true', 'pred', 'all'}`) – Normalization method. Supported options are: 'true' to normalize by row, 'pred' to normalize by column, or 'all' to normalize by all values. Defaults to 'true'.

**Returns** normalized version of the input confusion matrix. The column header represents the predicted labels while row header represents the actual labels.

**Return type** `pd.DataFrame`

```
evalml.model_understanding.graphs.partial_dependence(pipeline, X, features, per-
centiles=(0.05, 0.95),
grid_resolution=100,
kind='average')
```

Calculates one or two-way partial dependence. If a single integer or string is given for features, one-way partial dependence is calculated. If a tuple of two integers or strings is given, two-way partial dependence is calculated with the first feature in the y-axis and second feature in the x-axis.

#### Parameters

- **pipeline** (*PipelineBase or subclass*) – Fitted pipeline
- **X** (*pd.DataFrame, np.ndarray*) – The input data used to generate a grid of values for feature where partial dependence will be calculated at
- **features** (*int, string, tuple[int or string]*) – The target feature for which to create the partial dependence plot for. If features is an int, it must be the index of the feature to use. If features is a string, it must be a valid column name in X. If features is a tuple of int/strings, it must contain valid column integers/names in X.
- **percentiles** (*tuple[float]*) – The lower and upper percentile used to create the extreme values for the grid. Must be in [0, 1]. Defaults to (0.05, 0.95).
- **grid\_resolution** (*int*) – Number of samples of feature(s) for partial dependence plot. If this value is less than the maximum number of categories present in categorical data within X, it will be set to the max number of categories + 1. Defaults to 100.
- **{'average'}** (*kind*) – The type of predictions to return. 'individual' will return the predictions for all of the points in the grid for each sample in X. 'average' will return the predictions for all of the points in the grid but averaged over all of the samples in X.
- **'individual'** – The type of predictions to return. 'individual' will return the predictions for all of the points in the grid for each sample in X. 'average' will return the predictions for all of the points in the grid but averaged over all of the samples in X.
- **'both'** – The type of predictions to return. 'individual' will return the predictions for all of the points in the grid for each sample in X. 'average' will return the predictions for all of the points in the grid but averaged over all of the samples in X.

#### Returns

When *kind='average'*: DataFrame with averaged predictions for all points in the grid averaged over all samples of X and the values used to calculate those predictions.

When *kind='individual'*: DataFrame with individual predictions for all points in the grid for each sample of X and the values used to calculate those predictions. If a two-way partial dependence is calculated, then the result is a list of DataFrames with each DataFrame representing one sample's predictions.

When *kind='both'*: A tuple consisting of the averaged predictions (in a DataFrame) over all samples of X and the individual predictions (in a list of DataFrames) for each sample of X.

In the one-way case: The dataframe will contain two columns, “feature\_values” (grid points at which the partial dependence was calculated) and “partial\_dependence” (the partial dependence at that feature value). For classification problems, there will be a third column called “class\_label” (the class label for which the partial dependence was calculated). For binary classification, the partial dependence is only calculated for the “positive” class.

In the two-way case: The data frame will contain grid\_resolution number of columns and rows where the index and column headers are the sampled values of the first and second features, respectively, used to make the partial dependence contour. The values of the data frame contain the partial dependence data for each feature value pair.

**Return type** `pd.DataFrame`, `list(pd.DataFrame)`, or `tuple(pd.DataFrame, list(pd.DataFrame))`

#### Raises

- **PartialDependenceError** – if the user provides a tuple of not exactly two features.
- **PartialDependenceError** – if the provided pipeline isn’t fitted.
- **PartialDependenceError** – if the provided pipeline is a Baseline pipeline.
- **PartialDependenceError** – if any of the features passed in are completely NaN
- **PartialDependenceError** – if any of the features are low-variance. Defined as having one value occurring more than the upper percentile passed by the user. By default 95%.

`evalml.model_understanding.graphs.precision_recall_curve(y_true, y_pred_proba, pos_label_idx=-1)`

Given labels and binary classifier predicted probabilities, compute and return the data representing a precision-recall curve.

#### Parameters

- **y\_true** (`pd.Series` or `np.ndarray`) – True binary labels.
- **y\_pred\_proba** (`pd.Series` or `np.ndarray`) – Predictions from a binary classifier, before thresholding has been applied. Note this should be the predicted probability for the “true” label.
- **pos\_label\_idx** (`int`) – the column index corresponding to the positive class. If predicted probabilities are two-dimensional, this will be used to access the probabilities for the positive class.

#### Returns

Dictionary containing metrics used to generate a precision-recall plot, with the following keys:

- *precision*: Precision values.
- *recall*: Recall values.
- *thresholds*: Threshold values used to produce the precision and recall.
- *auc\_score*: The area under the ROC curve.

#### Return type

`list`

`evalml.model_understanding.graphs.roc_curve(y_true, y_pred_proba)`

Given labels and classifier predicted probabilities, compute and return the data representing a Receiver Operating Characteristic (ROC) curve. Works with binary or multiclass problems.

#### Parameters

- **y\_true** (`pd.Series` or `np.ndarray`) – True labels.

- **y\_pred\_proba** (*pd.Series* or *np.ndarray*) – Predictions from a classifier, before thresholding has been applied.

### Returns

A list of dictionaries (with one for each class) is returned. Binary classification problems return a list with one di

Each dictionary contains metrics used to generate an ROC plot with the following keys:

- *fpr\_rate*: False positive rate.
- *tpr\_rate*: True positive rate.
- *threshold*: Threshold values used to produce each pair of true/false positive rates.
- *auc\_score*: The area under the ROC curve.

**Return type** list(dict)

```
evalml.model_understanding.graphs.t_sne(X, n_components=2, perplexity=30.0, learning_rate=200.0, metric='euclidean', **kwargs)
```

Get the transformed output after fitting X to the embedded space using t-SNE.

**Arguments:** X (*np.ndarray*, *pd.DataFrame*): Data to be transformed. Must be numeric. n\_components (*int*, optional): Dimension of the embedded space. perplexity (*float*, optional): Related to the number of nearest neighbors that is used in other manifold learning algorithms. Larger datasets usually require a larger perplexity. Consider selecting a value between 5 and 50. learning\_rate (*float*, optional): Usually in the range [10.0, 1000.0]. If the cost function gets stuck in a bad local minimum, increasing the learning rate may help. metric (*str*, optional): The metric to use when calculating distance between instances in a feature array.

**Returns** *np.ndarray* (n\_samples, n\_components)

```
evalml.model_understanding.graphs.visualize_decision_tree(estimator,
                                                           max_depth=None,
                                                           rotate=False,
                                                           filled=False,
                                                           filepath=None)
```

Generate an image visualizing the decision tree

### Parameters

- **estimator** (*ComponentBase*) – A fitted DecisionTree-based estimator.
- **max\_depth** (*int*, optional) – The depth to which the tree should be displayed. If set to None (as by default),
- **is fully generated.** (*tree*) –
- **rotate** (*bool*, optional) – Orient tree left to right rather than top-down.
- **filled** (*bool*, optional) – Paint nodes to indicate majority class for classification, extremity of values for
- **regression** –
- **purity of node for multi-output.** (*or*) –
- **filepath** (*str*, optional) – Path to where the graph should be saved. If set to None (as by default), the graph
- **not be saved.** (*will*) –

**Returns** DOT object that can be directly displayed in Jupyter notebooks.

**Return type** graphviz.Source

permutation\_importance

Module Contents

Functions

<code>calculate_permutation_importance</code>	Calculates permutation importance for features.
<code>calculate_permutation_importance_one_column</code>	Calculates permutation importance for one column in the original dataframe.

Contents

`evalml.model_understanding.permutation_importance.calculate_permutation_importance` (*pipeline, X, y, objective, n\_repeats=None, n\_jobs=None, random\_seed=None*)

Calculates permutation importance for features.

Parameters

- **pipeline** (*PipelineBase or subclass*) – Fitted pipeline.
- **X** (*pd.DataFrame*) – The input data used to score and compute permutation importance.
- **y** (*pd.Series*) – The target data.
- **objective** (*str, ObjectiveBase*) – Objective to score on.
- **n\_repeats** (*int*) – Number of times to permute a feature. Defaults to 5.
- **n\_jobs** (*int or None*) – Non-negative integer describing level of parallelism used for pipelines. None and 1 are equivalent. If set to -1, all CPUs are used. For n\_jobs below -1, (n\_cpus + 1 + n\_jobs) are used. Defaults to None.
- **random\_seed** (*int*) – Seed for the random number generator. Defaults to 0.

**Returns** Mean feature importance scores over a number of shuffles.

**Return type** pd.DataFrame

---

```
evalml.model_understanding.permutation_importance.calculate_permutation_importance_one_col
```

Calculates permutation importance for one column in the original dataframe.

#### Parameters

- **pipeline** (*PipelineBase* or *subclass*) – Fitted pipeline.
- **x** (*pd.DataFrame*) – The input data used to score and compute permutation importance.
- **y** (*pd.Series*) – The target data.
- **col\_name** (*str*, *int*) – The column in X to calculate permutation importance for.
- **objective** (*str*, *ObjectiveBase*) – Objective to score on.
- **n\_repeats** (*int*) – Number of times to permute a feature. Defaults to 5.
- **fast** (*bool*) – Whether to use the fast method of calculating the permutation importance or not. Defaults to True.
- **precomputed\_features** (*pd.DataFrame*) – Precomputed features necessary to calculate permutation importance using the fast method. Defaults to None.
- **random\_seed** (*int*) – Seed for the random number generator. Defaults to 0.

**Returns** Mean feature importance scores over a number of shuffles.

**Return type** float

## Package Contents

### Functions

<i>binary_objective_vs_threshold</i>	Computes objective score as a function of potential binary classification
<i>calculate_permutation_importance</i>	Calculates permutation importance for features.
<i>calculate_permutation_importance_one_col</i>	Calculates permutation importance for one column in the original dataframe.
<i>confusion_matrix</i>	Confusion matrix for binary and multiclass classification.
<i>explain_predictions</i>	Creates a report summarizing the top contributing features for each data point in the input features.

continues on next page

Table 188 – continued from previous page

<code>explain_predictions_best_worst</code>	Creates a report summarizing the top contributing features for the best and worst points in the dataset as measured by error to true labels.
<code>get_linear_coefficients</code>	Returns a dataframe showing the features with the greatest predictive power for a linear model.
<code>get_prediction_vs_actual_data</code>	Combines <code>y_true</code> and <code>y_pred</code> into a single dataframe and adds a column for outliers. Used in <code>graph_prediction_vs_actual()</code> .
<code>get_prediction_vs_actual_over_time_data</code>	Get the data needed for the <code>prediction_vs_actual_over_time</code> plot.
<code>graph_binary_objective_vs_threshold</code>	Generates a plot graphing objective score vs. decision thresholds for a fitted binary classification pipeline.
<code>graph_confusion_matrix</code>	Generate and display a confusion matrix plot.
<code>graph_partial_dependence</code>	Create an one-way or two-way partial dependence plot. Passing a single integer or
<code>graph_permutation_importance</code>	Generate a bar graph of the pipeline’s permutation importance.
<code>graph_precision_recall_curve</code>	Generate and display a precision-recall plot.
<code>graph_prediction_vs_actual</code>	Generate a scatter plot comparing the true and predicted values. Used for regression plotting
<code>graph_prediction_vs_actual_over_time</code>	Plot the target values and predictions against time on the x-axis.
<code>graph_roc_curve</code>	Generate and display a Receiver Operating Characteristic (ROC) plot for binary and multiclass classification problems.
<code>graph_t_sne</code>	Plot high dimensional data into lower dimensional space using t-SNE .
<code>normalize_confusion_matrix</code>	Normalizes a confusion matrix.
<code>partial_dependence</code>	Calculates one or two-way partial dependence. If a single integer or
<code>precision_recall_curve</code>	Given labels and binary classifier predicted probabilities, compute and return the data representing a precision-recall curve.
<code>roc_curve</code>	Given labels and classifier predicted probabilities, compute and return the data representing a Receiver Operating Characteristic (ROC) curve. Works with binary or multiclass problems.
<code>t_sne</code>	Get the transformed output after fitting X to the embedded space using t-SNE.

## Contents

`evalml.model_understanding.binary_objective_vs_threshold`(*pipeline*, *X*, *y*, *objective*, *steps=100*)

**Computes objective score as a function of potential binary classification** decision thresholds for a fitted binary classification pipeline.

### Parameters

- **pipeline** (*BinaryClassificationPipeline obj*) – Fitted binary classification pipeline



- **x** (*pd.DataFrame*) – The input data used to compute objective score
- **y** (*pd.Series*) – The target labels
- **objective** (*ObjectiveBase obj, str*) – Objective used to score
- **steps** (*int*) – Number of intervals to divide and calculate objective score at

**Returns** DataFrame with thresholds and the corresponding objective score calculated at each threshold

**Return type** *pd.DataFrame*

```
evalml.model_understanding.calculate_permutation_importance(pipeline, X, y, objective, n_repeats=5, n_jobs=None, random_seed=0)
```

Calculates permutation importance for features.

#### Parameters

- **pipeline** (*PipelineBase or subclass*) – Fitted pipeline.
- **x** (*pd.DataFrame*) – The input data used to score and compute permutation importance.
- **y** (*pd.Series*) – The target data.
- **objective** (*str, ObjectiveBase*) – Objective to score on.
- **n\_repeats** (*int*) – Number of times to permute a feature. Defaults to 5.
- **n\_jobs** (*int or None*) – Non-negative integer describing level of parallelism used for pipelines. None and 1 are equivalent. If set to -1, all CPUs are used. For *n\_jobs* below -1, (*n\_cpus* + 1 + *n\_jobs*) are used. Defaults to None.
- **random\_seed** (*int*) – Seed for the random number generator. Defaults to 0.

**Returns** Mean feature importance scores over a number of shuffles.

**Return type** *pd.DataFrame*

```
evalml.model_understanding.calculate_permutation_importance_one_column(pipeline, X, y, col_name, objective, n_repeats=5, fast=True, precomputed_features=None, random_seed=0)
```

Calculates permutation importance for one column in the original dataframe.

#### Parameters

- **pipeline** (*PipelineBase or subclass*) – Fitted pipeline.
- **x** (*pd.DataFrame*) – The input data used to score and compute permutation importance.
- **y** (*pd.Series*) – The target data.
- **col\_name** (*str, int*) – The column in X to calculate permutation importance for.

- **objective** (*str*, *ObjectiveBase*) – Objective to score on.
- **n\_repeats** (*int*) – Number of times to permute a feature. Defaults to 5.
- **fast** (*bool*) – Whether to use the fast method of calculating the permutation importance or not. Defaults to True.
- **precomputed\_features** (*pd.DataFrame*) – Precomputed features necessary to calculate permutation importance using the fast method. Defaults to None.
- **random\_seed** (*int*) – Seed for the random number generator. Defaults to 0.

**Returns** Mean feature importance scores over a number of shuffles.

**Return type** float

```
evalml.model_understanding.confusion_matrix(y_true, y_predicted, normalize_method='true')
```

Confusion matrix for binary and multiclass classification.

#### Parameters

- **y\_true** (*pd.Series* or *np.ndarray*) – True binary labels.
- **y\_pred** (*pd.Series* or *np.ndarray*) – Predictions from a binary classifier.
- **normalize\_method** (*{'true', 'pred', 'all', None}*) – Normalization method to use, if not None. Supported options are: ‘true’ to normalize by row, ‘pred’ to normalize by column, or ‘all’ to normalize by all values. Defaults to ‘true’.

**Returns** Confusion matrix. The column header represents the predicted labels while row header represents the actual labels.

**Return type** *pd.DataFrame*

```
evalml.model_understanding.explain_predictions(pipeline, input_features, y, indices_to_explain, top_k_features=3, include_shap_values=False, include_expected_value=False, output_format='text')
```

Creates a report summarizing the top contributing features for each data point in the input features.

XGBoost and Stacked Ensemble models, as well as CatBoost multiclass classifiers, are not currently supported.

#### Parameters

- **pipeline** (*PipelineBase*) – Fitted pipeline whose predictions we want to explain with SHAP.
- **input\_features** (*pd.DataFrame*) – Dataframe of input data to evaluate the pipeline on.
- **y** (*pd.Series*) – Labels for the input data.
- **indices\_to\_explain** (*list(int)*) – List of integer indices to explain.
- **top\_k\_features** (*int*) – How many of the highest/lowest contributing feature to include in the table for each data point. Default is 3.
- **include\_shap\_values** (*bool*) – Whether SHAP values should be included in the table. Default is False.
- **include\_expected\_value** (*bool*) – Whether the expected value should be included in the table. Default is False.
- **output\_format** (*str*) – Either “text”, “dict”, or “dataframe”. Default is “text”.

**Returns**

**str, dict, or pd.DataFrame** - A report explaining the top contributing features to each prediction for each row of the report. The report will include the feature names, prediction contribution, and SHAP Value (optional).

**Raises**

- **ValueError** – if `input_features` is empty.
- **ValueError** – if an `output_format` outside of “text”, “dict” or “dataframe” is provided.
- **ValueError** – if the requested index falls outside the `input_feature`’s boundaries.

```
evalml.model_understanding.explain_predictions_best_worst(pipeline, input_features, y_true,
                                                         num_to_explain=5,
                                                         top_k_features=3, include_shap_values=False,
                                                         metric=None, output_format='text',
                                                         callback=None)
```

Creates a report summarizing the top contributing features for the best and worst points in the dataset as measured by error to true labels.

XGBoost and Stacked Ensemble models, as well as CatBoost multiclass classifiers, are not currently supported.

**Parameters**

- **pipeline** (*PipelineBase*) – Fitted pipeline whose predictions we want to explain with SHAP.
- **input\_features** (*pd.DataFrame*) – Input data to evaluate the pipeline on.
- **y\_true** (*pd.Series*) – True labels for the input data.
- **num\_to\_explain** (*int*) – How many of the best, worst, random data points to explain.
- **top\_k\_features** (*int*) – How many of the highest/lowest contributing feature to include in the table for each data point.
- **include\_shap\_values** (*bool*) – Whether SHAP values should be included in the table. Default is False.
- **metric** (*callable*) – The metric used to identify the best and worst points in the dataset. Function must accept the true labels and predicted value or probabilities as the only arguments and lower values must be better. By default, this will be the absolute error for regression problems and cross entropy loss for classification problems.
- **output\_format** (*str*) – Either “text” or “dict”. Default is “text”.
- **callback** (*callable*) – Function to be called with incremental updates. Has the following parameters: - `progress_stage`: stage of computation - `time_elapsed`: total time in seconds that has elapsed since start of call

**Returns**

**str, dict, or pd.DataFrame** - A report explaining the top contributing features for the best/worst predictions in the dataset. For each of the best/worst rows of `input_features`, the predicted values, true labels, metric value, feature names, prediction contribution, and SHAP Value (optional) will be listed.

**Raises**

- **ValueError** – if `input_features` does not have more than twice the requested features to explain.

- **ValueError** – if `y_true` and `input_features` have mismatched lengths.
- **ValueError** – if an `output_format` outside of “text”, “dict” or “dataframe” is provided.

`evalml.model_understanding.get_linear_coefficients` (*estimator, features=None*)

Returns a dataframe showing the features with the greatest predictive power for a linear model.

#### Parameters

- **estimator** (*Estimator*) – Fitted linear model family estimator.
- **features** (*list[str]*) – List of feature names associated with the underlying data.

**Returns** Displaying the features by importance.

**Return type** `pd.DataFrame`

`evalml.model_understanding.get_prediction_vs_actual_data` (*y\_true, y\_pred, outlier\_threshold=None*)

Combines `y_true` and `y_pred` into a single dataframe and adds a column for outliers. Used in `graph_prediction_vs_actual()`.

#### Parameters

- **y\_true** (*pd.Series, or np.ndarray*) – The real target values of the data
- **y\_pred** (*pd.Series, or np.ndarray*) – The predicted values outputted by the regression model.
- **outlier\_threshold** (*int, float*) – A positive threshold for what is considered an outlier value. This value is compared to the absolute difference between each value of `y_true` and `y_pred`. Values within this threshold will be blue, otherwise they will be yellow. Defaults to `None`

#### Returns

- *prediction*: Predicted values from regression model.
- *actual*: Real target values.
- *outlier*: Colors indicating which values are in the threshold for what is considered an outlier value.

**Return type** `pd.DataFrame` with the following columns

`evalml.model_understanding.get_prediction_vs_actual_over_time_data` (*pipeline, X, y, dates*)

Get the data needed for the `prediction_vs_actual_over_time` plot.

#### Parameters

- **pipeline** (*TimeSeriesRegressionPipeline*) – Fitted time series regression pipeline.
- **X** (*pd.DataFrame*) – Features used to generate new predictions.
- **y** (*pd.Series*) – Target values to compare predictions against.
- **dates** (*pd.Series*) – Dates corresponding to target values and predictions.

**Returns** `pd.DataFrame`

`evalml.model_understanding.graph_binary_objective_vs_threshold` (*pipeline, X, y, objective, steps=100*)

Generates a plot graphing objective score vs. decision thresholds for a fitted binary classification pipeline.

**Parameters**

- **pipeline** (*PipelineBase or subclass*) – Fitted pipeline
- **x** (*pd.DataFrame*) – The input data used to score and compute scores
- **y** (*pd.Series*) – The target labels
- **objective** (*ObjectiveBase obj, str*) – Objective used to score, shown on the y-axis of the graph
- **steps** (*int*) – Number of intervals to divide and calculate objective score at

**Returns** *plotly*.Figure representing the objective score vs. threshold graph generated

```
evalml.model_understanding.graph_confusion_matrix(y_true, y_pred, normalize_method='true', title_addition=None)
```

Generate and display a confusion matrix plot.

If *normalize\_method* is set, hover text will show raw count, otherwise hover text will show count normalized with method 'true'.

**Parameters**

- **y\_true** (*pd.Series or np.ndarray*) – True binary labels.
- **y\_pred** (*pd.Series or np.ndarray*) – Predictions from a binary classifier.
- **normalize\_method** (*{'true', 'pred', 'all', None}*) – Normalization method to use, if not None. Supported options are: 'true' to normalize by row, 'pred' to normalize by column, or 'all' to normalize by all values. Defaults to 'true'.
- **title\_addition** (*str or None*) – if not None, append to plot title. Defaults to None.

**Returns** *plotly*.Figure representing the confusion matrix plot generated

```
evalml.model_understanding.graph_partial_dependence(pipeline, X, features, class_label=None, grid_resolution=100, kind='average')
```

Create an one-way or two-way partial dependence plot. Passing a single integer or string as features will create a one-way partial dependence plot with the feature values plotted against the partial dependence. Passing features a tuple of int/strings will create a two-way partial dependence plot with a contour of feature[0] in the y-axis, feature[1] in the x-axis and the partial dependence in the z-axis.

**Parameters**

- **pipeline** (*PipelineBase or subclass*) – Fitted pipeline
- **x** (*pd.DataFrame, np.ndarray*) – The input data used to generate a grid of values for feature where partial dependence will be calculated at
- **features** (*int, string, tuple[int or string]*) – The target feature for which to create the partial dependence plot for. If features is an int, it must be the index of the feature to use. If features is a string, it must be a valid column name in X. If features is a tuple of strings, it must contain valid column int/names in X.
- **class\_label** (*string, optional*) – Name of class to plot for multiclass problems. If None, will plot the partial dependence for each class. This argument does not change behavior for regression or binary classification pipelines. For binary classification, the partial dependence for the positive label will always be displayed. Defaults to None.
- **grid\_resolution** (*int*) – Number of samples of feature(s) for partial dependence plot

- **'average'** (*kind*) – Type of partial dependence to plot. ‘average’ creates a regular partial dependence (PD) graph, ‘individual’ creates an individual conditional expectation (ICE) plot, and ‘both’ creates a single-figure PD and ICE plot. ICE plots can only be shown for one-way partial dependence plots.
- **'individual'** – Type of partial dependence to plot. ‘average’ creates a regular partial dependence (PD) graph, ‘individual’ creates an individual conditional expectation (ICE) plot, and ‘both’ creates a single-figure PD and ICE plot. ICE plots can only be shown for one-way partial dependence plots.
- **'both'** } – Type of partial dependence to plot. ‘average’ creates a regular partial dependence (PD) graph, ‘individual’ creates an individual conditional expectation (ICE) plot, and ‘both’ creates a single-figure PD and ICE plot. ICE plots can only be shown for one-way partial dependence plots.

**Returns** figure object containing the partial dependence data for plotting

**Return type** plotly.graph\_objects.Figure

**Raises**

- **PartialDependenceError** – if a graph is requested for a class name that isn’t present in the pipeline.
- **PartialDependenceError** – if an ICE plot is requested for a two-way partial dependence.

`evalml.model_understanding.graph_permutation_importance` (*pipeline*, *X*, *y*, *objective*, *importance\_threshold=0*)

Generate a bar graph of the pipeline’s permutation importance.

**Parameters**

- **pipeline** (*PipelineBase* or *subclass*) – Fitted pipeline
- **X** (*pd.DataFrame*) – The input data used to score and compute permutation importance
- **y** (*pd.Series*) – The target data
- **objective** (*str*, *ObjectiveBase*) – Objective to score on
- **importance\_threshold** (*float*, *optional*) – If provided, graph features with a permutation importance whose absolute value is larger than `importance_threshold`. Defaults to zero.

**Returns** plotly.Figure, a bar graph showing features and their respective permutation importance.

`evalml.model_understanding.graph_precision_recall_curve` (*y\_true*, *y\_pred\_proba*, *title\_addition=None*)

Generate and display a precision-recall plot.

**Parameters**

- **y\_true** (*pd.Series* or *np.ndarray*) – True binary labels.
- **y\_pred\_proba** (*pd.Series* or *np.ndarray*) – Predictions from a binary classifier, before thresholding has been applied. Note this should be the predicted probability for the “true” label.
- **title\_addition** (*str* or *None*) – If not *None*, append to plot title. Default *None*.

**Returns** plotly.Figure representing the precision-recall plot generated

`evalml.model_understanding.graph_prediction_vs_actual` (*y\_true*, *y\_pred*, *outlier\_threshold=None*)

Generate a scatter plot comparing the true and predicted values. Used for regression plotting

**Parameters**

- **y\_true** (*pd.Series*) – The real target values of the data
- **y\_pred** (*pd.Series*) – The predicted values outputted by the regression model.
- **outlier\_threshold** (*int, float*) – A positive threshold for what is considered an outlier value. This value is compared to the absolute difference between each value of y\_true and y\_pred. Values within this threshold will be blue, otherwise they will be yellow. Defaults to None

**Returns** *plotly*.Figure representing the predicted vs. actual values graph

`evalml.model_understanding.graph_prediction_vs_actual_over_time` (*pipeline, X, y, dates*)

Plot the target values and predictions against time on the x-axis.

**Parameters**

- **pipeline** (*TimeSeriesRegressionPipeline*) – Fitted time series regression pipeline.
- **X** (*pd.DataFrame*) – Features used to generate new predictions.
- **y** (*pd.Series*) – Target values to compare predictions against.
- **dates** (*pd.Series*) – Dates corresponding to target values and predictions.

**Returns** Showing the prediction vs actual over time.

**Return type** *plotly*.Figure

`evalml.model_understanding.graph_roc_curve` (*y\_true, y\_pred\_proba, custom\_class\_names=None, title\_addition=None*)

Generate and display a Receiver Operating Characteristic (ROC) plot for binary and multiclass classification problems.

**Parameters**

- **y\_true** (*pd.Series or np.ndarray*) – True labels.
- **y\_pred\_proba** (*pd.Series or np.ndarray*) – Predictions from a classifier, before thresholding has been applied. Note this should be a one dimensional array with the predicted probability for the “true” label in the binary case.
- **custom\_class\_labels** (*list or None*) – If not None, custom labels for classes. Default None.
- **title\_addition** (*str or None*) – if not None, append to plot title. Default None.

**Returns** *plotly*.Figure representing the ROC plot generated

`evalml.model_understanding.graph_t_sne` (*X, n\_components=2, perplexity=30.0, learning\_rate=200.0, metric='euclidean', marker\_line\_width=2, marker\_size=7, \*\*kwargs*)

Plot high dimensional data into lower dimensional space using t-SNE .

**Parameters**

- **X** (*np.ndarray, pd.DataFrame*) – Data to be transformed. Must be numeric.
- **n\_components** (*int, optional*) – Dimension of the embedded space.
- **perplexity** (*float, optional*) – Related to the number of nearest neighbors that is used in other manifold learning

- **Larger datasets usually require a larger perplexity.** Consider selecting a value between 5 and 50. (*algorithms.*)  
–
- **learning\_rate** (*float, optional*) – Usually in the range [10.0, 1000.0]. If the cost function gets stuck in a bad
- **minimum** (*local*) –
- **the learning rate may help.** (*increasing*) –
- **metric** (*str, optional*) – The metric to use when calculating distance between instances in a feature array.
- **marker\_line\_width** (*int, optional*) – Determines the line width of the marker boundary.
- **marker\_size** (*int, optional*) – Determines the size of the marker.

**Returns** `plotly.Figure` representing the transformed data

```
evalml.model_understanding.normalize_confusion_matrix(conf_mat, normalize_method='true')
```

Normalizes a confusion matrix.

#### Parameters

- **conf\_mat** (*pd.DataFrame or np.ndarray*) – Confusion matrix to normalize.
- **normalize\_method** (*{'true', 'pred', 'all'}*) – Normalization method. Supported options are: ‘true’ to normalize by row, ‘pred’ to normalize by column, or ‘all’ to normalize by all values. Defaults to ‘true’.

**Returns** normalized version of the input confusion matrix. The column header represents the predicted labels while row header represents the actual labels.

**Return type** `pd.DataFrame`

```
evalml.model_understanding.partial_dependence(pipeline, X, features, percentiles=(0.05, 0.95), grid_resolution=100, kind='average')
```

Calculates one or two-way partial dependence. If a single integer or string is given for features, one-way partial dependence is calculated. If a tuple of two integers or strings is given, two-way partial dependence is calculated with the first feature in the y-axis and second feature in the x-axis.

#### Parameters

- **pipeline** (*PipelineBase or subclass*) – Fitted pipeline
- **X** (*pd.DataFrame, np.ndarray*) – The input data used to generate a grid of values for feature where partial dependence will be calculated at
- **features** (*int, string, tuple[int or string]*) – The target feature for which to create the partial dependence plot for. If features is an int, it must be the index of the feature to use. If features is a string, it must be a valid column name in X. If features is a tuple of int/strings, it must contain valid column integers/names in X.
- **percentiles** (*tuple[float]*) – The lower and upper percentile used to create the extreme values for the grid. Must be in [0, 1]. Defaults to (0.05, 0.95).
- **grid\_resolution** (*int*) – Number of samples of feature(s) for partial dependence plot. If this value is less than the maximum number of categories present in categorical data within X, it will be set to the max number of categories + 1. Defaults to 100.



- **'average'** (*kind*) – The type of predictions to return. 'individual' will return the predictions for all of the points in the grid for each sample in X. 'average' will return the predictions for all of the points in the grid but averaged over all of the samples in X.
- **'individual'** – The type of predictions to return. 'individual' will return the predictions for all of the points in the grid for each sample in X. 'average' will return the predictions for all of the points in the grid but averaged over all of the samples in X.
- **'both'** – The type of predictions to return. 'individual' will return the predictions for all of the points in the grid for each sample in X. 'average' will return the predictions for all of the points in the grid but averaged over all of the samples in X.

## Returns

When *kind*='average': DataFrame with averaged predictions for all points in the grid averaged over all samples of X and the values used to calculate those predictions.

When *kind*='individual': DataFrame with individual predictions for all points in the grid for each sample of X and the values used to calculate those predictions. If a two-way partial dependence is calculated, then the result is a list of DataFrames with each DataFrame representing one sample's predictions.

When *kind*='both': A tuple consisting of the averaged predictions (in a DataFrame) over all samples of X and the individual predictions (in a list of DataFrames) for each sample of X.

In the one-way case: The dataframe will contain two columns, "feature\_values" (grid points at which the partial dependence was calculated) and "partial\_dependence" (the partial dependence at that feature value). For classification problems, there will be a third column called "class\_label" (the class label for which the partial dependence was calculated). For binary classification, the partial dependence is only calculated for the "positive" class.

In the two-way case: The data frame will contain grid\_resolution number of columns and rows where the index and column headers are the sampled values of the first and second features, respectively, used to make the partial dependence contour. The values of the data frame contain the partial dependence data for each feature value pair.

**Return type** pd.DataFrame, list(pd.DataFrame), or tuple(pd.DataFrame, list(pd.DataFrame))

## Raises

- **PartialDependenceError** – if the user provides a tuple of not exactly two features.
- **PartialDependenceError** – if the provided pipeline isn't fitted.
- **PartialDependenceError** – if the provided pipeline is a Baseline pipeline.
- **PartialDependenceError** – if any of the features passed in are completely NaN
- **PartialDependenceError** – if any of the features are low-variance. Defined as having one value occurring more than the upper percentile passed by the user. By default 95%.

```
evalml.model_understanding.precision_recall_curve(y_true, y_pred_proba,
                                                  pos_label_idx=-1)
```

Given labels and binary classifier predicted probabilities, compute and return the data representing a precision-recall curve.

## Parameters

- **y\_true** (pd.Series or np.ndarray) – True binary labels.
- **y\_pred\_proba** (pd.Series or np.ndarray) – Predictions from a binary classifier, before thresholding has been applied. Note this should be the predicted probability for the "true" label.

- **pos\_label\_idx** (*int*) – the column index corresponding to the positive class. If predicted probabilities are two-dimensional, this will be used to access the probabilities for the positive class.

**Returns**

Dictionary containing metrics used to generate a precision-recall plot, with the following keys:

- *precision*: Precision values.
- *recall*: Recall values.
- *thresholds*: Threshold values used to produce the precision and recall.
- *auc\_score*: The area under the ROC curve.

**Return type** list

`evalml.model_understanding.roc_curve(y_true, y_pred_proba)`

Given labels and classifier predicted probabilities, compute and return the data representing a Receiver Operating Characteristic (ROC) curve. Works with binary or multiclass problems.

**Parameters**

- **y\_true** (*pd.Series* or *np.ndarray*) – True labels.
- **y\_pred\_proba** (*pd.Series* or *np.ndarray*) – Predictions from a classifier, before thresholding has been applied.

**Returns**

A list of dictionaries (with one for each class) is returned. Binary classification problems return a list with one di

Each dictionary contains metrics used to generate an ROC plot with the following keys:

- *fpr\_rate*: False positive rate.
- *tpr\_rate*: True positive rate.
- *threshold*: Threshold values used to produce each pair of true/false positive rates.
- *auc\_score*: The area under the ROC curve.

**Return type** list(dict)

`evalml.model_understanding.t_sne(X, n_components=2, perplexity=30.0, learning_rate=200.0, metric='euclidean', **kwargs)`

Get the transformed output after fitting X to the embedded space using t-SNE.

**Arguments:** X (*np.ndarray*, *pd.DataFrame*): Data to be transformed. Must be numeric.  
n\_components (*int*, optional): Dimension of the embedded space. perplexity (*float*, optional): Related to the number of nearest neighbors that is used in other manifold learning algorithms. Larger datasets usually require a larger perplexity. Consider selecting a value between 5 and 50. learning\_rate (*float*, optional): Usually in the range [10.0, 1000.0]. If the cost function gets stuck in a bad local minimum, increasing the learning rate may help. metric (*str*, optional): The metric to use when calculating distance between instances in a feature array.

**Returns** *np.ndarray* (n\_samples, n\_components)

## Objectives

## Submodules

## binary\_classification\_objective

## Module Contents

## Classes Summary

<i>BinaryClassificationObjective</i>	Base class for all binary classification objectives.
--------------------------------------	--

## Contents

**class** evalml.objectives.binary\_classification\_objective.**BinaryClassificationObjective**  
Base class for all binary classification objectives.

### Attributes

<b>problem_types</b>	[ProblemTypes.BINARY, ProblemTypes.TIME_SERIES_BINARY]
----------------------	--

### Methods

<i>calculate_percent_difference</i>	Calculate the percent difference between scores.
<i>can_optimize_threshold</i>	Returns a boolean determining if we can optimize the binary classification objective threshold.
<i>decision_function</i>	Apply a learned threshold to predicted probabilities to get predicted classes.
<i>greater_is_better</i>	Returns a boolean determining if a greater score indicates better model performance.
<i>is_bounded_like_percentage</i>	Returns whether this objective is bounded between 0 and 1, inclusive.
<i>is_defined_for_problem_type</i>	
<i>name</i>	Returns a name describing the objective.
<i>objective_function</i>	Computes the relative value of the provided predictions compared to the actual labels, according a specified metric
<i>optimize_threshold</i>	Learn a binary classification threshold which optimizes the current objective.
<i>perfect_score</i>	Returns the score obtained by evaluating this objective on a perfect model.
<i>positive_only</i>	If True, this objective is only valid for positive data. Default False.
<i>score</i>	Returns a numerical score indicating performance based on the differences between the predicted and actual values.

continues on next page

Table 190 – continued from previous page

<code>score_needs_proba</code>	Returns a boolean determining if the <code>score()</code> method needs probability estimates. This should be true for objectives which work with predicted probabilities, like log loss or AUC, and false for objectives which compare predicted class labels to the actual labels, like F1 or correlation.
<code>validate_inputs</code>	Validates the input based on a few simple checks.

**classmethod** `calculate_percent_difference` (*cls*, *score*, *baseline\_score*)

Calculate the percent difference between scores.

#### Parameters

- **score** (*float*) – A score. Output of the score method of this objective.
- **baseline\_score** (*float*) – A score. Output of the score method of this objective. In practice, this is the score achieved on this objective with a baseline estimator.

#### Returns

**The percent difference between the scores. Note that for objectives that can be interpreted as percentages, this will be the difference between the reference score and score. For all other objectives, the difference will be normalized by the reference score.**

**Return type** float

**property** `can_optimize_threshold` (*cls*)

Returns a boolean determining if we can optimize the binary classification objective threshold. This will be false for any objective that works directly with predicted probabilities, like log loss and AUC. Otherwise, it will be true.

**decision\_function** (*self*, *ypred\_proba*, *threshold=0.5*, *X=None*)

Apply a learned threshold to predicted probabilities to get predicted classes.

#### Parameters

- **ypred\_proba** (*pd.Series*, *np.ndarray*) – The classifier’s predicted probabilities
- **threshold** (*float*, *optional*) – Threshold used to make a prediction. Defaults to 0.5.
- **X** (*pd.DataFrame*, *optional*) – Any extra columns that are needed from training data.

**Returns** predictions

**property** `greater_is_better` (*cls*)

Returns a boolean determining if a greater score indicates better model performance.

**property** `is_bounded_like_percentage` (*cls*)

Returns whether this objective is bounded between 0 and 1, inclusive.

**classmethod** `is_defined_for_problem_type` (*cls*, *problem\_type*)

**property** `name` (*cls*)

Returns a name describing the objective.

**abstract classmethod** `objective_function` (*cls*, *y\_true*, *y\_predicted*, *X=None*, *sample\_weight=None*)

Computes the relative value of the provided predictions compared to the actual labels, according a specified metric

**Arguments:** `y_predicted` (`pd.Series`): Predicted values of length `[n_samples]` `y_true` (`pd.Series`): Actual class labels of length `[n_samples]` `X` (`pd.DataFrame` or `np.ndarray`): Extra data of shape `[n_samples, n_features]` necessary to calculate score `sample_weight` (`pd.DataFrame` or `np.ndarray`): Sample weights used in computing objective value result

**Returns** Numerical value used to calculate score

**optimize\_threshold** (*self*, *ypred\_proba*, *y\_true*, *X=None*)

Learn a binary classification threshold which optimizes the current objective.

#### Parameters

- **ypred\_proba** (`pd.Series`) – The classifier’s predicted probabilities
- **y\_true** (`pd.Series`) – The ground truth for the predictions.
- **X** (`pd.DataFrame`, *optional*) – Any extra columns that are needed from training data.

**Returns** Optimal threshold for this objective

**property perfect\_score** (*cls*)

Returns the score obtained by evaluating this objective on a perfect model.

**positive\_only** (*cls*)

If True, this objective is only valid for positive data. Default False.

**score** (*self*, *y\_true*, *y\_predicted*, *X=None*, *sample\_weight=None*)

Returns a numerical score indicating performance based on the differences between the predicted and actual values.

#### Parameters

- **y\_predicted** (`pd.Series`) – Predicted values of length `[n_samples]`
- **y\_true** (`pd.Series`) – Actual class labels of length `[n_samples]`
- **X** (`pd.DataFrame` or `np.ndarray`) – Extra data of shape `[n_samples, n_features]` necessary to calculate score
- **sample\_weight** (`pd.DataFrame` or `np.ndarray`) – Sample weights used in computing objective value result

**Returns** score

**property score\_needs\_proba** (*cls*)

Returns a boolean determining if the `score()` method needs probability estimates. This should be true for objectives which work with predicted probabilities, like log loss or AUC, and false for objectives which compare predicted class labels to the actual labels, like F1 or correlation.

**validate\_inputs** (*self*, *y\_true*, *y\_predicted*)

Validates the input based on a few simple checks.

#### Parameters

- **y\_predicted** (`pd.Series`, or `pd.DataFrame`) – Predicted values of length `[n_samples]`
- **y\_true** (`pd.Series`) – Actual class labels of length `[n_samples]`

**Returns** None

## cost\_benefit\_matrix

### Module Contents

### Classes Summary

---

<i>CostBenefitMatrix</i>	Score using a cost-benefit matrix. Scores quantify the benefits of a given value, so greater numeric
--------------------------	--

---

### Contents

```
class evalml.objectives.cost_benefit_matrix.CostBenefitMatrix(true_positive,  
                                                             true_negative,  
                                                             false_positive,  
                                                             false_negative)
```

Score using a cost-benefit matrix. Scores quantify the benefits of a given value, so greater numeric scores represents a better score. Costs and scores can be negative, indicating that a value is not beneficial. For example, in the case of monetary profit, a negative cost and/or score represents loss of cash flow.

#### Parameters

- **true\_positive** (*float*) – Cost associated with true positive predictions
- **true\_negative** (*float*) – Cost associated with true negative predictions
- **false\_positive** (*float*) – Cost associated with false positive predictions
- **false\_negative** (*float*) – Cost associated with false negative predictions

#### Attributes

<b>greater_is_better</b>	True
<b>is_bounded_like_percentage</b>	False
<b>name</b>	Cost Benefit Matrix
<b>perfect_score</b>	None
<b>problem_types</b>	[ProblemTypes.BINARY, ProblemTypes.TIME_SERIES_BINARY]
<b>score_needs_proba</b>	False

#### Methods

---

<i>calculate_percent_difference</i>	Calculate the percent difference between scores.
<i>can_optimize_threshold</i>	Returns a boolean determining if we can optimize the binary classification objective threshold.
<i>decision_function</i>	Apply a learned threshold to predicted probabilities to get predicted classes.
<i>is_defined_for_problem_type</i>	
<i>objective_function</i>	Calculates cost-benefit of the using the predicted and true values.

---

continues on next page

Table 192 – continued from previous page

<code>optimize_threshold</code>	Learn a binary classification threshold which optimizes the current objective.
<code>positive_only</code>	If True, this objective is only valid for positive data. Default False.
<code>score</code>	Returns a numerical score indicating performance based on the differences between the predicted and actual values.
<code>validate_inputs</code>	Validates the input based on a few simple checks.

**classmethod** `calculate_percent_difference` (*cls*, *score*, *baseline\_score*)

Calculate the percent difference between scores.

#### Parameters

- **score** (*float*) – A score. Output of the score method of this objective.
- **baseline\_score** (*float*) – A score. Output of the score method of this objective. In practice, this is the score achieved on this objective with a baseline estimator.

#### Returns

**The percent difference between the scores. Note that for objectives that can be interpreted as percentages, this will be the difference between the reference score and score. For all other objectives, the difference will be normalized by the reference score.**

**Return type** `float`

**property** `can_optimize_threshold` (*cls*)

Returns a boolean determining if we can optimize the binary classification objective threshold. This will be false for any objective that works directly with predicted probabilities, like log loss and AUC. Otherwise, it will be true.

**decision\_function** (*self*, *ypred\_proba*, *threshold=0.5*, *X=None*)

Apply a learned threshold to predicted probabilities to get predicted classes.

#### Parameters

- **ypred\_proba** (*pd.Series*, *np.ndarray*) – The classifier’s predicted probabilities
- **threshold** (*float*, *optional*) – Threshold used to make a prediction. Defaults to 0.5.
- **X** (*pd.DataFrame*, *optional*) – Any extra columns that are needed from training data.

**Returns** `predictions`

**classmethod** `is_defined_for_problem_type` (*cls*, *problem\_type*)

**objective\_function** (*self*, *y\_true*, *y\_predicted*, *X=None*, *sample\_weight=None*)

Calculates cost-benefit of the using the predicted and true values.

#### Parameters

- **y\_predicted** (*pd.Series*) – Predicted labels
- **y\_true** (*pd.Series*) – True labels
- **X** (*pd.DataFrame*) – Ignored.
- **sample\_weight** (*pd.DataFrame*) – Ignored.

**Returns** Cost-benefit matrix score

**Return type** float

**optimize\_threshold** (*self*, *ypred\_proba*, *y\_true*, *X=None*)

Learn a binary classification threshold which optimizes the current objective.

**Parameters**

- **ypred\_proba** (*pd.Series*) – The classifier’s predicted probabilities
- **y\_true** (*pd.Series*) – The ground truth for the predictions.
- **X** (*pd.DataFrame*, *optional*) – Any extra columns that are needed from training data.

**Returns** Optimal threshold for this objective

**positive\_only** (*cls*)

If True, this objective is only valid for positive data. Default False.

**score** (*self*, *y\_true*, *y\_predicted*, *X=None*, *sample\_weight=None*)

Returns a numerical score indicating performance based on the differences between the predicted and actual values.

**Parameters**

- **y\_predicted** (*pd.Series*) – Predicted values of length [n\_samples]
- **y\_true** (*pd.Series*) – Actual class labels of length [n\_samples]
- **X** (*pd.DataFrame* or *np.ndarray*) – Extra data of shape [n\_samples, n\_features] necessary to calculate score
- **sample\_weight** (*pd.DataFrame* or *np.ndarray*) – Sample weights used in computing objective value result

**Returns** score

**validate\_inputs** (*self*, *y\_true*, *y\_predicted*)

Validates the input based on a few simple checks.

**Parameters**

- **y\_predicted** (*pd.Series*, or *pd.DataFrame*) – Predicted values of length [n\_samples]
- **y\_true** (*pd.Series*) – Actual class labels of length [n\_samples]

**Returns** None

## **fraud\_cost**

## **Module Contents**

## **Classes Summary**

---

*FraudCost*

Score the percentage of money lost of the total transaction amount process due to fraud.

---



## Contents

**class** evalml.objectives.fraud\_cost.**FraudCost** (*retry\_percentage=0.5*, *interchange\_fee=0.02*, *fraud\_payout\_percentage=1.0*, *amount\_col='amount'*) *in-*

Score the percentage of money lost of the total transaction amount process due to fraud.

### Parameters

- **retry\_percentage** (*float*) – What percentage of customers that will retry a transaction if it is declined. Between 0 and 1. Defaults to 0.5.
- **interchange\_fee** (*float*) – How much of each successful transaction you pay. Between 0 and 1. Defaults to 0.02.
- **fraud\_payout\_percentage** (*float*) – Percentage of fraud you will not be able to collect. Between 0 and 1. Defaults to 1.0.
- **amount\_col** (*str*) – Name of column in data that contains the amount. Defaults to “amount”.

### Attributes

<b>greater_is_better</b>	False
<b>is_bounded_like_percentage</b>	True
<b>name</b>	Fraud Cost
<b>perfect_score</b>	0.0
<b>problem_types</b>	[ProblemTypes.BINARY, ProblemTypes.TIME_SERIES_BINARY]
<b>score_needs_proba</b>	True

### Methods

<i>calculate_percent_difference</i>	Calculate the percent difference between scores.
<i>can_optimize_threshold</i>	Returns a boolean determining if we can optimize the binary classification objective threshold.
<i>decision_function</i>	Apply a learned threshold to predicted probabilities to get predicted classes.
<i>is_defined_for_problem_type</i>	
<i>objective_function</i>	Calculate amount lost to fraud per transaction given predictions, true values, and dataframe with transaction amount.
<i>optimize_threshold</i>	Learn a binary classification threshold which optimizes the current objective.
<i>positive_only</i>	If True, this objective is only valid for positive data. Default False.
<i>score</i>	Returns a numerical score indicating performance based on the differences between the predicted and actual values.
<i>validate_inputs</i>	Validates the input based on a few simple checks.

**classmethod** **calculate\_percent\_difference** (*cls*, *score*, *baseline\_score*)

Calculate the percent difference between scores.

#### Parameters

- **score** (*float*) – A score. Output of the score method of this objective.
- **baseline\_score** (*float*) – A score. Output of the score method of this objective. In practice, this is the score achieved on this objective with a baseline estimator.

#### Returns

**The percent difference between the scores. Note that for objectives that can be interpreted as percentages, this will be the difference between the reference score and score. For all other objectives, the difference will be normalized by the reference score.**

**Return type** float

**property** `can_optimize_threshold` (*cls*)

Returns a boolean determining if we can optimize the binary classification objective threshold. This will be false for any objective that works directly with predicted probabilities, like log loss and AUC. Otherwise, it will be true.

**decision\_function** (*self*, *ypred\_proba*, *threshold=0.5*, *X=None*)

Apply a learned threshold to predicted probabilities to get predicted classes.

#### Parameters

- **ypred\_proba** (*pd.Series*, *np.ndarray*) – The classifier’s predicted probabilities
- **threshold** (*float*, *optional*) – Threshold used to make a prediction. Defaults to 0.5.
- **X** (*pd.DataFrame*, *optional*) – Any extra columns that are needed from training data.

**Returns** predictions

**classmethod** `is_defined_for_problem_type` (*cls*, *problem\_type*)

**objective\_function** (*self*, *y\_true*, *y\_predicted*, *X*, *sample\_weight=None*)

Calculate amount lost to fraud per transaction given predictions, true values, and dataframe with transaction amount.

#### Parameters

- **y\_predicted** (*pd.Series*) – Predicted fraud labels
- **y\_true** (*pd.Series*) – True fraud labels
- **X** (*pd.DataFrame*) – Data with transaction amounts
- **sample\_weight** (*pd.DataFrame*) – Ignored.

**Returns** Amount lost to fraud per transaction

**Return type** float

**optimize\_threshold** (*self*, *ypred\_proba*, *y\_true*, *X=None*)

Learn a binary classification threshold which optimizes the current objective.

#### Parameters

- **ypred\_proba** (*pd.Series*) – The classifier’s predicted probabilities
- **y\_true** (*pd.Series*) – The ground truth for the predictions.

- **X** (*pd.DataFrame*, *optional*) – Any extra columns that are needed from training data.

**Returns** Optimal threshold for this objective

**positive\_only** (*cls*)

If True, this objective is only valid for positive data. Default False.

**score** (*self*, *y\_true*, *y\_predicted*, *X=None*, *sample\_weight=None*)

Returns a numerical score indicating performance based on the differences between the predicted and actual values.

**Parameters**

- **y\_predicted** (*pd.Series*) – Predicted values of length [n\_samples]
- **y\_true** (*pd.Series*) – Actual class labels of length [n\_samples]
- **X** (*pd.DataFrame* or *np.ndarray*) – Extra data of shape [n\_samples, n\_features] necessary to calculate score
- **sample\_weight** (*pd.DataFrame* or *np.ndarray*) – Sample weights used in computing objective value result

**Returns** score

**validate\_inputs** (*self*, *y\_true*, *y\_predicted*)

Validates the input based on a few simple checks.

**Parameters**

- **y\_predicted** (*pd.Series*, or *pd.DataFrame*) – Predicted values of length [n\_samples]
- **y\_true** (*pd.Series*) – Actual class labels of length [n\_samples]

**Returns** None

## lead\_scoring

## Module Contents

## Classes Summary

---

<i>LeadScoring</i>	Lead scoring.
--------------------	---------------

---

## Contents

**class** evalml.objectives.lead\_scoring.**LeadScoring** (*true\_positives=1*, *false\_positives=-1*)

Lead scoring.

**Parameters**

- **true\_positives** (*int*) – Reward for a true positive. Defaults to 1.
- **false\_positives** (*int*) – Cost for a false positive. Should be negative. Defaults to -1.

**Attributes**

<b>greater_is_better</b>	True
<b>is_bounded_like_percentage</b>	False
<b>name</b>	Lead Scoring
<b>perfect_score</b>	None
<b>problem_types</b>	[ProblemTypes.BINARY, ProblemTypes.TIME_SERIES_BINARY]
<b>score_needs_proba</b>	True

## Methods

<code>calculate_percent_difference</code>	Calculate the percent difference between scores.
<code>can_optimize_threshold</code>	Returns a boolean determining if we can optimize the binary classification objective threshold.
<code>decision_function</code>	Apply a learned threshold to predicted probabilities to get predicted classes.
<code>is_defined_for_problem_type</code>	
<code>objective_function</code>	Calculate the profit per lead.
<code>optimize_threshold</code>	Learn a binary classification threshold which optimizes the current objective.
<code>positive_only</code>	If True, this objective is only valid for positive data. Default False.
<code>score</code>	Returns a numerical score indicating performance based on the differences between the predicted and actual values.
<code>validate_inputs</code>	Validates the input based on a few simple checks.

**classmethod** `calculate_percent_difference` (*cls*, *score*, *baseline\_score*)

Calculate the percent difference between scores.

### Parameters

- **score** (*float*) – A score. Output of the score method of this objective.
- **baseline\_score** (*float*) – A score. Output of the score method of this objective. In practice, this is the score achieved on this objective with a baseline estimator.

### Returns

**The percent difference between the scores. Note that for objectives that can be interpreted as percentages, this will be the difference between the reference score and score. For all other objectives, the difference will be normalized by the reference score.**

**Return type** float

**property** `can_optimize_threshold` (*cls*)

Returns a boolean determining if we can optimize the binary classification objective threshold. This will be false for any objective that works directly with predicted probabilities, like log loss and AUC. Otherwise, it will be true.

**decision\_function** (*self*, *ypred\_proba*, *threshold=0.5*, *X=None*)

Apply a learned threshold to predicted probabilities to get predicted classes.

### Parameters

- **ypred\_proba** (*pd.Series, np.ndarray*) – The classifier’s predicted probabilities
- **threshold** (*float, optional*) – Threshold used to make a prediction. Defaults to 0.5.
- **X** (*pd.DataFrame, optional*) – Any extra columns that are needed from training data.

**Returns** predictions

**classmethod is\_defined\_for\_problem\_type** (*cls, problem\_type*)

**objective\_function** (*self, y\_true, y\_predicted, X=None, sample\_weight=None*)

Calculate the profit per lead.

**Parameters**

- **y\_predicted** (*pd.Series*) – Predicted labels
- **y\_true** (*pd.Series*) – True labels
- **X** (*pd.DataFrame*) – Ignored.
- **sample\_weight** (*pd.DataFrame*) – Ignored.

**Returns** Profit per lead

**Return type** float

**optimize\_threshold** (*self, ypred\_proba, y\_true, X=None*)

Learn a binary classification threshold which optimizes the current objective.

**Parameters**

- **ypred\_proba** (*pd.Series*) – The classifier’s predicted probabilities
- **y\_true** (*pd.Series*) – The ground truth for the predictions.
- **X** (*pd.DataFrame, optional*) – Any extra columns that are needed from training data.

**Returns** Optimal threshold for this objective

**positive\_only** (*cls*)

If True, this objective is only valid for positive data. Default False.

**score** (*self, y\_true, y\_predicted, X=None, sample\_weight=None*)

Returns a numerical score indicating performance based on the differences between the predicted and actual values.

**Parameters**

- **y\_predicted** (*pd.Series*) – Predicted values of length [n\_samples]
- **y\_true** (*pd.Series*) – Actual class labels of length [n\_samples]
- **X** (*pd.DataFrame or np.ndarray*) – Extra data of shape [n\_samples, n\_features] necessary to calculate score
- **sample\_weight** (*pd.DataFrame or np.ndarray*) – Sample weights used in computing objective value result

**Returns** score

**validate\_inputs** (*self, y\_true, y\_predicted*)

Validates the input based on a few simple checks.

**Parameters**

- **y\_predicted** (*pd.Series*, or *pd.DataFrame*) – Predicted values of length [n\_samples]
- **y\_true** (*pd.Series*) – Actual class labels of length [n\_samples]

**Returns** None**multiclass\_classification\_objective****Module Contents****Classes Summary**

---

<i>MulticlassClassificationObjective</i>	Base class for all multiclass classification objectives.
--	--

---

**Contents**

**class** evalml.objectives.multiclass\_classification\_objective.**MulticlassClassificationObjective**  
Base class for all multiclass classification objectives.

**Attributes**

<b>problem_types</b>	[ProblemTypes.MULTICLASS, ProblemTypes.TIME_SERIES_MULTICLASS]
----------------------	--

**Methods**

<i>calculate_percent_difference</i>	Calculate the percent difference between scores.
<i>greater_is_better</i>	Returns a boolean determining if a greater score indicates better model performance.
<i>is_bounded_like_percentage</i>	Returns whether this objective is bounded between 0 and 1, inclusive.
<i>is_defined_for_problem_type</i>	
<i>name</i>	Returns a name describing the objective.
<i>objective_function</i>	Computes the relative value of the provided predictions compared to the actual labels, according a specified metric
<i>perfect_score</i>	Returns the score obtained by evaluating this objective on a perfect model.
<i>positive_only</i>	If True, this objective is only valid for positive data. Default False.
<i>score</i>	Returns a numerical score indicating performance based on the differences between the predicted and actual values.

continues on next page

Table 198 – continued from previous page

<code>score_needs_proba</code>	Returns a boolean determining if the <code>score()</code> method needs probability estimates. This should be true for objectives which work with predicted probabilities, like log loss or AUC, and false for objectives which compare predicted class labels to the actual labels, like F1 or correlation.
<code>validate_inputs</code>	Validates the input based on a few simple checks.

**classmethod** `calculate_percent_difference` (*cls*, *score*, *baseline\_score*)

Calculate the percent difference between scores.

#### Parameters

- **score** (*float*) – A score. Output of the score method of this objective.
- **baseline\_score** (*float*) – A score. Output of the score method of this objective. In practice, this is the score achieved on this objective with a baseline estimator.

#### Returns

**The percent difference between the scores. Note that for objectives that can be interpreted** as percentages, this will be the difference between the reference score and score. For all other objectives, the difference will be normalized by the reference score.

**Return type** float

**property** `greater_is_better` (*cls*)

Returns a boolean determining if a greater score indicates better model performance.

**property** `is_bounded_like_percentage` (*cls*)

Returns whether this objective is bounded between 0 and 1, inclusive.

**classmethod** `is_defined_for_problem_type` (*cls*, *problem\_type*)

**property** `name` (*cls*)

Returns a name describing the objective.

**abstract classmethod** `objective_function` (*cls*, *y\_true*, *y\_predicted*, *X=None*, *sample\_weight=None*)

Computes the relative value of the provided predictions compared to the actual labels, according a specified metric

**Arguments:** *y\_predicted* (pd.Series): Predicted values of length [n\_samples] *y\_true* (pd.Series): Actual class labels of length [n\_samples] *X* (pd.DataFrame or np.ndarray): Extra data of shape [n\_samples, n\_features] necessary to calculate score *sample\_weight* (pd.DataFrame or np.ndarray): Sample weights used in computing objective value result

**Returns** Numerical value used to calculate score

**property** `perfect_score` (*cls*)

Returns the score obtained by evaluating this objective on a perfect model.

**property** `positive_only` (*cls*)

If True, this objective is only valid for positive data. Default False.

**score** (*self*, *y\_true*, *y\_predicted*, *X=None*, *sample\_weight=None*)

Returns a numerical score indicating performance based on the differences between the predicted and actual values.

#### Parameters

- **y\_predicted** (*pd.Series*) – Predicted values of length [n\_samples]
- **y\_true** (*pd.Series*) – Actual class labels of length [n\_samples]
- **X** (*pd.DataFrame* or *np.ndarray*) – Extra data of shape [n\_samples, n\_features] necessary to calculate score
- **sample\_weight** (*pd.DataFrame* or *np.ndarray*) – Sample weights used in computing objective value result

**Returns** score

**property score\_needs\_proba** (*cls*)

Returns a boolean determining if the score() method needs probability estimates. This should be true for objectives which work with predicted probabilities, like log loss or AUC, and false for objectives which compare predicted class labels to the actual labels, like F1 or correlation.

**validate\_inputs** (*self, y\_true, y\_predicted*)

Validates the input based on a few simple checks.

**Parameters**

- **y\_predicted** (*pd.Series*, or *pd.DataFrame*) – Predicted values of length [n\_samples]
- **y\_true** (*pd.Series*) – Actual class labels of length [n\_samples]

**Returns** None

## objective\_base

### Module Contents

### Classes Summary

---

<i>ObjectiveBase</i>	Base class for all objectives.
----------------------	--------------------------------

---

### Contents

**class** evalml.objectives.objective\_base.**ObjectiveBase**

Base class for all objectives.

**Attributes**

<b>problem_types</b>	None
----------------------	------

**Methods**

<i>calculate_percent_difference</i>	Calculate the percent difference between scores.
<i>greater_is_better</i>	Returns a boolean determining if a greater score indicates better model performance.
<i>is_bounded_like_percentage</i>	Returns whether this objective is bounded between 0 and 1, inclusive.

continues on next page



Table 200 – continued from previous page

<i>is_defined_for_problem_type</i>	
<i>name</i>	Returns a name describing the objective.
<i>objective_function</i>	Computes the relative value of the provided predictions compared to the actual labels, according a specified metric
<i>perfect_score</i>	Returns the score obtained by evaluating this objective on a perfect model.
<i>positive_only</i>	If True, this objective is only valid for positive data. Default False.
<i>score</i>	Returns a numerical score indicating performance based on the differences between the predicted and actual values.
<i>score_needs_proba</i>	Returns a boolean determining if the score() method needs probability estimates. This should be true for objectives which work with predicted probabilities, like log loss or AUC, and false for objectives which compare predicted class labels to the actual labels, like F1 or correlation.
<i>validate_inputs</i>	Validates the input based on a few simple checks.

**classmethod** `calculate_percent_difference` (*cls*, *score*, *baseline\_score*)

Calculate the percent difference between scores.

#### Parameters

- **score** (*float*) – A score. Output of the score method of this objective.
- **baseline\_score** (*float*) – A score. Output of the score method of this objective. In practice, this is the score achieved on this objective with a baseline estimator.

#### Returns

**The percent difference between the scores. Note that for objectives that can be interpreted** as percentages, this will be the difference between the reference score and score. For all other objectives, the difference will be normalized by the reference score.

**Return type** float

**property** `greater_is_better` (*cls*)

Returns a boolean determining if a greater score indicates better model performance.

**property** `is_bounded_like_percentage` (*cls*)

Returns whether this objective is bounded between 0 and 1, inclusive.

**classmethod** `is_defined_for_problem_type` (*cls*, *problem\_type*)

**property** `name` (*cls*)

Returns a name describing the objective.

**abstract classmethod** `objective_function` (*cls*, *y\_true*, *y\_predicted*, *X=None*, *sample\_weight=None*)

Computes the relative value of the provided predictions compared to the actual labels, according a specified metric

**Arguments:** *y\_predicted* (pd.Series): Predicted values of length [n\_samples] *y\_true* (pd.Series): Actual class labels of length [n\_samples] *X* (pd.DataFrame or np.ndarray): Extra data of shape [n\_samples, n\_features] necessary to calculate score *sample\_weight* (pd.DataFrame or np.ndarray): Sample weights used in computing objective value result

**Returns** Numerical value used to calculate score

**property perfect\_score** (*cls*)

Returns the score obtained by evaluating this objective on a perfect model.

**positive\_only** (*cls*)

If True, this objective is only valid for positive data. Default False.

**score** (*self*, *y\_true*, *y\_predicted*, *X=None*, *sample\_weight=None*)

Returns a numerical score indicating performance based on the differences between the predicted and actual values.

**Parameters**

- **y\_predicted** (*pd.Series*) – Predicted values of length [n\_samples]
- **y\_true** (*pd.Series*) – Actual class labels of length [n\_samples]
- **x** (*pd.DataFrame* or *np.ndarray*) – Extra data of shape [n\_samples, n\_features] necessary to calculate score
- **sample\_weight** (*pd.DataFrame* or *np.ndarray*) – Sample weights used in computing objective value result

**Returns** score

**property score\_needs\_proba** (*cls*)

Returns a boolean determining if the score() method needs probability estimates. This should be true for objectives which work with predicted probabilities, like log loss or AUC, and false for objectives which compare predicted class labels to the actual labels, like F1 or correlation.

**validate\_inputs** (*self*, *y\_true*, *y\_predicted*)

Validates the input based on a few simple checks.

**Parameters**

- **y\_predicted** (*pd.Series*, or *pd.DataFrame*) – Predicted values of length [n\_samples]
- **y\_true** (*pd.Series*) – Actual class labels of length [n\_samples]

**Returns** None

## regression\_objective

## Module Contents

## Classes Summary

---

*RegressionObjective*

Base class for all regression objectives.

---

## Contents

**class** evalml.objectives.regression\_objective.**RegressionObjective**

Base class for all regression objectives.

### Attributes

<b>problem_types</b>	[ProblemTypes.REGRESSION, ProblemTypes.TIME_SERIES_REGRESSION]
----------------------	--

### Methods

<i>calculate_percent_difference</i>	Calculate the percent difference between scores.
<i>greater_is_better</i>	Returns a boolean determining if a greater score indicates better model performance.
<i>is_bounded_like_percentage</i>	Returns whether this objective is bounded between 0 and 1, inclusive.
<i>is_defined_for_problem_type</i>	
<i>name</i>	Returns a name describing the objective.
<i>objective_function</i>	Computes the relative value of the provided predictions compared to the actual labels, according a specified metric
<i>perfect_score</i>	Returns the score obtained by evaluating this objective on a perfect model.
<i>positive_only</i>	If True, this objective is only valid for positive data. Default False.
<i>score</i>	Returns a numerical score indicating performance based on the differences between the predicted and actual values.
<i>score_needs_proba</i>	Returns a boolean determining if the score() method needs probability estimates. This should be true for objectives which work with predicted probabilities, like log loss or AUC, and false for objectives which compare predicted class labels to the actual labels, like F1 or correlation.
<i>validate_inputs</i>	Validates the input based on a few simple checks.

**classmethod** **calculate\_percent\_difference** (*cls*, *score*, *baseline\_score*)

Calculate the percent difference between scores.

### Parameters

- **score** (*float*) – A score. Output of the score method of this objective.
- **baseline\_score** (*float*) – A score. Output of the score method of this objective. In practice, this is the score achieved on this objective with a baseline estimator.

### Returns

**The percent difference between the scores. Note that for objectives that can be interpreted as percentages, this will be the difference between the reference score and score. For all other objectives, the difference will be normalized by the reference score.**

**Return type** float

**property** `greater_is_better` (*cls*)

Returns a boolean determining if a greater score indicates better model performance.

**property** `is_bounded_like_percentage` (*cls*)

Returns whether this objective is bounded between 0 and 1, inclusive.

**classmethod** `is_defined_for_problem_type` (*cls*, *problem\_type*)

**property** `name` (*cls*)

Returns a name describing the objective.

**abstract classmethod** `objective_function` (*cls*, *y\_true*, *y\_predicted*, *X=None*, *sample\_weight=None*)

Computes the relative value of the provided predictions compared to the actual labels, according a specified metric

**Arguments:** *y\_predicted* (*pd.Series*): Predicted values of length [*n\_samples*] *y\_true* (*pd.Series*): Actual class labels of length [*n\_samples*] *X* (*pd.DataFrame* or *np.ndarray*): Extra data of shape [*n\_samples*, *n\_features*] necessary to calculate score *sample\_weight* (*pd.DataFrame* or *np.ndarray*): Sample weights used in computing objective value result

**Returns** Numerical value used to calculate score

**property** `perfect_score` (*cls*)

Returns the score obtained by evaluating this objective on a perfect model.

**property** `positive_only` (*cls*)

If True, this objective is only valid for positive data. Default False.

**score** (*self*, *y\_true*, *y\_predicted*, *X=None*, *sample\_weight=None*)

Returns a numerical score indicating performance based on the differences between the predicted and actual values.

#### Parameters

- ***y\_predicted*** (*pd.Series*) – Predicted values of length [*n\_samples*]
- ***y\_true*** (*pd.Series*) – Actual class labels of length [*n\_samples*]
- ***X*** (*pd.DataFrame* or *np.ndarray*) – Extra data of shape [*n\_samples*, *n\_features*] necessary to calculate score
- ***sample\_weight*** (*pd.DataFrame* or *np.ndarray*) – Sample weights used in computing objective value result

**Returns** score

**property** `score_needs_proba` (*cls*)

Returns a boolean determining if the `score()` method needs probability estimates. This should be true for objectives which work with predicted probabilities, like log loss or AUC, and false for objectives which compare predicted class labels to the actual labels, like F1 or correlation.

**validate\_inputs** (*self*, *y\_true*, *y\_predicted*)

Validates the input based on a few simple checks.

#### Parameters

- ***y\_predicted*** (*pd.Series*, or *pd.DataFrame*) – Predicted values of length [*n\_samples*]
- ***y\_true*** (*pd.Series*) – Actual class labels of length [*n\_samples*]

**Returns** None

## sensitivity\_low\_alert

### Module Contents

### Classes Summary

<i>SensitivityLowAlert</i>	Base class for all binary classification objectives.
----------------------------	--

### Attributes Summary

<i>logger</i>
---------------

### Contents

evalml.objectives.sensitivity\_low\_alert.**logger**

**class** evalml.objectives.sensitivity\_low\_alert.**SensitivityLowAlert** (*alert\_rate=0.01*)  
Base class for all binary classification objectives.

#### Attributes

<b>greater_is_better</b>	True
<b>is_bounded_like_percentage</b>	True
<b>name</b>	Sensitivity at Low Alert Rates
<b>perfect_score</b>	1.0
<b>problem_types</b>	[ProblemTypes.BINARY, ProblemTypes.TIME_SERIES_BINARY]
<b>score_needs_proba</b>	True

#### Methods

<i>calculate_percent_difference</i>	Calculate the percent difference between scores.
<i>can_optimize_threshold</i>	Returns a boolean determining if we can optimize the binary classification objective threshold.
<i>decision_function</i>	Determine if an observation is high risk given an alert rate
<i>is_defined_for_problem_type</i>	
<i>objective_function</i>	Calculate sensitivity across all predictions, using the top alert_rate percent of observations as the predicted positive class
<i>optimize_threshold</i>	Learn a binary classification threshold which optimizes the current objective.
<i>positive_only</i>	If True, this objective is only valid for positive data. Default False.

continues on next page

Table 205 – continued from previous page

<code>score</code>	Returns a numerical score indicating performance based on the differences between the predicted and actual values.
<code>validate_inputs</code>	Validates the input based on a few simple checks.

**classmethod** `calculate_percent_difference` (*cls*, *score*, *baseline\_score*)

Calculate the percent difference between scores.

#### Parameters

- **score** (*float*) – A score. Output of the score method of this objective.
- **baseline\_score** (*float*) – A score. Output of the score method of this objective. In practice, this is the score achieved on this objective with a baseline estimator.

#### Returns

**The percent difference between the scores. Note that for objectives that can be interpreted as percentages, this will be the difference between the reference score and score. For all other objectives, the difference will be normalized by the reference score.**

**Return type** `float`

**property** `can_optimize_threshold` (*cls*)

Returns a boolean determining if we can optimize the binary classification objective threshold. This will be false for any objective that works directly with predicted probabilities, like log loss and AUC. Otherwise, it will be true.

**decision\_function** (*self*, *ypred\_proba*, *\*\*kwargs*)

Determine if an observation is high risk given an alert rate

**Parameters** `ypred_proba` (*pd.Series*) – Predicted probabilities

**classmethod** `is_defined_for_problem_type` (*cls*, *problem\_type*)

**objective\_function** (*self*, *y\_true*, *y\_predicted*, *\*\*kwargs*)

Calculate sensitivity across all predictions, using the top alert\_rate percent of observations as the predicted positive class

#### Parameters

- **y\_true** (*pd.Series*) – True labels
- **y\_predicted** (*pd.Series*) – Predicted labels based on alert\_rate

**Returns** sensitivity using the observations with the top scores as the predicted positive class

**Return type** `float`

**optimize\_threshold** (*self*, *ypred\_proba*, *y\_true*, *X=None*)

Learn a binary classification threshold which optimizes the current objective.

#### Parameters

- **ypred\_proba** (*pd.Series*) – The classifier’s predicted probabilities
- **y\_true** (*pd.Series*) – The ground truth for the predictions.
- **X** (*pd.DataFrame*, *optional*) – Any extra columns that are needed from training data.

**Returns** Optimal threshold for this objective

**positive\_only** (*cls*)

If True, this objective is only valid for positive data. Default False.

**score** (*self*, *y\_true*, *y\_predicted*, *X=None*, *sample\_weight=None*)

Returns a numerical score indicating performance based on the differences between the predicted and actual values.

**Parameters**

- **y\_predicted** (*pd.Series*) – Predicted values of length [n\_samples]
- **y\_true** (*pd.Series*) – Actual class labels of length [n\_samples]
- **X** (*pd.DataFrame* or *np.ndarray*) – Extra data of shape [n\_samples, n\_features] necessary to calculate score
- **sample\_weight** (*pd.DataFrame* or *np.ndarray*) – Sample weights used in computing objective value result

**Returns** score

**validate\_inputs** (*self*, *y\_true*, *y\_predicted*)

Validates the input based on a few simple checks.

**Parameters**

- **y\_predicted** (*pd.Series*, or *pd.DataFrame*) – Predicted values of length [n\_samples]
- **y\_true** (*pd.Series*) – Actual class labels of length [n\_samples]

**Returns** None

**standard\_metrics****Module Contents****Classes Summary**

<i>AccuracyBinary</i>	Accuracy score for binary classification.
<i>AccuracyMulticlass</i>	Accuracy score for multiclass classification.
<i>AUC</i>	AUC score for binary classification.
<i>AUCMacro</i>	AUC score for multiclass classification using macro averaging.
<i>AUCMicro</i>	AUC score for multiclass classification using micro averaging.
<i>AUCWeighted</i>	AUC Score for multiclass classification using weighted averaging.
<i>BalancedAccuracyBinary</i>	Balanced accuracy score for binary classification.
<i>BalancedAccuracyMulticlass</i>	Balanced accuracy score for multiclass classification.
<i>ExpVariance</i>	Explained variance score for regression.
<i>F1</i>	F1 score for binary classification.
<i>F1Macro</i>	F1 score for multiclass classification using macro averaging.
<i>F1Micro</i>	F1 score for multiclass classification using micro averaging.

continues on next page

Table 206 – continued from previous page

<i>F1Weighted</i>	F1 score for multiclass classification using weighted averaging.
<i>Gini</i>	Gini coefficient for binary classification.
<i>LogLossBinary</i>	Log Loss for binary classification.
<i>LogLossMulticlass</i>	Log Loss for multiclass classification.
<i>MAE</i>	Mean absolute error for regression.
<i>MAPE</i>	Mean absolute percentage error for time series regression. Scaled by 100 to return a percentage.
<i>MaxError</i>	Maximum residual error for regression.
<i>MCCBinary</i>	Matthews correlation coefficient for binary classification.
<i>MCCMulticlass</i>	Matthews correlation coefficient for multiclass classification.
<i>MeanSquaredLogError</i>	Mean squared log error for regression.
<i>MedianAE</i>	Median absolute error for regression.
<i>MSE</i>	Mean squared error for regression.
<i>Precision</i>	Precision score for binary classification.
<i>PrecisionMacro</i>	Precision score for multiclass classification using macro averaging.
<i>PrecisionMicro</i>	Precision score for multiclass classification using micro averaging.
<i>PrecisionWeighted</i>	Precision score for multiclass classification using weighted averaging.
<i>R2</i>	Coefficient of determination for regression.
<i>Recall</i>	Recall score for binary classification.
<i>RecallMacro</i>	Recall score for multiclass classification using macro averaging.
<i>RecallMicro</i>	Recall score for multiclass classification using micro averaging.
<i>RecallWeighted</i>	Recall score for multiclass classification using weighted averaging.
<i>RootMeanSquaredError</i>	Root mean squared error for regression.
<i>RootMeanSquaredLogError</i>	Root mean squared log error for regression.

## Contents

**class** evalml.objectives.standard\_metrics.**AccuracyBinary**

Accuracy score for binary classification.

### Attributes

<b>greater_is_better</b>	True
<b>is_bounded_like_percentage</b>	True
<b>name</b>	Accuracy Binary
<b>perfect_score</b>	1.0
<b>problem_types</b>	[ProblemTypes.BINARY, ProblemTypes.TIME_SERIES_BINARY]
<b>score_needs_proba</b>	False

### Methods



<code>calculate_percent_difference</code>	Calculate the percent difference between scores.
<code>can_optimize_threshold</code>	Returns a boolean determining if we can optimize the binary classification objective threshold.
<code>decision_function</code>	Apply a learned threshold to predicted probabilities to get predicted classes.
<code>is_defined_for_problem_type</code>	
<code>objective_function</code>	Computes the relative value of the provided predictions compared to the actual labels, according a specified metric
<code>optimize_threshold</code>	Learn a binary classification threshold which optimizes the current objective.
<code>positive_only</code>	If True, this objective is only valid for positive data. Default False.
<code>score</code>	Returns a numerical score indicating performance based on the differences between the predicted and actual values.
<code>validate_inputs</code>	Validates the input based on a few simple checks.

**classmethod** `calculate_percent_difference` (*cls*, *score*, *baseline\_score*)

Calculate the percent difference between scores.

#### Parameters

- **score** (*float*) – A score. Output of the score method of this objective.
- **baseline\_score** (*float*) – A score. Output of the score method of this objective. In practice, this is the score achieved on this objective with a baseline estimator.

#### Returns

**The percent difference between the scores. Note that for objectives that can be interpreted as percentages, this will be the difference between the reference score and score. For all other objectives, the difference will be normalized by the reference score.**

**Return type** float

**property** `can_optimize_threshold` (*cls*)

Returns a boolean determining if we can optimize the binary classification objective threshold. This will be false for any objective that works directly with predicted probabilities, like log loss and AUC. Otherwise, it will be true.

**decision\_function** (*self*, *ypred\_proba*, *threshold=0.5*, *X=None*)

Apply a learned threshold to predicted probabilities to get predicted classes.

#### Parameters

- **ypred\_proba** (*pd.Series*, *np.ndarray*) – The classifier’s predicted probabilities
- **threshold** (*float*, *optional*) – Threshold used to make a prediction. Defaults to 0.5.
- **X** (*pd.DataFrame*, *optional*) – Any extra columns that are needed from training data.

**Returns** predictions

**classmethod** `is_defined_for_problem_type` (*cls*, *problem\_type*)

**objective\_function** (*self*, *y\_true*, *y\_predicted*, *X=None*, *sample\_weight=None*)

Computes the relative value of the provided predictions compared to the actual labels, according a specified metric

**Arguments:** *y\_predicted* (*pd.Series*): Predicted values of length [*n\_samples*] *y\_true* (*pd.Series*): Actual class labels of length [*n\_samples*] *X* (*pd.DataFrame* or *np.ndarray*): Extra data of shape [*n\_samples*, *n\_features*] necessary to calculate score *sample\_weight* (*pd.DataFrame* or *np.ndarray*): Sample weights used in computing objective value result

**Returns** Numerical value used to calculate score

**optimize\_threshold** (*self*, *ypred\_proba*, *y\_true*, *X=None*)

Learn a binary classification threshold which optimizes the current objective.

**Parameters**

- **ypred\_proba** (*pd.Series*) – The classifier’s predicted probabilities
- **y\_true** (*pd.Series*) – The ground truth for the predictions.
- **X** (*pd.DataFrame*, *optional*) – Any extra columns that are needed from training data.

**Returns** Optimal threshold for this objective

**positive\_only** (*cls*)

If True, this objective is only valid for positive data. Default False.

**score** (*self*, *y\_true*, *y\_predicted*, *X=None*, *sample\_weight=None*)

Returns a numerical score indicating performance based on the differences between the predicted and actual values.

**Parameters**

- **y\_predicted** (*pd.Series*) – Predicted values of length [*n\_samples*]
- **y\_true** (*pd.Series*) – Actual class labels of length [*n\_samples*]
- **X** (*pd.DataFrame* or *np.ndarray*) – Extra data of shape [*n\_samples*, *n\_features*] necessary to calculate score
- **sample\_weight** (*pd.DataFrame* or *np.ndarray*) – Sample weights used in computing objective value result

**Returns** score

**validate\_inputs** (*self*, *y\_true*, *y\_predicted*)

Validates the input based on a few simple checks.

**Parameters**

- **y\_predicted** (*pd.Series*, or *pd.DataFrame*) – Predicted values of length [*n\_samples*]
- **y\_true** (*pd.Series*) – Actual class labels of length [*n\_samples*]

**Returns** None

**class** evalml.objectives.standard\_metrics.**AccuracyMulticlass**

Accuracy score for multiclass classification.

**Attributes**

<b>greater_is_better</b>	True
<b>is_bounded_like_percentage</b>	True
<b>name</b>	Accuracy Multiclass
<b>perfect_score</b>	1.0
<b>problem_types</b>	[ProblemTypes.MULTICLASS, ProblemTypes.TIME_SERIES_MULTICLASS]
<b>score_needs_proba</b>	False

## Methods

<i>calculate_percent_difference</i>	Calculate the percent difference between scores.
<i>is_defined_for_problem_type</i>	
<i>objective_function</i>	Computes the relative value of the provided predictions compared to the actual labels, according a specified metric
<i>positive_only</i>	If True, this objective is only valid for positive data. Default False.
<i>score</i>	Returns a numerical score indicating performance based on the differences between the predicted and actual values.
<i>validate_inputs</i>	Validates the input based on a few simple checks.

**classmethod** `calculate_percent_difference` (*cls*, *score*, *baseline\_score*)

Calculate the percent difference between scores.

### Parameters

- **score** (*float*) – A score. Output of the score method of this objective.
- **baseline\_score** (*float*) – A score. Output of the score method of this objective. In practice, this is the score achieved on this objective with a baseline estimator.

### Returns

**The percent difference between the scores. Note that for objectives that can be interpreted as percentages, this will be the difference between the reference score and score. For all other objectives, the difference will be normalized by the reference score.**

**Return type** float

**classmethod** `is_defined_for_problem_type` (*cls*, *problem\_type*)

**objective\_function** (*self*, *y\_true*, *y\_predicted*, *X=None*, *sample\_weight=None*)

Computes the relative value of the provided predictions compared to the actual labels, according a specified metric

**Arguments:** *y\_predicted* (pd.Series): Predicted values of length [n\_samples] *y\_true* (pd.Series): Actual class labels of length [n\_samples] *X* (pd.DataFrame or np.ndarray): Extra data of shape [n\_samples, n\_features] necessary to calculate score *sample\_weight* (pd.DataFrame or np.ndarray): Sample weights used in computing objective value result

**Returns** Numerical value used to calculate score

**positive\_only** (*cls*)

If True, this objective is only valid for positive data. Default False.

**score** (*self*, *y\_true*, *y\_predicted*, *X=None*, *sample\_weight=None*)

Returns a numerical score indicating performance based on the differences between the predicted and actual values.

#### Parameters

- **y\_predicted** (*pd.Series*) – Predicted values of length [*n\_samples*]
- **y\_true** (*pd.Series*) – Actual class labels of length [*n\_samples*]
- **X** (*pd.DataFrame* or *np.ndarray*) – Extra data of shape [*n\_samples*, *n\_features*] necessary to calculate score
- **sample\_weight** (*pd.DataFrame* or *np.ndarray*) – Sample weights used in computing objective value result

**Returns** score

**validate\_inputs** (*self*, *y\_true*, *y\_predicted*)

Validates the input based on a few simple checks.

#### Parameters

- **y\_predicted** (*pd.Series*, or *pd.DataFrame*) – Predicted values of length [*n\_samples*]
- **y\_true** (*pd.Series*) – Actual class labels of length [*n\_samples*]

**Returns** None

**class** evalml.objectives.standard\_metrics.**AUC**

AUC score for binary classification.

#### Attributes

<b>greater_is_better</b>	True
<b>is_bounded_like_percentage</b>	True
<b>name</b>	AUC
<b>perfect_score</b>	1.0
<b>problem_types</b>	[ProblemTypes.BINARY, ProblemTypes.TIME_SERIES_BINARY]
<b>score_needs_proba</b>	True

#### Methods

<i>calculate_percent_difference</i>	Calculate the percent difference between scores.
<i>can_optimize_threshold</i>	Returns a boolean determining if we can optimize the binary classification objective threshold.
<i>decision_function</i>	Apply a learned threshold to predicted probabilities to get predicted classes.
<i>is_defined_for_problem_type</i>	
<i>objective_function</i>	Computes the relative value of the provided predictions compared to the actual labels, according a specified metric

continues on next page

Table 209 – continued from previous page

<code>optimize_threshold</code>	Learn a binary classification threshold which optimizes the current objective.
<code>positive_only</code>	If True, this objective is only valid for positive data. Default False.
<code>score</code>	Returns a numerical score indicating performance based on the differences between the predicted and actual values.
<code>validate_inputs</code>	Validates the input based on a few simple checks.

**classmethod** `calculate_percent_difference` (*cls*, *score*, *baseline\_score*)

Calculate the percent difference between scores.

#### Parameters

- **score** (*float*) – A score. Output of the score method of this objective.
- **baseline\_score** (*float*) – A score. Output of the score method of this objective. In practice, this is the score achieved on this objective with a baseline estimator.

#### Returns

**The percent difference between the scores. Note that for objectives that can be interpreted as percentages, this will be the difference between the reference score and score. For all other objectives, the difference will be normalized by the reference score.**

**Return type** float

**property** `can_optimize_threshold` (*cls*)

Returns a boolean determining if we can optimize the binary classification objective threshold. This will be false for any objective that works directly with predicted probabilities, like log loss and AUC. Otherwise, it will be true.

**decision\_function** (*self*, *ypred\_proba*, *threshold=0.5*, *X=None*)

Apply a learned threshold to predicted probabilities to get predicted classes.

#### Parameters

- **ypred\_proba** (*pd.Series*, *np.ndarray*) – The classifier’s predicted probabilities
- **threshold** (*float*, *optional*) – Threshold used to make a prediction. Defaults to 0.5.
- **X** (*pd.DataFrame*, *optional*) – Any extra columns that are needed from training data.

**Returns** predictions

**classmethod** `is_defined_for_problem_type` (*cls*, *problem\_type*)

**objective\_function** (*self*, *y\_true*, *y\_predicted*, *X=None*, *sample\_weight=None*)

Computes the relative value of the provided predictions compared to the actual labels, according a specified metric

**Arguments:** *y\_predicted* (*pd.Series*): Predicted values of length [n\_samples] *y\_true* (*pd.Series*): Actual class labels of length [n\_samples] *X* (*pd.DataFrame* or *np.ndarray*): Extra data of shape [n\_samples, n\_features] necessary to calculate score *sample\_weight* (*pd.DataFrame* or *np.ndarray*): Sample weights used in computing objective value result

**Returns** Numerical value used to calculate score

**optimize\_threshold** (*self*, *ypred\_proba*, *y\_true*, *X=None*)

Learn a binary classification threshold which optimizes the current objective.

**Parameters**

- **ypred\_proba** (*pd.Series*) – The classifier’s predicted probabilities
- **y\_true** (*pd.Series*) – The ground truth for the predictions.
- **X** (*pd.DataFrame*, *optional*) – Any extra columns that are needed from training data.

**Returns** Optimal threshold for this objective

**positive\_only** (*cls*)

If True, this objective is only valid for positive data. Default False.

**score** (*self*, *y\_true*, *y\_predicted*, *X=None*, *sample\_weight=None*)

Returns a numerical score indicating performance based on the differences between the predicted and actual values.

**Parameters**

- **y\_predicted** (*pd.Series*) – Predicted values of length [n\_samples]
- **y\_true** (*pd.Series*) – Actual class labels of length [n\_samples]
- **X** (*pd.DataFrame* or *np.ndarray*) – Extra data of shape [n\_samples, n\_features] necessary to calculate score
- **sample\_weight** (*pd.DataFrame* or *np.ndarray*) – Sample weights used in computing objective value result

**Returns** score

**validate\_inputs** (*self*, *y\_true*, *y\_predicted*)

Validates the input based on a few simple checks.

**Parameters**

- **y\_predicted** (*pd.Series*, or *pd.DataFrame*) – Predicted values of length [n\_samples]
- **y\_true** (*pd.Series*) – Actual class labels of length [n\_samples]

**Returns** None

**class** evalml.objectives.standard\_metrics.**AUCMacro**

AUC score for multiclass classification using macro averaging.

**Attributes**

<b>greater_is_better</b>	True
<b>is_bounded_like_percentage</b>	True
<b>name</b>	AUC Macro
<b>perfect_score</b>	1.0
<b>problem_types</b>	[ProblemTypes.MULTICLASS, ProblemTypes.TIME_SERIES_MULTICLASS]
<b>score_needs_proba</b>	True

**Methods**

<code>calculate_percent_difference</code>	Calculate the percent difference between scores.
<code>is_defined_for_problem_type</code>	
<code>objective_function</code>	Computes the relative value of the provided predictions compared to the actual labels, according a specified metric
<code>positive_only</code>	If True, this objective is only valid for positive data. Default False.
<code>score</code>	Returns a numerical score indicating performance based on the differences between the predicted and actual values.
<code>validate_inputs</code>	Validates the input based on a few simple checks.

**classmethod** `calculate_percent_difference` (*cls*, *score*, *baseline\_score*)

Calculate the percent difference between scores.

#### Parameters

- **score** (*float*) – A score. Output of the score method of this objective.
- **baseline\_score** (*float*) – A score. Output of the score method of this objective. In practice, this is the score achieved on this objective with a baseline estimator.

#### Returns

**The percent difference between the scores. Note that for objectives that can be interpreted** as percentages, this will be the difference between the reference score and score. For all other objectives, the difference will be normalized by the reference score.

**Return type** float

**classmethod** `is_defined_for_problem_type` (*cls*, *problem\_type*)

**objective\_function** (*self*, *y\_true*, *y\_predicted*, *X=None*, *sample\_weight=None*)

Computes the relative value of the provided predictions compared to the actual labels, according a specified metric

**Arguments:** *y\_predicted* (*pd.Series*): Predicted values of length [*n\_samples*] *y\_true* (*pd.Series*): Actual class labels of length [*n\_samples*] *X* (*pd.DataFrame* or *np.ndarray*): Extra data of shape [*n\_samples*, *n\_features*] necessary to calculate score *sample\_weight* (*pd.DataFrame* or *np.ndarray*): Sample weights used in computing objective value result

**Returns** Numerical value used to calculate score

**positive\_only** (*cls*)

If True, this objective is only valid for positive data. Default False.

**score** (*self*, *y\_true*, *y\_predicted*, *X=None*, *sample\_weight=None*)

Returns a numerical score indicating performance based on the differences between the predicted and actual values.

#### Parameters

- **y\_predicted** (*pd.Series*) – Predicted values of length [*n\_samples*]
- **y\_true** (*pd.Series*) – Actual class labels of length [*n\_samples*]
- **X** (*pd.DataFrame* or *np.ndarray*) – Extra data of shape [*n\_samples*, *n\_features*] necessary to calculate score

- **sample\_weight** (*pd.DataFrame* or *np.ndarray*) – Sample weights used in computing objective value result

**Returns** score

**validate\_inputs** (*self*, *y\_true*, *y\_predicted*)

Validates the input based on a few simple checks.

**Parameters**

- **y\_predicted** (*pd.Series*, or *pd.DataFrame*) – Predicted values of length [n\_samples]
- **y\_true** (*pd.Series*) – Actual class labels of length [n\_samples]

**Returns** None

**class** evalml.objectives.standard\_metrics.**AUCMicro**

AUC score for multiclass classification using micro averaging.

**Attributes**

<b>greater_is_better</b>	True
<b>is_bounded_like_percentage</b>	True
<b>name</b>	AUC Micro
<b>perfect_score</b>	1.0
<b>problem_types</b>	[ProblemTypes.MULTICLASS, ProblemTypes.TIME_SERIES_MULTICLASS]
<b>score_needs_proba</b>	True

**Methods**

<i>calculate_percent_difference</i>	Calculate the percent difference between scores.
<i>is_defined_for_problem_type</i>	
<i>objective_function</i>	Computes the relative value of the provided predictions compared to the actual labels, according a specified metric
<i>positive_only</i>	If True, this objective is only valid for positive data. Default False.
<i>score</i>	Returns a numerical score indicating performance based on the differences between the predicted and actual values.
<i>validate_inputs</i>	Validates the input based on a few simple checks.

**classmethod** **calculate\_percent\_difference** (*cls*, *score*, *baseline\_score*)

Calculate the percent difference between scores.

**Parameters**

- **score** (*float*) – A score. Output of the score method of this objective.
- **baseline\_score** (*float*) – A score. Output of the score method of this objective. In practice, this is the score achieved on this objective with a baseline estimator.

**Returns**

The percent difference between the scores. Note that for objectives that can be interpreted



as percentages, this will be the difference between the reference score and score. For all other objectives, the difference will be normalized by the reference score.

**Return type** float

**classmethod** `is_defined_for_problem_type(cls, problem_type)`

**objective\_function** (*self*, *y\_true*, *y\_predicted*, *X=None*, *sample\_weight=None*)

Computes the relative value of the provided predictions compared to the actual labels, according a specified metric

**Arguments:** *y\_predicted* (pd.Series): Predicted values of length [n\_samples] *y\_true* (pd.Series): Actual class labels of length [n\_samples] *X* (pd.DataFrame or np.ndarray): Extra data of shape [n\_samples, n\_features] necessary to calculate score *sample\_weight* (pd.DataFrame or np.ndarray): Sample weights used in computing objective value result

**Returns** Numerical value used to calculate score

**positive\_only** (*cls*)

If True, this objective is only valid for positive data. Default False.

**score** (*self*, *y\_true*, *y\_predicted*, *X=None*, *sample\_weight=None*)

Returns a numerical score indicating performance based on the differences between the predicted and actual values.

**Parameters**

- **y\_predicted** (pd.Series) – Predicted values of length [n\_samples]
- **y\_true** (pd.Series) – Actual class labels of length [n\_samples]
- **X** (pd.DataFrame or np.ndarray) – Extra data of shape [n\_samples, n\_features] necessary to calculate score
- **sample\_weight** (pd.DataFrame or np.ndarray) – Sample weights used in computing objective value result

**Returns** score

**validate\_inputs** (*self*, *y\_true*, *y\_predicted*)

Validates the input based on a few simple checks.

**Parameters**

- **y\_predicted** (pd.Series, or pd.DataFrame) – Predicted values of length [n\_samples]
- **y\_true** (pd.Series) – Actual class labels of length [n\_samples]

**Returns** None

**class** evalml.objectives.standard\_metrics.**AUCWeighted**

AUC Score for multiclass classification using weighted averaging.

**Attributes**

<b>greater_is_better</b>	True
<b>is_bounded_like_percentage</b>	True
<b>name</b>	AUC Weighted
<b>perfect_score</b>	1.0
<b>problem_types</b>	[ProblemTypes.MULTICLASS, ProblemTypes.TIME_SERIES_MULTICLASS]
<b>score_needs_proba</b>	True

## Methods

<i>calculate_percent_difference</i>	Calculate the percent difference between scores.
<i>is_defined_for_problem_type</i>	
<i>objective_function</i>	Computes the relative value of the provided predictions compared to the actual labels, according a specified metric
<i>positive_only</i>	If True, this objective is only valid for positive data. Default False.
<i>score</i>	Returns a numerical score indicating performance based on the differences between the predicted and actual values.
<i>validate_inputs</i>	Validates the input based on a few simple checks.

**classmethod** `calculate_percent_difference` (*cls*, *score*, *baseline\_score*)

Calculate the percent difference between scores.

### Parameters

- **score** (*float*) – A score. Output of the score method of this objective.
- **baseline\_score** (*float*) – A score. Output of the score method of this objective. In practice, this is the score achieved on this objective with a baseline estimator.

### Returns

**The percent difference between the scores. Note that for objectives that can be interpreted as percentages, this will be the difference between the reference score and score. For all other objectives, the difference will be normalized by the reference score.**

**Return type** float

**classmethod** `is_defined_for_problem_type` (*cls*, *problem\_type*)

**objective\_function** (*self*, *y\_true*, *y\_predicted*, *X=None*, *sample\_weight=None*)

Computes the relative value of the provided predictions compared to the actual labels, according a specified metric

**Arguments:** *y\_predicted* (pd.Series): Predicted values of length [n\_samples] *y\_true* (pd.Series): Actual class labels of length [n\_samples] *X* (pd.DataFrame or np.ndarray): Extra data of shape [n\_samples, n\_features] necessary to calculate score *sample\_weight* (pd.DataFrame or np.ndarray): Sample weights used in computing objective value result

**Returns** Numerical value used to calculate score

**positive\_only** (*cls*)

If True, this objective is only valid for positive data. Default False.

**score** (*self, y\_true, y\_predicted, X=None, sample\_weight=None*)

Returns a numerical score indicating performance based on the differences between the predicted and actual values.

#### Parameters

- **y\_predicted** (*pd.Series*) – Predicted values of length [n\_samples]
- **y\_true** (*pd.Series*) – Actual class labels of length [n\_samples]
- **X** (*pd.DataFrame* or *np.ndarray*) – Extra data of shape [n\_samples, n\_features] necessary to calculate score
- **sample\_weight** (*pd.DataFrame* or *np.ndarray*) – Sample weights used in computing objective value result

**Returns** score

**validate\_inputs** (*self, y\_true, y\_predicted*)

Validates the input based on a few simple checks.

#### Parameters

- **y\_predicted** (*pd.Series*, or *pd.DataFrame*) – Predicted values of length [n\_samples]
- **y\_true** (*pd.Series*) – Actual class labels of length [n\_samples]

**Returns** None

**class** evalml.objectives.standard\_metrics.**BalancedAccuracyBinary**

Balanced accuracy score for binary classification.

#### Attributes

<b>greater_is_better</b>	True
<b>is_bounded_like_percentage</b>	True
<b>name</b>	Balanced Accuracy Binary
<b>perfect_score</b>	1.0
<b>problem_types</b>	[ProblemTypes.BINARY, ProblemTypes.TIME_SERIES_BINARY]
<b>score_needs_proba</b>	False

#### Methods

<i>calculate_percent_difference</i>	Calculate the percent difference between scores.
<i>can_optimize_threshold</i>	Returns a boolean determining if we can optimize the binary classification objective threshold.
<i>decision_function</i>	Apply a learned threshold to predicted probabilities to get predicted classes.
<i>is_defined_for_problem_type</i>	
<i>objective_function</i>	Computes the relative value of the provided predictions compared to the actual labels, according a specified metric

continues on next page

Table 213 – continued from previous page

<code>optimize_threshold</code>	Learn a binary classification threshold which optimizes the current objective.
<code>positive_only</code>	If True, this objective is only valid for positive data. Default False.
<code>score</code>	Returns a numerical score indicating performance based on the differences between the predicted and actual values.
<code>validate_inputs</code>	Validates the input based on a few simple checks.

**classmethod** `calculate_percent_difference` (*cls*, *score*, *baseline\_score*)

Calculate the percent difference between scores.

#### Parameters

- **score** (*float*) – A score. Output of the score method of this objective.
- **baseline\_score** (*float*) – A score. Output of the score method of this objective. In practice, this is the score achieved on this objective with a baseline estimator.

#### Returns

**The percent difference between the scores. Note that for objectives that can be interpreted** as percentages, this will be the difference between the reference score and score. For all other objectives, the difference will be normalized by the reference score.

**Return type** float

**property** `can_optimize_threshold` (*cls*)

Returns a boolean determining if we can optimize the binary classification objective threshold. This will be false for any objective that works directly with predicted probabilities, like log loss and AUC. Otherwise, it will be true.

**decision\_function** (*self*, *ypred\_proba*, *threshold=0.5*, *X=None*)

Apply a learned threshold to predicted probabilities to get predicted classes.

#### Parameters

- **ypred\_proba** (*pd.Series*, *np.ndarray*) – The classifier’s predicted probabilities
- **threshold** (*float*, *optional*) – Threshold used to make a prediction. Defaults to 0.5.
- **X** (*pd.DataFrame*, *optional*) – Any extra columns that are needed from training data.

**Returns** predictions

**classmethod** `is_defined_for_problem_type` (*cls*, *problem\_type*)

**objective\_function** (*self*, *y\_true*, *y\_predicted*, *X=None*, *sample\_weight=None*)

Computes the relative value of the provided predictions compared to the actual labels, according a specified metric

**Arguments:** *y\_predicted* (*pd.Series*): Predicted values of length [n\_samples] *y\_true* (*pd.Series*): Actual class labels of length [n\_samples] *X* (*pd.DataFrame* or *np.ndarray*): Extra data of shape [n\_samples, n\_features] necessary to calculate score *sample\_weight* (*pd.DataFrame* or *np.ndarray*): Sample weights used in computing objective value result

**Returns** Numerical value used to calculate score

**optimize\_threshold** (*self*, *ypred\_proba*, *y\_true*, *X=None*)

Learn a binary classification threshold which optimizes the current objective.

#### Parameters

- **ypred\_proba** (*pd.Series*) – The classifier’s predicted probabilities
- **y\_true** (*pd.Series*) – The ground truth for the predictions.
- **X** (*pd.DataFrame*, *optional*) – Any extra columns that are needed from training data.

**Returns** Optimal threshold for this objective

**positive\_only** (*cls*)

If True, this objective is only valid for positive data. Default False.

**score** (*self*, *y\_true*, *y\_predicted*, *X=None*, *sample\_weight=None*)

Returns a numerical score indicating performance based on the differences between the predicted and actual values.

#### Parameters

- **y\_predicted** (*pd.Series*) – Predicted values of length [n\_samples]
- **y\_true** (*pd.Series*) – Actual class labels of length [n\_samples]
- **X** (*pd.DataFrame* or *np.ndarray*) – Extra data of shape [n\_samples, n\_features] necessary to calculate score
- **sample\_weight** (*pd.DataFrame* or *np.ndarray*) – Sample weights used in computing objective value result

**Returns** score

**validate\_inputs** (*self*, *y\_true*, *y\_predicted*)

Validates the input based on a few simple checks.

#### Parameters

- **y\_predicted** (*pd.Series*, or *pd.DataFrame*) – Predicted values of length [n\_samples]
- **y\_true** (*pd.Series*) – Actual class labels of length [n\_samples]

**Returns** None

**class** evalml.objectives.standard\_metrics.**BalancedAccuracyMulticlass**

Balanced accuracy score for multiclass classification.

#### Attributes

<b>greater_is_better</b>	True
<b>is_bounded_like_percentage</b>	True
<b>name</b>	Balanced Accuracy Multiclass
<b>perfect_score</b>	1.0
<b>problem_types</b>	[ProblemTypes.MULTICLASS, ProblemTypes.TIME_SERIES_MULTICLASS]
<b>score_needs_proba</b>	False

#### Methods

<code>calculate_percent_difference</code>	Calculate the percent difference between scores.
<code>is_defined_for_problem_type</code>	
<code>objective_function</code>	Computes the relative value of the provided predictions compared to the actual labels, according a specified metric
<code>positive_only</code>	If True, this objective is only valid for positive data. Default False.
<code>score</code>	Returns a numerical score indicating performance based on the differences between the predicted and actual values.
<code>validate_inputs</code>	Validates the input based on a few simple checks.

**classmethod** `calculate_percent_difference` (*cls*, *score*, *baseline\_score*)

Calculate the percent difference between scores.

#### Parameters

- **score** (*float*) – A score. Output of the score method of this objective.
- **baseline\_score** (*float*) – A score. Output of the score method of this objective. In practice, this is the score achieved on this objective with a baseline estimator.

#### Returns

**The percent difference between the scores. Note that for objectives that can be interpreted** as percentages, this will be the difference between the reference score and score. For all other objectives, the difference will be normalized by the reference score.

**Return type** float

**classmethod** `is_defined_for_problem_type` (*cls*, *problem\_type*)

**objective\_function** (*self*, *y\_true*, *y\_predicted*, *X=None*, *sample\_weight=None*)

Computes the relative value of the provided predictions compared to the actual labels, according a specified metric

**Arguments:** *y\_predicted* (*pd.Series*): Predicted values of length [*n\_samples*] *y\_true* (*pd.Series*): Actual class labels of length [*n\_samples*] *X* (*pd.DataFrame* or *np.ndarray*): Extra data of shape [*n\_samples*, *n\_features*] necessary to calculate score *sample\_weight* (*pd.DataFrame* or *np.ndarray*): Sample weights used in computing objective value result

**Returns** Numerical value used to calculate score

**positive\_only** (*cls*)

If True, this objective is only valid for positive data. Default False.

**score** (*self*, *y\_true*, *y\_predicted*, *X=None*, *sample\_weight=None*)

Returns a numerical score indicating performance based on the differences between the predicted and actual values.

#### Parameters

- **y\_predicted** (*pd.Series*) – Predicted values of length [*n\_samples*]
- **y\_true** (*pd.Series*) – Actual class labels of length [*n\_samples*]
- **X** (*pd.DataFrame* or *np.ndarray*) – Extra data of shape [*n\_samples*, *n\_features*] necessary to calculate score

- **sample\_weight** (*pd.DataFrame* or *np.ndarray*) – Sample weights used in computing objective value result

**Returns** score

**validate\_inputs** (*self*, *y\_true*, *y\_predicted*)

Validates the input based on a few simple checks.

**Parameters**

- **y\_predicted** (*pd.Series*, or *pd.DataFrame*) – Predicted values of length [n\_samples]
- **y\_true** (*pd.Series*) – Actual class labels of length [n\_samples]

**Returns** None

**class** evalml.objectives.standard\_metrics.**ExpVariance**

Explained variance score for regression.

**Attributes**

<b>greater_is_better</b>	True
<b>is_bounded_like_percentage</b>	False
<b>name</b>	ExpVariance
<b>perfect_score</b>	1.0
<b>problem_types</b>	[ProblemTypes.REGRESSION, ProblemTypes.TIME_SERIES_REGRESSION]
<b>score_needs_proba</b>	False

**Methods**

<i>calculate_percent_difference</i>	Calculate the percent difference between scores.
<i>is_defined_for_problem_type</i>	
<i>objective_function</i>	Computes the relative value of the provided predictions compared to the actual labels, according a specified metric
<i>positive_only</i>	If True, this objective is only valid for positive data. Default False.
<i>score</i>	Returns a numerical score indicating performance based on the differences between the predicted and actual values.
<i>validate_inputs</i>	Validates the input based on a few simple checks.

**classmethod** **calculate\_percent\_difference** (*cls*, *score*, *baseline\_score*)

Calculate the percent difference between scores.

**Parameters**

- **score** (*float*) – A score. Output of the score method of this objective.
- **baseline\_score** (*float*) – A score. Output of the score method of this objective. In practice, this is the score achieved on this objective with a baseline estimator.

**Returns**

The percent difference between the scores. Note that for objectives that can be interpreted

as percentages, this will be the difference between the reference score and score. For all other objectives, the difference will be normalized by the reference score.

**Return type** float

**classmethod** `is_defined_for_problem_type(cls, problem_type)`

**objective\_function** (*self*, *y\_true*, *y\_predicted*, *X=None*, *sample\_weight=None*)

Computes the relative value of the provided predictions compared to the actual labels, according a specified metric

**Arguments:** *y\_predicted* (pd.Series): Predicted values of length [n\_samples] *y\_true* (pd.Series): Actual class labels of length [n\_samples] *X* (pd.DataFrame or np.ndarray): Extra data of shape [n\_samples, n\_features] necessary to calculate score *sample\_weight* (pd.DataFrame or np.ndarray): Sample weights used in computing objective value result

**Returns** Numerical value used to calculate score

**positive\_only** (*cls*)

If True, this objective is only valid for positive data. Default False.

**score** (*self*, *y\_true*, *y\_predicted*, *X=None*, *sample\_weight=None*)

Returns a numerical score indicating performance based on the differences between the predicted and actual values.

**Parameters**

- **y\_predicted** (pd.Series) – Predicted values of length [n\_samples]
- **y\_true** (pd.Series) – Actual class labels of length [n\_samples]
- **X** (pd.DataFrame or np.ndarray) – Extra data of shape [n\_samples, n\_features] necessary to calculate score
- **sample\_weight** (pd.DataFrame or np.ndarray) – Sample weights used in computing objective value result

**Returns** score

**validate\_inputs** (*self*, *y\_true*, *y\_predicted*)

Validates the input based on a few simple checks.

**Parameters**

- **y\_predicted** (pd.Series, or pd.DataFrame) – Predicted values of length [n\_samples]
- **y\_true** (pd.Series) – Actual class labels of length [n\_samples]

**Returns** None

**class** evalml.objectives.standard\_metrics.F1

F1 score for binary classification.

**Attributes**



<b>greater_is_better</b>	True
<b>is_bounded_like_percentage</b>	True
<b>name</b>	F1
<b>perfect_score</b>	1.0
<b>problem_types</b>	[ProblemTypes.BINARY, ProblemTypes.TIME_SERIES_BINARY]
<b>score_needs_proba</b>	True

## Methods

<code>calculate_percent_difference</code>	Calculate the percent difference between scores.
<code>can_optimize_threshold</code>	Returns a boolean determining if we can optimize the binary classification objective threshold.
<code>decision_function</code>	Apply a learned threshold to predicted probabilities to get predicted classes.
<code>is_defined_for_problem_type</code>	
<code>objective_function</code>	Computes the relative value of the provided predictions compared to the actual labels, according a specified metric
<code>optimize_threshold</code>	Learn a binary classification threshold which optimizes the current objective.
<code>positive_only</code>	If True, this objective is only valid for positive data. Default False.
<code>score</code>	Returns a numerical score indicating performance based on the differences between the predicted and actual values.
<code>validate_inputs</code>	Validates the input based on a few simple checks.

**classmethod** `calculate_percent_difference` (*cls*, *score*, *baseline\_score*)

Calculate the percent difference between scores.

### Parameters

- **score** (*float*) – A score. Output of the score method of this objective.
- **baseline\_score** (*float*) – A score. Output of the score method of this objective. In practice, this is the score achieved on this objective with a baseline estimator.

### Returns

**The percent difference between the scores. Note that for objectives that can be interpreted as percentages, this will be the difference between the reference score and score. For all other objectives, the difference will be normalized by the reference score.**

**Return type** float

**property** `can_optimize_threshold` (*cls*)

Returns a boolean determining if we can optimize the binary classification objective threshold. This will be false for any objective that works directly with predicted probabilities, like log loss and AUC. Otherwise, it will be true.

**decision\_function** (*self*, *ypred\_proba*, *threshold=0.5*, *X=None*)

Apply a learned threshold to predicted probabilities to get predicted classes.

**Parameters**

- **ypred\_proba** (*pd.Series, np.ndarray*) – The classifier’s predicted probabilities
- **threshold** (*float, optional*) – Threshold used to make a prediction. Defaults to 0.5.
- **X** (*pd.DataFrame, optional*) – Any extra columns that are needed from training data.

**Returns** predictions

**classmethod is\_defined\_for\_problem\_type** (*cls, problem\_type*)

**objective\_function** (*self, y\_true, y\_predicted, X=None, sample\_weight=None*)

Computes the relative value of the provided predictions compared to the actual labels, according a specified metric

**Arguments:** *y\_predicted* (*pd.Series*): Predicted values of length [*n\_samples*] *y\_true* (*pd.Series*): Actual class labels of length [*n\_samples*] *X* (*pd.DataFrame* or *np.ndarray*): Extra data of shape [*n\_samples*, *n\_features*] necessary to calculate score *sample\_weight* (*pd.DataFrame* or *np.ndarray*): Sample weights used in computing objective value result

**Returns** Numerical value used to calculate score

**optimize\_threshold** (*self, ypred\_proba, y\_true, X=None*)

Learn a binary classification threshold which optimizes the current objective.

**Parameters**

- **ypred\_proba** (*pd.Series*) – The classifier’s predicted probabilities
- **y\_true** (*pd.Series*) – The ground truth for the predictions.
- **X** (*pd.DataFrame, optional*) – Any extra columns that are needed from training data.

**Returns** Optimal threshold for this objective

**positive\_only** (*cls*)

If True, this objective is only valid for positive data. Default False.

**score** (*self, y\_true, y\_predicted, X=None, sample\_weight=None*)

Returns a numerical score indicating performance based on the differences between the predicted and actual values.

**Parameters**

- **y\_predicted** (*pd.Series*) – Predicted values of length [*n\_samples*]
- **y\_true** (*pd.Series*) – Actual class labels of length [*n\_samples*]
- **X** (*pd.DataFrame* or *np.ndarray*) – Extra data of shape [*n\_samples*, *n\_features*] necessary to calculate score
- **sample\_weight** (*pd.DataFrame* or *np.ndarray*) – Sample weights used in computing objective value result

**Returns** score

**validate\_inputs** (*self, y\_true, y\_predicted*)

Validates the input based on a few simple checks.

**Parameters**

- **y\_predicted** (*pd.Series*, or *pd.DataFrame*) – Predicted values of length [n\_samples]
- **y\_true** (*pd.Series*) – Actual class labels of length [n\_samples]

Returns None

**class** evalml.objectives.standard\_metrics.**F1Macro**

F1 score for multiclass classification using macro averaging.

Attributes

<b>greater_is_better</b>	True
<b>is_bounded_like_percentage</b>	True
<b>name</b>	F1 Macro
<b>perfect_score</b>	1.0
<b>problem_types</b>	[ProblemTypes.MULTICLASS, ProblemTypes.TIME_SERIES_MULTICLASS]
<b>score_needs_proba</b>	True

Methods

<i>calculate_percent_difference</i>	Calculate the percent difference between scores.
<i>is_defined_for_problem_type</i>	
<i>objective_function</i>	Computes the relative value of the provided predictions compared to the actual labels, according a specified metric
<i>positive_only</i>	If True, this objective is only valid for positive data. Default False.
<i>score</i>	Returns a numerical score indicating performance based on the differences between the predicted and actual values.
<i>validate_inputs</i>	Validates the input based on a few simple checks.

**classmethod** **calculate\_percent\_difference** (*cls*, *score*, *baseline\_score*)

Calculate the percent difference between scores.

Parameters

- **score** (*float*) – A score. Output of the score method of this objective.
- **baseline\_score** (*float*) – A score. Output of the score method of this objective. In practice, this is the score achieved on this objective with a baseline estimator.

Returns

**The percent difference between the scores. Note that for objectives that can be interpreted as percentages, this will be the difference between the reference score and score. For all other objectives, the difference will be normalized by the reference score.**

Return type float

**classmethod** **is\_defined\_for\_problem\_type** (*cls*, *problem\_type*)

**objective\_function** (*self*, *y\_true*, *y\_predicted*, *X=None*, *sample\_weight=None*)

Computes the relative value of the provided predictions compared to the actual labels, according a specified metric

**Arguments:** `y_predicted` (`pd.Series`): Predicted values of length `[n_samples]` `y_true` (`pd.Series`): Actual class labels of length `[n_samples]` `X` (`pd.DataFrame` or `np.ndarray`): Extra data of shape `[n_samples, n_features]` necessary to calculate score `sample_weight` (`pd.DataFrame` or `np.ndarray`): Sample weights used in computing objective value result

**Returns** Numerical value used to calculate score

**positive\_only** (*cls*)

If True, this objective is only valid for positive data. Default False.

**score** (*self*, *y\_true*, *y\_predicted*, *X=None*, *sample\_weight=None*)

Returns a numerical score indicating performance based on the differences between the predicted and actual values.

#### Parameters

- **y\_predicted** (*pd.Series*) – Predicted values of length `[n_samples]`
- **y\_true** (*pd.Series*) – Actual class labels of length `[n_samples]`
- **X** (*pd.DataFrame* or *np.ndarray*) – Extra data of shape `[n_samples, n_features]` necessary to calculate score
- **sample\_weight** (*pd.DataFrame* or *np.ndarray*) – Sample weights used in computing objective value result

**Returns** score

**validate\_inputs** (*self*, *y\_true*, *y\_predicted*)

Validates the input based on a few simple checks.

#### Parameters

- **y\_predicted** (*pd.Series*, or *pd.DataFrame*) – Predicted values of length `[n_samples]`
- **y\_true** (*pd.Series*) – Actual class labels of length `[n_samples]`

**Returns** None

**class** `evalml.objectives.standard_metrics.F1Micro`

F1 score for multiclass classification using micro averaging.

#### Attributes

<b>greater_is_better</b>	True
<b>is_bounded_like_percentage</b>	True
<b>name</b>	F1 Micro
<b>perfect_score</b>	1.0
<b>problem_types</b>	[ProblemTypes.MULTICLASS, ProblemTypes.TIME_SERIES_MULTICLASS]
<b>score_needs_proba</b>	False

#### Methods

<code>calculate_percent_difference</code>	Calculate the percent difference between scores.
<code>is_defined_for_problem_type</code>	

---

continues on next page

Table 218 – continued from previous page

<i>objective_function</i>	Computes the relative value of the provided predictions compared to the actual labels, according a specified metric
<i>positive_only</i>	If True, this objective is only valid for positive data. Default False.
<i>score</i>	Returns a numerical score indicating performance based on the differences between the predicted and actual values.
<i>validate_inputs</i>	Validates the input based on a few simple checks.

**classmethod** `calculate_percent_difference` (*cls*, *score*, *baseline\_score*)

Calculate the percent difference between scores.

#### Parameters

- **score** (*float*) – A score. Output of the score method of this objective.
- **baseline\_score** (*float*) – A score. Output of the score method of this objective. In practice, this is the score achieved on this objective with a baseline estimator.

#### Returns

**The percent difference between the scores. Note that for objectives that can be interpreted** as percentages, this will be the difference between the reference score and score. For all other objectives, the difference will be normalized by the reference score.

**Return type** float

**classmethod** `is_defined_for_problem_type` (*cls*, *problem\_type*)

**objective\_function** (*self*, *y\_true*, *y\_predicted*, *X=None*, *sample\_weight=None*)

Computes the relative value of the provided predictions compared to the actual labels, according a specified metric

**Arguments:** *y\_predicted* (*pd.Series*): Predicted values of length [*n\_samples*] *y\_true* (*pd.Series*): Actual class labels of length [*n\_samples*] *X* (*pd.DataFrame* or *np.ndarray*): Extra data of shape [*n\_samples*, *n\_features*] necessary to calculate score *sample\_weight* (*pd.DataFrame* or *np.ndarray*): Sample weights used in computing objective value result

**Returns** Numerical value used to calculate score

**positive\_only** (*cls*)

If True, this objective is only valid for positive data. Default False.

**score** (*self*, *y\_true*, *y\_predicted*, *X=None*, *sample\_weight=None*)

Returns a numerical score indicating performance based on the differences between the predicted and actual values.

#### Parameters

- **y\_predicted** (*pd.Series*) – Predicted values of length [*n\_samples*]
- **y\_true** (*pd.Series*) – Actual class labels of length [*n\_samples*]
- **X** (*pd.DataFrame* or *np.ndarray*) – Extra data of shape [*n\_samples*, *n\_features*] necessary to calculate score
- **sample\_weight** (*pd.DataFrame* or *np.ndarray*) – Sample weights used in computing objective value result

**Returns** score

**validate\_inputs** (*self*, *y\_true*, *y\_predicted*)

Validates the input based on a few simple checks.

**Parameters**

- **y\_predicted** (*pd.Series*, or *pd.DataFrame*) – Predicted values of length [n\_samples]
- **y\_true** (*pd.Series*) – Actual class labels of length [n\_samples]

**Returns** None

**class** evalml.objectives.standard\_metrics.**F1Weighted**

F1 score for multiclass classification using weighted averaging.

**Attributes**

<b>greater_is_better</b>	True
<b>is_bounded_like_percentage</b>	True
<b>name</b>	F1 Weighted
<b>perfect_score</b>	1.0
<b>problem_types</b>	[ProblemTypes.MULTICLASS, ProblemTypes.TIME_SERIES_MULTICLASS]
<b>score_needs_proba</b>	False

**Methods**

<i>calculate_percent_difference</i>	Calculate the percent difference between scores.
<i>is_defined_for_problem_type</i>	
<i>objective_function</i>	Computes the relative value of the provided predictions compared to the actual labels, according a specified metric
<i>positive_only</i>	If True, this objective is only valid for positive data. Default False.
<i>score</i>	Returns a numerical score indicating performance based on the differences between the predicted and actual values.
<i>validate_inputs</i>	Validates the input based on a few simple checks.

**classmethod** **calculate\_percent\_difference** (*cls*, *score*, *baseline\_score*)

Calculate the percent difference between scores.

**Parameters**

- **score** (*float*) – A score. Output of the score method of this objective.
- **baseline\_score** (*float*) – A score. Output of the score method of this objective. In practice, this is the score achieved on this objective with a baseline estimator.

**Returns**

**The percent difference between the scores. Note that for objectives that can be interpreted as percentages, this will be the difference between the reference score and score. For all other objectives, the difference will be normalized by the reference score.**

**Return type** float

**classmethod** `is_defined_for_problem_type(cls, problem_type)`

**objective\_function** (*self*, *y\_true*, *y\_predicted*, *X=None*, *sample\_weight=None*)

Computes the relative value of the provided predictions compared to the actual labels, according a specified metric

**Arguments:** *y\_predicted* (*pd.Series*): Predicted values of length [*n\_samples*] *y\_true* (*pd.Series*): Actual class labels of length [*n\_samples*] *X* (*pd.DataFrame* or *np.ndarray*): Extra data of shape [*n\_samples*, *n\_features*] necessary to calculate score *sample\_weight* (*pd.DataFrame* or *np.ndarray*): Sample weights used in computing objective value result

**Returns** Numerical value used to calculate score

**positive\_only** (*cls*)

If True, this objective is only valid for positive data. Default False.

**score** (*self*, *y\_true*, *y\_predicted*, *X=None*, *sample\_weight=None*)

Returns a numerical score indicating performance based on the differences between the predicted and actual values.

**Parameters**

- **y\_predicted** (*pd.Series*) – Predicted values of length [*n\_samples*]
- **y\_true** (*pd.Series*) – Actual class labels of length [*n\_samples*]
- **X** (*pd.DataFrame* or *np.ndarray*) – Extra data of shape [*n\_samples*, *n\_features*] necessary to calculate score
- **sample\_weight** (*pd.DataFrame* or *np.ndarray*) – Sample weights used in computing objective value result

**Returns** score

**validate\_inputs** (*self*, *y\_true*, *y\_predicted*)

Validates the input based on a few simple checks.

**Parameters**

- **y\_predicted** (*pd.Series*, or *pd.DataFrame*) – Predicted values of length [*n\_samples*]
- **y\_true** (*pd.Series*) – Actual class labels of length [*n\_samples*]

**Returns** None

**class** `evalml.objectives.standard_metrics.Gini`

Gini coefficient for binary classification.

**Attributes**

<b>greater_is_better</b>	True
<b>is_bounded_like_percentage</b>	False
<b>name</b>	Gini
<b>perfect_score</b>	1.0
<b>problem_types</b>	[ProblemTypes.BINARY, ProblemTypes.TIME_SERIES_BINARY]
<b>score_needs_proba</b>	True

**Methods**

<code>calculate_percent_difference</code>	Calculate the percent difference between scores.
<code>can_optimize_threshold</code>	Returns a boolean determining if we can optimize the binary classification objective threshold.
<code>decision_function</code>	Apply a learned threshold to predicted probabilities to get predicted classes.
<code>is_defined_for_problem_type</code>	
<code>objective_function</code>	Computes the relative value of the provided predictions compared to the actual labels, according a specified metric
<code>optimize_threshold</code>	Learn a binary classification threshold which optimizes the current objective.
<code>positive_only</code>	If True, this objective is only valid for positive data. Default False.
<code>score</code>	Returns a numerical score indicating performance based on the differences between the predicted and actual values.
<code>validate_inputs</code>	Validates the input based on a few simple checks.

**classmethod** `calculate_percent_difference` (*cls*, *score*, *baseline\_score*)

Calculate the percent difference between scores.

#### Parameters

- **score** (*float*) – A score. Output of the score method of this objective.
- **baseline\_score** (*float*) – A score. Output of the score method of this objective. In practice, this is the score achieved on this objective with a baseline estimator.

#### Returns

**The percent difference between the scores. Note that for objectives that can be interpreted as percentages, this will be the difference between the reference score and score. For all other objectives, the difference will be normalized by the reference score.**

**Return type** float

**property** `can_optimize_threshold` (*cls*)

Returns a boolean determining if we can optimize the binary classification objective threshold. This will be false for any objective that works directly with predicted probabilities, like log loss and AUC. Otherwise, it will be true.

**decision\_function** (*self*, *ypred\_proba*, *threshold=0.5*, *X=None*)

Apply a learned threshold to predicted probabilities to get predicted classes.

#### Parameters

- **ypred\_proba** (*pd.Series*, *np.ndarray*) – The classifier’s predicted probabilities
- **threshold** (*float*, *optional*) – Threshold used to make a prediction. Defaults to 0.5.
- **X** (*pd.DataFrame*, *optional*) – Any extra columns that are needed from training data.

**Returns** predictions

**classmethod** `is_defined_for_problem_type` (*cls*, *problem\_type*)



**objective\_function** (*self*, *y\_true*, *y\_predicted*, *X=None*, *sample\_weight=None*)

Computes the relative value of the provided predictions compared to the actual labels, according a specified metric

**Arguments:** *y\_predicted* (*pd.Series*): Predicted values of length [*n\_samples*] *y\_true* (*pd.Series*): Actual class labels of length [*n\_samples*] *X* (*pd.DataFrame* or *np.ndarray*): Extra data of shape [*n\_samples*, *n\_features*] necessary to calculate score *sample\_weight* (*pd.DataFrame* or *np.ndarray*): Sample weights used in computing objective value result

**Returns** Numerical value used to calculate score

**optimize\_threshold** (*self*, *ypred\_proba*, *y\_true*, *X=None*)

Learn a binary classification threshold which optimizes the current objective.

**Parameters**

- **ypred\_proba** (*pd.Series*) – The classifier’s predicted probabilities
- **y\_true** (*pd.Series*) – The ground truth for the predictions.
- **X** (*pd.DataFrame*, *optional*) – Any extra columns that are needed from training data.

**Returns** Optimal threshold for this objective

**positive\_only** (*cls*)

If True, this objective is only valid for positive data. Default False.

**score** (*self*, *y\_true*, *y\_predicted*, *X=None*, *sample\_weight=None*)

Returns a numerical score indicating performance based on the differences between the predicted and actual values.

**Parameters**

- **y\_predicted** (*pd.Series*) – Predicted values of length [*n\_samples*]
- **y\_true** (*pd.Series*) – Actual class labels of length [*n\_samples*]
- **X** (*pd.DataFrame* or *np.ndarray*) – Extra data of shape [*n\_samples*, *n\_features*] necessary to calculate score
- **sample\_weight** (*pd.DataFrame* or *np.ndarray*) – Sample weights used in computing objective value result

**Returns** score

**validate\_inputs** (*self*, *y\_true*, *y\_predicted*)

Validates the input based on a few simple checks.

**Parameters**

- **y\_predicted** (*pd.Series*, or *pd.DataFrame*) – Predicted values of length [*n\_samples*]
- **y\_true** (*pd.Series*) – Actual class labels of length [*n\_samples*]

**Returns** None

**class** evalml.objectives.standard\_metrics.**LogLossBinary**

Log Loss for binary classification.

**Attributes**

<b>greater_is_better</b>	False
<b>is_bounded_like_percentage</b>	False
<b>name</b>	Log Loss Binary
<b>perfect_score</b>	0.0
<b>problem_types</b>	[ProblemTypes.BINARY, ProblemTypes.TIME_SERIES_BINARY]
<b>score_needs_proba</b>	True

## Methods

<code>calculate_percent_difference</code>	Calculate the percent difference between scores.
<code>can_optimize_threshold</code>	Returns a boolean determining if we can optimize the binary classification objective threshold.
<code>decision_function</code>	Apply a learned threshold to predicted probabilities to get predicted classes.
<code>is_defined_for_problem_type</code>	
<code>objective_function</code>	Computes the relative value of the provided predictions compared to the actual labels, according a specified metric
<code>optimize_threshold</code>	Learn a binary classification threshold which optimizes the current objective.
<code>positive_only</code>	If True, this objective is only valid for positive data. Default False.
<code>score</code>	Returns a numerical score indicating performance based on the differences between the predicted and actual values.
<code>validate_inputs</code>	Validates the input based on a few simple checks.

**classmethod** `calculate_percent_difference` (*cls*, *score*, *baseline\_score*)

Calculate the percent difference between scores.

### Parameters

- **score** (*float*) – A score. Output of the score method of this objective.
- **baseline\_score** (*float*) – A score. Output of the score method of this objective. In practice, this is the score achieved on this objective with a baseline estimator.

### Returns

**The percent difference between the scores. Note that for objectives that can be interpreted as percentages, this will be the difference between the reference score and score. For all other objectives, the difference will be normalized by the reference score.**

**Return type** float

**property** `can_optimize_threshold` (*cls*)

Returns a boolean determining if we can optimize the binary classification objective threshold. This will be false for any objective that works directly with predicted probabilities, like log loss and AUC. Otherwise, it will be true.

**decision\_function** (*self*, *ypred\_proba*, *threshold=0.5*, *X=None*)

Apply a learned threshold to predicted probabilities to get predicted classes.

**Parameters**

- **ypred\_proba** (*pd.Series, np.ndarray*) – The classifier’s predicted probabilities
- **threshold** (*float, optional*) – Threshold used to make a prediction. Defaults to 0.5.
- **X** (*pd.DataFrame, optional*) – Any extra columns that are needed from training data.

**Returns** predictions

**classmethod is\_defined\_for\_problem\_type** (*cls, problem\_type*)

**objective\_function** (*self, y\_true, y\_predicted, X=None, sample\_weight=None*)

Computes the relative value of the provided predictions compared to the actual labels, according a specified metric

**Arguments:** *y\_predicted* (*pd.Series*): Predicted values of length [*n\_samples*] *y\_true* (*pd.Series*): Actual class labels of length [*n\_samples*] *X* (*pd.DataFrame* or *np.ndarray*): Extra data of shape [*n\_samples*, *n\_features*] necessary to calculate score *sample\_weight* (*pd.DataFrame* or *np.ndarray*): Sample weights used in computing objective value result

**Returns** Numerical value used to calculate score

**optimize\_threshold** (*self, ypred\_proba, y\_true, X=None*)

Learn a binary classification threshold which optimizes the current objective.

**Parameters**

- **ypred\_proba** (*pd.Series*) – The classifier’s predicted probabilities
- **y\_true** (*pd.Series*) – The ground truth for the predictions.
- **X** (*pd.DataFrame, optional*) – Any extra columns that are needed from training data.

**Returns** Optimal threshold for this objective

**positive\_only** (*cls*)

If True, this objective is only valid for positive data. Default False.

**score** (*self, y\_true, y\_predicted, X=None, sample\_weight=None*)

Returns a numerical score indicating performance based on the differences between the predicted and actual values.

**Parameters**

- **y\_predicted** (*pd.Series*) – Predicted values of length [*n\_samples*]
- **y\_true** (*pd.Series*) – Actual class labels of length [*n\_samples*]
- **X** (*pd.DataFrame* or *np.ndarray*) – Extra data of shape [*n\_samples*, *n\_features*] necessary to calculate score
- **sample\_weight** (*pd.DataFrame* or *np.ndarray*) – Sample weights used in computing objective value result

**Returns** score

**validate\_inputs** (*self, y\_true, y\_predicted*)

Validates the input based on a few simple checks.

**Parameters**

- **y\_predicted** (*pd.Series*, or *pd.DataFrame*) – Predicted values of length [n\_samples]
- **y\_true** (*pd.Series*) – Actual class labels of length [n\_samples]

**Returns** None

**class** evalml.objectives.standard\_metrics.**LogLossMulticlass**

Log Loss for multiclass classification.

**Attributes**

<b>greater_is_better</b>	False
<b>is_bounded_like_percentage</b>	True
<b>name</b>	Log Loss Multiclass
<b>perfect_score</b>	0.0
<b>problem_types</b>	[ProblemTypes.MULTICLASS, ProblemTypes.TIME_SERIES_MULTICLASS]
<b>score_needs_proba</b>	True

**Methods**

<i>calculate_percent_difference</i>	Calculate the percent difference between scores.
<i>is_defined_for_problem_type</i>	
<i>objective_function</i>	Computes the relative value of the provided predictions compared to the actual labels, according a specified metric
<i>positive_only</i>	If True, this objective is only valid for positive data. Default False.
<i>score</i>	Returns a numerical score indicating performance based on the differences between the predicted and actual values.
<i>validate_inputs</i>	Validates the input based on a few simple checks.

**classmethod** **calculate\_percent\_difference** (*cls*, *score*, *baseline\_score*)

Calculate the percent difference between scores.

**Parameters**

- **score** (*float*) – A score. Output of the score method of this objective.
- **baseline\_score** (*float*) – A score. Output of the score method of this objective. In practice, this is the score achieved on this objective with a baseline estimator.

**Returns**

**The percent difference between the scores. Note that for objectives that can be interpreted as percentages, this will be the difference between the reference score and score. For all other objectives, the difference will be normalized by the reference score.**

**Return type** float

**classmethod** **is\_defined\_for\_problem\_type** (*cls*, *problem\_type*)

**objective\_function** (*self*, *y\_true*, *y\_predicted*, *X=None*, *sample\_weight=None*)

Computes the relative value of the provided predictions compared to the actual labels, according a specified metric

**Arguments:** `y_predicted` (pd.Series): Predicted values of length `[n_samples]` `y_true` (pd.Series): Actual class labels of length `[n_samples]` `X` (pd.DataFrame or np.ndarray): Extra data of shape `[n_samples, n_features]` necessary to calculate score `sample_weight` (pd.DataFrame or np.ndarray): Sample weights used in computing objective value result

**Returns** Numerical value used to calculate score

**positive\_only** (*cls*)

If True, this objective is only valid for positive data. Default False.

**score** (*self*, *y\_true*, *y\_predicted*, *X=None*, *sample\_weight=None*)

Returns a numerical score indicating performance based on the differences between the predicted and actual values.

**Parameters**

- **y\_predicted** (pd.Series) – Predicted values of length `[n_samples]`
- **y\_true** (pd.Series) – Actual class labels of length `[n_samples]`
- **X** (pd.DataFrame or np.ndarray) – Extra data of shape `[n_samples, n_features]` necessary to calculate score
- **sample\_weight** (pd.DataFrame or np.ndarray) – Sample weights used in computing objective value result

**Returns** score

**validate\_inputs** (*self*, *y\_true*, *y\_predicted*)

Validates the input based on a few simple checks.

**Parameters**

- **y\_predicted** (pd.Series, or pd.DataFrame) – Predicted values of length `[n_samples]`
- **y\_true** (pd.Series) – Actual class labels of length `[n_samples]`

**Returns** None

**class** evalml.objectives.standard\_metrics.**MAE**

Mean absolute error for regression.

**Attributes**

<b>greater_is_better</b>	False
<b>is_bounded_like</b>	percentage
<b>name</b>	MAE
<b>perfect_score</b>	0.0
<b>problem_types</b>	[ProblemTypes.REGRESSION, ProblemTypes.TIME_SERIES_REGRESSION]
<b>score_needs_problem</b>	True

**Methods**

<a href="#"><code>calculate_percent_difference</code></a>	Calculate the percent difference between scores.
<a href="#"><code>is_defined_for_problem_type</code></a>	

continues on next page

Table 223 – continued from previous page

<i>objective_function</i>	Computes the relative value of the provided predictions compared to the actual labels, according a specified metric
<i>positive_only</i>	If True, this objective is only valid for positive data. Default False.
<i>score</i>	Returns a numerical score indicating performance based on the differences between the predicted and actual values.
<i>validate_inputs</i>	Validates the input based on a few simple checks.

**classmethod** `calculate_percent_difference` (*cls*, *score*, *baseline\_score*)

Calculate the percent difference between scores.

#### Parameters

- **score** (*float*) – A score. Output of the score method of this objective.
- **baseline\_score** (*float*) – A score. Output of the score method of this objective. In practice, this is the score achieved on this objective with a baseline estimator.

#### Returns

**The percent difference between the scores. Note that for objectives that can be interpreted as percentages, this will be the difference between the reference score and score. For all other objectives, the difference will be normalized by the reference score.**

**Return type** float

**classmethod** `is_defined_for_problem_type` (*cls*, *problem\_type*)

**objective\_function** (*self*, *y\_true*, *y\_predicted*, *X=None*, *sample\_weight=None*)

Computes the relative value of the provided predictions compared to the actual labels, according a specified metric

**Arguments:** *y\_predicted* (*pd.Series*): Predicted values of length [*n\_samples*] *y\_true* (*pd.Series*): Actual class labels of length [*n\_samples*] *X* (*pd.DataFrame* or *np.ndarray*): Extra data of shape [*n\_samples*, *n\_features*] necessary to calculate score *sample\_weight* (*pd.DataFrame* or *np.ndarray*): Sample weights used in computing objective value result

**Returns** Numerical value used to calculate score

**positive\_only** (*cls*)

If True, this objective is only valid for positive data. Default False.

**score** (*self*, *y\_true*, *y\_predicted*, *X=None*, *sample\_weight=None*)

Returns a numerical score indicating performance based on the differences between the predicted and actual values.

#### Parameters

- **y\_predicted** (*pd.Series*) – Predicted values of length [*n\_samples*]
- **y\_true** (*pd.Series*) – Actual class labels of length [*n\_samples*]
- **X** (*pd.DataFrame* or *np.ndarray*) – Extra data of shape [*n\_samples*, *n\_features*] necessary to calculate score
- **sample\_weight** (*pd.DataFrame* or *np.ndarray*) – Sample weights used in computing objective value result

**Returns** score

**validate\_inputs** (*self*, *y\_true*, *y\_predicted*)

Validates the input based on a few simple checks.

#### Parameters

- **y\_predicted** (*pd.Series*, or *pd.DataFrame*) – Predicted values of length [n\_samples]
- **y\_true** (*pd.Series*) – Actual class labels of length [n\_samples]

**Returns** None

**class** evalml.objectives.standard\_metrics.**MAPE**

Mean absolute percentage error for time series regression. Scaled by 100 to return a percentage.

Only valid for nonzero inputs. Otherwise, will throw a ValueError

#### Attributes

<b>greater_is_better</b>	False
<b>is_bounded_like_percentage</b>	True
<b>name</b>	Mean Absolute Percentage Error
<b>perfect_score</b>	0.0
<b>problem_types</b>	[ProblemTypes.TIME_SERIES_REGRESSION]
<b>score_needs_positive</b>	True

#### Methods

<i>calculate_percent_difference</i>	Calculate the percent difference between scores.
<i>is_defined_for_problem_type</i>	
<i>objective_function</i>	Computes the relative value of the provided predictions compared to the actual labels, according a specified metric
<i>positive_only</i>	If True, this objective is only valid for positive data. Default False.
<i>score</i>	Returns a numerical score indicating performance based on the differences between the predicted and actual values.
<i>validate_inputs</i>	Validates the input based on a few simple checks.

**classmethod** **calculate\_percent\_difference** (*cls*, *score*, *baseline\_score*)

Calculate the percent difference between scores.

#### Parameters

- **score** (*float*) – A score. Output of the score method of this objective.
- **baseline\_score** (*float*) – A score. Output of the score method of this objective. In practice, this is the score achieved on this objective with a baseline estimator.

#### Returns

**The percent difference between the scores. Note that for objectives that can be interpreted as percentages, this will be the difference between the reference score and score. For all other objectives, the difference will be normalized by the reference score.**

**Return type** float

**classmethod** `is_defined_for_problem_type(cls, problem_type)`

**objective\_function** (*self*, *y\_true*, *y\_predicted*, *X=None*, *sample\_weight=None*)

Computes the relative value of the provided predictions compared to the actual labels, according a specified metric

**Arguments:** *y\_predicted* (pd.Series): Predicted values of length [n\_samples] *y\_true* (pd.Series): Actual class labels of length [n\_samples] *X* (pd.DataFrame or np.ndarray): Extra data of shape [n\_samples, n\_features] necessary to calculate score *sample\_weight* (pd.DataFrame or np.ndarray): Sample weights used in computing objective value result

**Returns** Numerical value used to calculate score

**positive\_only** (*self*)

If True, this objective is only valid for positive data. Default False.

**score** (*self*, *y\_true*, *y\_predicted*, *X=None*, *sample\_weight=None*)

Returns a numerical score indicating performance based on the differences between the predicted and actual values.

**Parameters**

- **y\_predicted** (pd.Series) – Predicted values of length [n\_samples]
- **y\_true** (pd.Series) – Actual class labels of length [n\_samples]
- **X** (pd.DataFrame or np.ndarray) – Extra data of shape [n\_samples, n\_features] necessary to calculate score
- **sample\_weight** (pd.DataFrame or np.ndarray) – Sample weights used in computing objective value result

**Returns** score

**validate\_inputs** (*self*, *y\_true*, *y\_predicted*)

Validates the input based on a few simple checks.

**Parameters**

- **y\_predicted** (pd.Series, or pd.DataFrame) – Predicted values of length [n\_samples]
- **y\_true** (pd.Series) – Actual class labels of length [n\_samples]

**Returns** None

**class** evalml.objectives.standard\_metrics.**MaxError**

Maximum residual error for regression.

**Attributes**

<b>greater_is_better</b>	False
<b>is_bounded_like_percentage</b>	False
<b>name</b>	MaxError
<b>perfect_score</b>	0.0
<b>problem_types</b>	[ProblemTypes.REGRESSION, ProblemTypes.TIME_SERIES_REGRESSION]
<b>score_needs_proba</b>	False



## Methods

<code>calculate_percent_difference</code>	Calculate the percent difference between scores.
<code>is_defined_for_problem_type</code>	
<code>objective_function</code>	Computes the relative value of the provided predictions compared to the actual labels, according a specified metric
<code>positive_only</code>	If True, this objective is only valid for positive data. Default False.
<code>score</code>	Returns a numerical score indicating performance based on the differences between the predicted and actual values.
<code>validate_inputs</code>	Validates the input based on a few simple checks.

**classmethod** `calculate_percent_difference` (*cls*, *score*, *baseline\_score*)

Calculate the percent difference between scores.

### Parameters

- **score** (*float*) – A score. Output of the score method of this objective.
- **baseline\_score** (*float*) – A score. Output of the score method of this objective. In practice, this is the score achieved on this objective with a baseline estimator.

### Returns

**The percent difference between the scores. Note that for objectives that can be interpreted** as percentages, this will be the difference between the reference score and score. For all other objectives, the difference will be normalized by the reference score.

**Return type** float

**classmethod** `is_defined_for_problem_type` (*cls*, *problem\_type*)

**objective\_function** (*self*, *y\_true*, *y\_predicted*, *X=None*, *sample\_weight=None*)

Computes the relative value of the provided predictions compared to the actual labels, according a specified metric

**Arguments:** *y\_predicted* (*pd.Series*): Predicted values of length [*n\_samples*] *y\_true* (*pd.Series*): Actual class labels of length [*n\_samples*] *X* (*pd.DataFrame* or *np.ndarray*): Extra data of shape [*n\_samples*, *n\_features*] necessary to calculate score *sample\_weight* (*pd.DataFrame* or *np.ndarray*): Sample weights used in computing objective value result

**Returns** Numerical value used to calculate score

**positive\_only** (*cls*)

If True, this objective is only valid for positive data. Default False.

**score** (*self*, *y\_true*, *y\_predicted*, *X=None*, *sample\_weight=None*)

Returns a numerical score indicating performance based on the differences between the predicted and actual values.

### Parameters

- **y\_predicted** (*pd.Series*) – Predicted values of length [*n\_samples*]
- **y\_true** (*pd.Series*) – Actual class labels of length [*n\_samples*]

- **X** (*pd.DataFrame* or *np.ndarray*) – Extra data of shape [n\_samples, n\_features] necessary to calculate score
- **sample\_weight** (*pd.DataFrame* or *np.ndarray*) – Sample weights used in computing objective value result

**Returns** score

**validate\_inputs** (*self*, *y\_true*, *y\_predicted*)

Validates the input based on a few simple checks.

**Parameters**

- **y\_predicted** (*pd.Series*, or *pd.DataFrame*) – Predicted values of length [n\_samples]
- **y\_true** (*pd.Series*) – Actual class labels of length [n\_samples]

**Returns** None

**class** evalml.objectives.standard\_metrics.**MCCBinary**

Matthews correlation coefficient for binary classification.

**Attributes**

<b>greater_is_better</b>	True
<b>is_bounded_like_percentage</b>	False
<b>name</b>	MCC Binary
<b>perfect_score</b>	1.0
<b>problem_types</b>	[ProblemTypes.BINARY, ProblemTypes.TIME_SERIES_BINARY]
<b>score_needs_proba</b>	False

**Methods**

<i>calculate_percent_difference</i>	Calculate the percent difference between scores.
<i>can_optimize_threshold</i>	Returns a boolean determining if we can optimize the binary classification objective threshold.
<i>decision_function</i>	Apply a learned threshold to predicted probabilities to get predicted classes.
<i>is_defined_for_problem_type</i>	
<i>objective_function</i>	Computes the relative value of the provided predictions compared to the actual labels, according a specified metric
<i>optimize_threshold</i>	Learn a binary classification threshold which optimizes the current objective.
<i>positive_only</i>	If True, this objective is only valid for positive data. Default False.
<i>score</i>	Returns a numerical score indicating performance based on the differences between the predicted and actual values.
<i>validate_inputs</i>	Validates the input based on a few simple checks.

**classmethod** **calculate\_percent\_difference** (*cls*, *score*, *baseline\_score*)

Calculate the percent difference between scores.

**Parameters**

- **score** (*float*) – A score. Output of the score method of this objective.
- **baseline\_score** (*float*) – A score. Output of the score method of this objective. In practice, this is the score achieved on this objective with a baseline estimator.

**Returns**

**The percent difference between the scores. Note that for objectives that can be interpreted** as percentages, this will be the difference between the reference score and score. For all other objectives, the difference will be normalized by the reference score.

**Return type** float

**property can\_optimize\_threshold** (*cls*)

Returns a boolean determining if we can optimize the binary classification objective threshold. This will be false for any objective that works directly with predicted probabilities, like log loss and AUC. Otherwise, it will be true.

**decision\_function** (*self, ypred\_proba, threshold=0.5, X=None*)

Apply a learned threshold to predicted probabilities to get predicted classes.

**Parameters**

- **ypred\_proba** (*pd.Series, np.ndarray*) – The classifier’s predicted probabilities
- **threshold** (*float, optional*) – Threshold used to make a prediction. Defaults to 0.5.
- **X** (*pd.DataFrame, optional*) – Any extra columns that are needed from training data.

**Returns** predictions

**classmethod is\_defined\_for\_problem\_type** (*cls, problem\_type*)

**objective\_function** (*self, y\_true, y\_predicted, X=None, sample\_weight=None*)

Computes the relative value of the provided predictions compared to the actual labels, according a specified metric

**Arguments:** *y\_predicted* (*pd.Series*): Predicted values of length [n\_samples] *y\_true* (*pd.Series*): Actual class labels of length [n\_samples] *X* (*pd.DataFrame* or *np.ndarray*): Extra data of shape [n\_samples, n\_features] necessary to calculate score *sample\_weight* (*pd.DataFrame* or *np.ndarray*): Sample weights used in computing objective value result

**Returns** Numerical value used to calculate score

**optimize\_threshold** (*self, ypred\_proba, y\_true, X=None*)

Learn a binary classification threshold which optimizes the current objective.

**Parameters**

- **ypred\_proba** (*pd.Series*) – The classifier’s predicted probabilities
- **y\_true** (*pd.Series*) – The ground truth for the predictions.
- **X** (*pd.DataFrame, optional*) – Any extra columns that are needed from training data.

**Returns** Optimal threshold for this objective

**positive\_only** (*cls*)

If True, this objective is only valid for positive data. Default False.

**score** (*self*, *y\_true*, *y\_predicted*, *X=None*, *sample\_weight=None*)

Returns a numerical score indicating performance based on the differences between the predicted and actual values.

#### Parameters

- **y\_predicted** (*pd.Series*) – Predicted values of length [n\_samples]
- **y\_true** (*pd.Series*) – Actual class labels of length [n\_samples]
- **X** (*pd.DataFrame* or *np.ndarray*) – Extra data of shape [n\_samples, n\_features] necessary to calculate score
- **sample\_weight** (*pd.DataFrame* or *np.ndarray*) – Sample weights used in computing objective value result

**Returns** score

**validate\_inputs** (*self*, *y\_true*, *y\_predicted*)

Validates the input based on a few simple checks.

#### Parameters

- **y\_predicted** (*pd.Series*, or *pd.DataFrame*) – Predicted values of length [n\_samples]
- **y\_true** (*pd.Series*) – Actual class labels of length [n\_samples]

**Returns** None

**class** evalml.objectives.standard\_metrics.**MCCMulticlass**

Matthews correlation coefficient for multiclass classification.

#### Attributes

<b>greater_is_better</b>	True
<b>is_bounded_like_percentage</b>	False
<b>name</b>	MCC Multiclass
<b>perfect_score</b>	1.0
<b>problem_types</b>	[ProblemTypes.MULTICLASS, ProblemTypes.TIME_SERIES_MULTICLASS]
<b>score_needs_proba</b>	False

#### Methods

<i>calculate_percent_difference</i>	Calculate the percent difference between scores.
<i>is_defined_for_problem_type</i>	
<i>objective_function</i>	Computes the relative value of the provided predictions compared to the actual labels, according a specified metric
<i>positive_only</i>	If True, this objective is only valid for positive data. Default False.

continues on next page

Table 227 – continued from previous page

<code>score</code>	Returns a numerical score indicating performance based on the differences between the predicted and actual values.
<code>validate_inputs</code>	Validates the input based on a few simple checks.

**classmethod** `calculate_percent_difference` (*cls*, *score*, *baseline\_score*)

Calculate the percent difference between scores.

#### Parameters

- **score** (*float*) – A score. Output of the score method of this objective.
- **baseline\_score** (*float*) – A score. Output of the score method of this objective. In practice, this is the score achieved on this objective with a baseline estimator.

#### Returns

**The percent difference between the scores. Note that for objectives that can be interpreted as percentages, this will be the difference between the reference score and score. For all other objectives, the difference will be normalized by the reference score.**

**Return type** `float`

**classmethod** `is_defined_for_problem_type` (*cls*, *problem\_type*)

**objective\_function** (*self*, *y\_true*, *y\_predicted*, *X=None*, *sample\_weight=None*)

Computes the relative value of the provided predictions compared to the actual labels, according a specified metric

**Arguments:** *y\_predicted* (`pd.Series`): Predicted values of length [*n\_samples*] *y\_true* (`pd.Series`): Actual class labels of length [*n\_samples*] *X* (`pd.DataFrame` or `np.ndarray`): Extra data of shape [*n\_samples*, *n\_features*] necessary to calculate score *sample\_weight* (`pd.DataFrame` or `np.ndarray`): Sample weights used in computing objective value result

**Returns** Numerical value used to calculate score

**positive\_only** (*cls*)

If True, this objective is only valid for positive data. Default False.

**score** (*self*, *y\_true*, *y\_predicted*, *X=None*, *sample\_weight=None*)

Returns a numerical score indicating performance based on the differences between the predicted and actual values.

#### Parameters

- **y\_predicted** (`pd.Series`) – Predicted values of length [*n\_samples*]
- **y\_true** (`pd.Series`) – Actual class labels of length [*n\_samples*]
- **X** (`pd.DataFrame` or `np.ndarray`) – Extra data of shape [*n\_samples*, *n\_features*] necessary to calculate score
- **sample\_weight** (`pd.DataFrame` or `np.ndarray`) – Sample weights used in computing objective value result

**Returns** `score`

**validate\_inputs** (*self*, *y\_true*, *y\_predicted*)

Validates the input based on a few simple checks.

#### Parameters

- **y\_predicted** (*pd.Series*, or *pd.DataFrame*) – Predicted values of length [n\_samples]
- **y\_true** (*pd.Series*) – Actual class labels of length [n\_samples]

**Returns** None

**class** evalml.objectives.standard\_metrics.**MeanSquaredLogError**

Mean squared log error for regression.

Only valid for nonnegative inputs. Otherwise, will throw a ValueError

**Attributes**

<b>greater_is_better</b>	False
<b>is_bounded_like_percentage</b>	False
<b>name</b>	Mean Squared Log Error
<b>perfect_score</b>	0.0
<b>problem_types</b>	[ProblemTypes.REGRESSION, ProblemTypes.TIME_SERIES_REGRESSION]
<b>score_needs_positive</b>	True

**Methods**

<i>calculate_percent_difference</i>	Calculate the percent difference between scores.
<i>is_defined_for_problem_type</i>	
<i>objective_function</i>	Computes the relative value of the provided predictions compared to the actual labels, according a specified metric
<i>positive_only</i>	If True, this objective is only valid for positive data. Default False.
<i>score</i>	Returns a numerical score indicating performance based on the differences between the predicted and actual values.
<i>validate_inputs</i>	Validates the input based on a few simple checks.

**classmethod** **calculate\_percent\_difference** (*cls*, *score*, *baseline\_score*)

Calculate the percent difference between scores.

**Parameters**

- **score** (*float*) – A score. Output of the score method of this objective.
- **baseline\_score** (*float*) – A score. Output of the score method of this objective. In practice, this is the score achieved on this objective with a baseline estimator.

**Returns**

**The percent difference between the scores. Note that for objectives that can be interpreted as percentages, this will be the difference between the reference score and score. For all other objectives, the difference will be normalized by the reference score.**

**Return type** float

**classmethod** **is\_defined\_for\_problem\_type** (*cls*, *problem\_type*)

**objective\_function** (*self*, *y\_true*, *y\_predicted*, *X=None*, *sample\_weight=None*)

Computes the relative value of the provided predictions compared to the actual labels, according a specified metric

**Arguments:** *y\_predicted* (*pd.Series*): Predicted values of length [*n\_samples*] *y\_true* (*pd.Series*): Actual class labels of length [*n\_samples*] *X* (*pd.DataFrame* or *np.ndarray*): Extra data of shape [*n\_samples*, *n\_features*] necessary to calculate score *sample\_weight* (*pd.DataFrame* or *np.ndarray*): Sample weights used in computing objective value result

**Returns** Numerical value used to calculate score

**positive\_only** (*self*)

If True, this objective is only valid for positive data. Default False.

**score** (*self*, *y\_true*, *y\_predicted*, *X=None*, *sample\_weight=None*)

Returns a numerical score indicating performance based on the differences between the predicted and actual values.

**Parameters**

- **y\_predicted** (*pd.Series*) – Predicted values of length [*n\_samples*]
- **y\_true** (*pd.Series*) – Actual class labels of length [*n\_samples*]
- **X** (*pd.DataFrame* or *np.ndarray*) – Extra data of shape [*n\_samples*, *n\_features*] necessary to calculate score
- **sample\_weight** (*pd.DataFrame* or *np.ndarray*) – Sample weights used in computing objective value result

**Returns** score

**validate\_inputs** (*self*, *y\_true*, *y\_predicted*)

Validates the input based on a few simple checks.

**Parameters**

- **y\_predicted** (*pd.Series*, or *pd.DataFrame*) – Predicted values of length [*n\_samples*]
- **y\_true** (*pd.Series*) – Actual class labels of length [*n\_samples*]

**Returns** None

**class** evalml.objectives.standard\_metrics.**MedianAE**

Median absolute error for regression.

**Attributes**

<b>greater_is_better</b>	False
<b>is_bounded_like_percentage</b>	False
<b>name</b>	MedianAE
<b>perfect_score</b>	0.0
<b>problem_types</b>	[ProblemTypes.REGRESSION, ProblemTypes.TIME_SERIES_REGRESSION]
<b>score_needs_problem</b>	True

**Methods**

<code>calculate_percent_difference</code>	Calculate the percent difference between scores.
<code>is_defined_for_problem_type</code>	
<code>objective_function</code>	Computes the relative value of the provided predictions compared to the actual labels, according a specified metric
<code>positive_only</code>	If True, this objective is only valid for positive data. Default False.
<code>score</code>	Returns a numerical score indicating performance based on the differences between the predicted and actual values.
<code>validate_inputs</code>	Validates the input based on a few simple checks.

**classmethod** `calculate_percent_difference` (*cls*, *score*, *baseline\_score*)

Calculate the percent difference between scores.

#### Parameters

- **score** (*float*) – A score. Output of the score method of this objective.
- **baseline\_score** (*float*) – A score. Output of the score method of this objective. In practice, this is the score achieved on this objective with a baseline estimator.

#### Returns

**The percent difference between the scores. Note that for objectives that can be interpreted** as percentages, this will be the difference between the reference score and score. For all other objectives, the difference will be normalized by the reference score.

**Return type** float

**classmethod** `is_defined_for_problem_type` (*cls*, *problem\_type*)

**objective\_function** (*self*, *y\_true*, *y\_predicted*, *X=None*, *sample\_weight=None*)

Computes the relative value of the provided predictions compared to the actual labels, according a specified metric

**Arguments:** *y\_predicted* (*pd.Series*): Predicted values of length [*n\_samples*] *y\_true* (*pd.Series*): Actual class labels of length [*n\_samples*] *X* (*pd.DataFrame* or *np.ndarray*): Extra data of shape [*n\_samples*, *n\_features*] necessary to calculate score *sample\_weight* (*pd.DataFrame* or *np.ndarray*): Sample weights used in computing objective value result

**Returns** Numerical value used to calculate score

**positive\_only** (*cls*)

If True, this objective is only valid for positive data. Default False.

**score** (*self*, *y\_true*, *y\_predicted*, *X=None*, *sample\_weight=None*)

Returns a numerical score indicating performance based on the differences between the predicted and actual values.

#### Parameters

- **y\_predicted** (*pd.Series*) – Predicted values of length [*n\_samples*]
- **y\_true** (*pd.Series*) – Actual class labels of length [*n\_samples*]
- **X** (*pd.DataFrame* or *np.ndarray*) – Extra data of shape [*n\_samples*, *n\_features*] necessary to calculate score



- **sample\_weight** (*pd.DataFrame* or *np.ndarray*) – Sample weights used in computing objective value result

**Returns** score

**validate\_inputs** (*self*, *y\_true*, *y\_predicted*)

Validates the input based on a few simple checks.

**Parameters**

- **y\_predicted** (*pd.Series*, or *pd.DataFrame*) – Predicted values of length [n\_samples]
- **y\_true** (*pd.Series*) – Actual class labels of length [n\_samples]

**Returns** None

**class** evalml.objectives.standard\_metrics.**MSE**

Mean squared error for regression.

**Attributes**

<b>greater_is_better</b>	False
<b>is_bounded_like_percentage</b>	False
<b>name</b>	MSE
<b>perfect_score</b>	0.0
<b>problem_types</b>	[ProblemTypes.REGRESSION, ProblemTypes.TIME_SERIES_REGRESSION]
<b>score_needs_proba</b>	False

**Methods**

<i>calculate_percent_difference</i>	Calculate the percent difference between scores.
<i>is_defined_for_problem_type</i>	
<i>objective_function</i>	Computes the relative value of the provided predictions compared to the actual labels, according a specified metric
<i>positive_only</i>	If True, this objective is only valid for positive data. Default False.
<i>score</i>	Returns a numerical score indicating performance based on the differences between the predicted and actual values.
<i>validate_inputs</i>	Validates the input based on a few simple checks.

**classmethod** **calculate\_percent\_difference** (*cls*, *score*, *baseline\_score*)

Calculate the percent difference between scores.

**Parameters**

- **score** (*float*) – A score. Output of the score method of this objective.
- **baseline\_score** (*float*) – A score. Output of the score method of this objective. In practice, this is the score achieved on this objective with a baseline estimator.

**Returns**

The percent difference between the scores. Note that for objectives that can be interpreted

as percentages, this will be the difference between the reference score and score. For all other objectives, the difference will be normalized by the reference score.

**Return type** float

**classmethod** `is_defined_for_problem_type(cls, problem_type)`

**objective\_function** (*self*, *y\_true*, *y\_predicted*, *X=None*, *sample\_weight=None*)

Computes the relative value of the provided predictions compared to the actual labels, according a specified metric

**Arguments:** *y\_predicted* (pd.Series): Predicted values of length [n\_samples] *y\_true* (pd.Series): Actual class labels of length [n\_samples] *X* (pd.DataFrame or np.ndarray): Extra data of shape [n\_samples, n\_features] necessary to calculate score *sample\_weight* (pd.DataFrame or np.ndarray): Sample weights used in computing objective value result

**Returns** Numerical value used to calculate score

**positive\_only** (*cls*)

If True, this objective is only valid for positive data. Default False.

**score** (*self*, *y\_true*, *y\_predicted*, *X=None*, *sample\_weight=None*)

Returns a numerical score indicating performance based on the differences between the predicted and actual values.

**Parameters**

- **y\_predicted** (pd.Series) – Predicted values of length [n\_samples]
- **y\_true** (pd.Series) – Actual class labels of length [n\_samples]
- **X** (pd.DataFrame or np.ndarray) – Extra data of shape [n\_samples, n\_features] necessary to calculate score
- **sample\_weight** (pd.DataFrame or np.ndarray) – Sample weights used in computing objective value result

**Returns** score

**validate\_inputs** (*self*, *y\_true*, *y\_predicted*)

Validates the input based on a few simple checks.

**Parameters**

- **y\_predicted** (pd.Series, or pd.DataFrame) – Predicted values of length [n\_samples]
- **y\_true** (pd.Series) – Actual class labels of length [n\_samples]

**Returns** None

**class** evalml.objectives.standard\_metrics.Precision

Precision score for binary classification.

**Attributes**

<b>greater_is_better</b>	True
<b>is_bounded_like_percentage</b>	True
<b>name</b>	Precision
<b>perfect_score</b>	1.0
<b>problem_types</b>	[ProblemTypes.BINARY, ProblemTypes.TIME_SERIES_BINARY]
<b>score_needs_proba</b>	True

## Methods

<code>calculate_percent_difference</code>	Calculate the percent difference between scores.
<code>can_optimize_threshold</code>	Returns a boolean determining if we can optimize the binary classification objective threshold.
<code>decision_function</code>	Apply a learned threshold to predicted probabilities to get predicted classes.
<code>is_defined_for_problem_type</code>	
<code>objective_function</code>	Computes the relative value of the provided predictions compared to the actual labels, according a specified metric
<code>optimize_threshold</code>	Learn a binary classification threshold which optimizes the current objective.
<code>positive_only</code>	If True, this objective is only valid for positive data. Default False.
<code>score</code>	Returns a numerical score indicating performance based on the differences between the predicted and actual values.
<code>validate_inputs</code>	Validates the input based on a few simple checks.

**classmethod** `calculate_percent_difference` (*cls*, *score*, *baseline\_score*)

Calculate the percent difference between scores.

### Parameters

- **score** (*float*) – A score. Output of the score method of this objective.
- **baseline\_score** (*float*) – A score. Output of the score method of this objective. In practice, this is the score achieved on this objective with a baseline estimator.

### Returns

**The percent difference between the scores. Note that for objectives that can be interpreted as percentages, this will be the difference between the reference score and score. For all other objectives, the difference will be normalized by the reference score.**

**Return type** float

**property** `can_optimize_threshold` (*cls*)

Returns a boolean determining if we can optimize the binary classification objective threshold. This will be false for any objective that works directly with predicted probabilities, like log loss and AUC. Otherwise, it will be true.

**decision\_function** (*self*, *ypred\_proba*, *threshold=0.5*, *X=None*)

Apply a learned threshold to predicted probabilities to get predicted classes.

**Parameters**

- **ypred\_proba** (*pd.Series, np.ndarray*) – The classifier’s predicted probabilities
- **threshold** (*float, optional*) – Threshold used to make a prediction. Defaults to 0.5.
- **X** (*pd.DataFrame, optional*) – Any extra columns that are needed from training data.

**Returns** predictions

**classmethod is\_defined\_for\_problem\_type** (*cls, problem\_type*)

**objective\_function** (*self, y\_true, y\_predicted, X=None, sample\_weight=None*)

Computes the relative value of the provided predictions compared to the actual labels, according a specified metric

**Arguments:** *y\_predicted* (*pd.Series*): Predicted values of length [*n\_samples*] *y\_true* (*pd.Series*): Actual class labels of length [*n\_samples*] *X* (*pd.DataFrame* or *np.ndarray*): Extra data of shape [*n\_samples*, *n\_features*] necessary to calculate score *sample\_weight* (*pd.DataFrame* or *np.ndarray*): Sample weights used in computing objective value result

**Returns** Numerical value used to calculate score

**optimize\_threshold** (*self, ypred\_proba, y\_true, X=None*)

Learn a binary classification threshold which optimizes the current objective.

**Parameters**

- **ypred\_proba** (*pd.Series*) – The classifier’s predicted probabilities
- **y\_true** (*pd.Series*) – The ground truth for the predictions.
- **X** (*pd.DataFrame, optional*) – Any extra columns that are needed from training data.

**Returns** Optimal threshold for this objective

**positive\_only** (*cls*)

If True, this objective is only valid for positive data. Default False.

**score** (*self, y\_true, y\_predicted, X=None, sample\_weight=None*)

Returns a numerical score indicating performance based on the differences between the predicted and actual values.

**Parameters**

- **y\_predicted** (*pd.Series*) – Predicted values of length [*n\_samples*]
- **y\_true** (*pd.Series*) – Actual class labels of length [*n\_samples*]
- **X** (*pd.DataFrame* or *np.ndarray*) – Extra data of shape [*n\_samples*, *n\_features*] necessary to calculate score
- **sample\_weight** (*pd.DataFrame* or *np.ndarray*) – Sample weights used in computing objective value result

**Returns** score

**validate\_inputs** (*self, y\_true, y\_predicted*)

Validates the input based on a few simple checks.

**Parameters**

- **y\_predicted** (*pd.Series*, or *pd.DataFrame*) – Predicted values of length [n\_samples]
- **y\_true** (*pd.Series*) – Actual class labels of length [n\_samples]

Returns None

**class** evalml.objectives.standard\_metrics.**PrecisionMacro**

Precision score for multiclass classification using macro averaging.

Attributes

<b>greater_is_better</b>	True
<b>is_bounded_like_percentage</b>	True
<b>name</b>	Precision Macro
<b>perfect_score</b>	1.0
<b>problem_types</b>	[ProblemTypes.MULTICLASS, ProblemTypes.TIME_SERIES_MULTICLASS]
<b>score_needs_proba</b>	True

Methods

<i>calculate_percent_difference</i>	Calculate the percent difference between scores.
<i>is_defined_for_problem_type</i>	
<i>objective_function</i>	Computes the relative value of the provided predictions compared to the actual labels, according a specified metric
<i>positive_only</i>	If True, this objective is only valid for positive data. Default False.
<i>score</i>	Returns a numerical score indicating performance based on the differences between the predicted and actual values.
<i>validate_inputs</i>	Validates the input based on a few simple checks.

**classmethod** **calculate\_percent\_difference** (*cls*, *score*, *baseline\_score*)

Calculate the percent difference between scores.

Parameters

- **score** (*float*) – A score. Output of the score method of this objective.
- **baseline\_score** (*float*) – A score. Output of the score method of this objective. In practice, this is the score achieved on this objective with a baseline estimator.

Returns

**The percent difference between the scores. Note that for objectives that can be interpreted as percentages, this will be the difference between the reference score and score. For all other objectives, the difference will be normalized by the reference score.**

Return type float

**classmethod** **is\_defined\_for\_problem\_type** (*cls*, *problem\_type*)

**objective\_function** (*self*, *y\_true*, *y\_predicted*, *X=None*, *sample\_weight=None*)

Computes the relative value of the provided predictions compared to the actual labels, according a specified metric

**Arguments:** `y_predicted` (`pd.Series`): Predicted values of length `[n_samples]` `y_true` (`pd.Series`): Actual class labels of length `[n_samples]` `X` (`pd.DataFrame` or `np.ndarray`): Extra data of shape `[n_samples, n_features]` necessary to calculate score `sample_weight` (`pd.DataFrame` or `np.ndarray`): Sample weights used in computing objective value result

**Returns** Numerical value used to calculate score

**positive\_only** (*cls*)

If True, this objective is only valid for positive data. Default False.

**score** (*self*, *y\_true*, *y\_predicted*, *X=None*, *sample\_weight=None*)

Returns a numerical score indicating performance based on the differences between the predicted and actual values.

**Parameters**

- **y\_predicted** (*pd.Series*) – Predicted values of length `[n_samples]`
- **y\_true** (*pd.Series*) – Actual class labels of length `[n_samples]`
- **X** (*pd.DataFrame* or *np.ndarray*) – Extra data of shape `[n_samples, n_features]` necessary to calculate score
- **sample\_weight** (*pd.DataFrame* or *np.ndarray*) – Sample weights used in computing objective value result

**Returns** score

**validate\_inputs** (*self*, *y\_true*, *y\_predicted*)

Validates the input based on a few simple checks.

**Parameters**

- **y\_predicted** (*pd.Series*, or *pd.DataFrame*) – Predicted values of length `[n_samples]`
- **y\_true** (*pd.Series*) – Actual class labels of length `[n_samples]`

**Returns** None

**class** `evalml.objectives.standard_metrics.PrecisionMicro`

Precision score for multiclass classification using micro averaging.

**Attributes**

<b>greater_is_better</b>	True
<b>is_bounded_like_percentage</b>	True
<b>name</b>	Precision Micro
<b>perfect_score</b>	1.0
<b>problem_types</b>	[ProblemTypes.MULTICLASS, ProblemTypes.TIME_SERIES_MULTICLASS]
<b>score_needs_proba</b>	False

**Methods**

<code>calculate_percent_difference</code>	Calculate the percent difference between scores.
<code>is_defined_for_problem_type</code>	

---

continues on next page

Table 233 – continued from previous page

<i>objective_function</i>	Computes the relative value of the provided predictions compared to the actual labels, according a specified metric
<i>positive_only</i>	If True, this objective is only valid for positive data. Default False.
<i>score</i>	Returns a numerical score indicating performance based on the differences between the predicted and actual values.
<i>validate_inputs</i>	Validates the input based on a few simple checks.

**classmethod** `calculate_percent_difference` (*cls*, *score*, *baseline\_score*)

Calculate the percent difference between scores.

#### Parameters

- **score** (*float*) – A score. Output of the score method of this objective.
- **baseline\_score** (*float*) – A score. Output of the score method of this objective. In practice, this is the score achieved on this objective with a baseline estimator.

#### Returns

**The percent difference between the scores. Note that for objectives that can be interpreted** as percentages, this will be the difference between the reference score and score. For all other objectives, the difference will be normalized by the reference score.

**Return type** float

**classmethod** `is_defined_for_problem_type` (*cls*, *problem\_type*)

**objective\_function** (*self*, *y\_true*, *y\_predicted*, *X=None*, *sample\_weight=None*)

Computes the relative value of the provided predictions compared to the actual labels, according a specified metric

**Arguments:** *y\_predicted* (*pd.Series*): Predicted values of length [*n\_samples*] *y\_true* (*pd.Series*): Actual class labels of length [*n\_samples*] *X* (*pd.DataFrame* or *np.ndarray*): Extra data of shape [*n\_samples*, *n\_features*] necessary to calculate score *sample\_weight* (*pd.DataFrame* or *np.ndarray*): Sample weights used in computing objective value result

**Returns** Numerical value used to calculate score

**positive\_only** (*cls*)

If True, this objective is only valid for positive data. Default False.

**score** (*self*, *y\_true*, *y\_predicted*, *X=None*, *sample\_weight=None*)

Returns a numerical score indicating performance based on the differences between the predicted and actual values.

#### Parameters

- **y\_predicted** (*pd.Series*) – Predicted values of length [*n\_samples*]
- **y\_true** (*pd.Series*) – Actual class labels of length [*n\_samples*]
- **X** (*pd.DataFrame* or *np.ndarray*) – Extra data of shape [*n\_samples*, *n\_features*] necessary to calculate score
- **sample\_weight** (*pd.DataFrame* or *np.ndarray*) – Sample weights used in computing objective value result

**Returns** score

**validate\_inputs** (*self*, *y\_true*, *y\_predicted*)

Validates the input based on a few simple checks.

#### Parameters

- **y\_predicted** (*pd.Series*, or *pd.DataFrame*) – Predicted values of length [n\_samples]
- **y\_true** (*pd.Series*) – Actual class labels of length [n\_samples]

**Returns** None

**class** evalml.objectives.standard\_metrics.**PrecisionWeighted**

Precision score for multiclass classification using weighted averaging.

#### Attributes

<b>greater_is_better</b>	True
<b>is_bounded_like_percentage</b>	True
<b>name</b>	Precision Weighted
<b>perfect_score</b>	1.0
<b>problem_types</b>	[ProblemTypes.MULTICLASS, ProblemTypes.TIME_SERIES_MULTICLASS]
<b>score_needs_proba</b>	True

#### Methods

<i>calculate_percent_difference</i>	Calculate the percent difference between scores.
<i>is_defined_for_problem_type</i>	
<i>objective_function</i>	Computes the relative value of the provided predictions compared to the actual labels, according a specified metric
<i>positive_only</i>	If True, this objective is only valid for positive data. Default False.
<i>score</i>	Returns a numerical score indicating performance based on the differences between the predicted and actual values.
<i>validate_inputs</i>	Validates the input based on a few simple checks.

**classmethod** **calculate\_percent\_difference** (*cls*, *score*, *baseline\_score*)

Calculate the percent difference between scores.

#### Parameters

- **score** (*float*) – A score. Output of the score method of this objective.
- **baseline\_score** (*float*) – A score. Output of the score method of this objective. In practice, this is the score achieved on this objective with a baseline estimator.

#### Returns

**The percent difference between the scores. Note that for objectives that can be interpreted as percentages, this will be the difference between the reference score and score. For all other objectives, the difference will be normalized by the reference score.**

**Return type** float



**classmethod** `is_defined_for_problem_type(cls, problem_type)`

**objective\_function** (*self*, *y\_true*, *y\_predicted*, *X=None*, *sample\_weight=None*)

Computes the relative value of the provided predictions compared to the actual labels, according a specified metric

**Arguments:** *y\_predicted* (pd.Series): Predicted values of length [n\_samples] *y\_true* (pd.Series): Actual class labels of length [n\_samples] *X* (pd.DataFrame or np.ndarray): Extra data of shape [n\_samples, n\_features] necessary to calculate score *sample\_weight* (pd.DataFrame or np.ndarray): Sample weights used in computing objective value result

**Returns** Numerical value used to calculate score

**positive\_only** (*cls*)

If True, this objective is only valid for positive data. Default False.

**score** (*self*, *y\_true*, *y\_predicted*, *X=None*, *sample\_weight=None*)

Returns a numerical score indicating performance based on the differences between the predicted and actual values.

**Parameters**

- **y\_predicted** (pd.Series) – Predicted values of length [n\_samples]
- **y\_true** (pd.Series) – Actual class labels of length [n\_samples]
- **X** (pd.DataFrame or np.ndarray) – Extra data of shape [n\_samples, n\_features] necessary to calculate score
- **sample\_weight** (pd.DataFrame or np.ndarray) – Sample weights used in computing objective value result

**Returns** score

**validate\_inputs** (*self*, *y\_true*, *y\_predicted*)

Validates the input based on a few simple checks.

**Parameters**

- **y\_predicted** (pd.Series, or pd.DataFrame) – Predicted values of length [n\_samples]
- **y\_true** (pd.Series) – Actual class labels of length [n\_samples]

**Returns** None

**class** evalml.objectives.standard\_metrics.**R2**

Coefficient of determination for regression.

**Attributes**

<b>greater_is_better</b>	True
<b>is_bounded_like_percentage</b>	False
<b>name</b>	R2
<b>perfect_score</b>	1
<b>problem_types</b>	[ProblemTypes.REGRESSION, ProblemTypes.TIME_SERIES_REGRESSION]
<b>score_needs_proba</b>	False

**Methods**

<code>calculate_percent_difference</code>	Calculate the percent difference between scores.
<code>is_defined_for_problem_type</code>	
<code>objective_function</code>	Computes the relative value of the provided predictions compared to the actual labels, according a specified metric
<code>positive_only</code>	If True, this objective is only valid for positive data. Default False.
<code>score</code>	Returns a numerical score indicating performance based on the differences between the predicted and actual values.
<code>validate_inputs</code>	Validates the input based on a few simple checks.

**classmethod** `calculate_percent_difference` (*cls*, *score*, *baseline\_score*)

Calculate the percent difference between scores.

#### Parameters

- **score** (*float*) – A score. Output of the score method of this objective.
- **baseline\_score** (*float*) – A score. Output of the score method of this objective. In practice, this is the score achieved on this objective with a baseline estimator.

#### Returns

**The percent difference between the scores. Note that for objectives that can be interpreted** as percentages, this will be the difference between the reference score and score. For all other objectives, the difference will be normalized by the reference score.

**Return type** float

**classmethod** `is_defined_for_problem_type` (*cls*, *problem\_type*)

**objective\_function** (*self*, *y\_true*, *y\_predicted*, *X=None*, *sample\_weight=None*)

Computes the relative value of the provided predictions compared to the actual labels, according a specified metric

**Arguments:** *y\_predicted* (*pd.Series*): Predicted values of length [*n\_samples*] *y\_true* (*pd.Series*): Actual class labels of length [*n\_samples*] *X* (*pd.DataFrame* or *np.ndarray*): Extra data of shape [*n\_samples*, *n\_features*] necessary to calculate score *sample\_weight* (*pd.DataFrame* or *np.ndarray*): Sample weights used in computing objective value result

**Returns** Numerical value used to calculate score

**positive\_only** (*cls*)

If True, this objective is only valid for positive data. Default False.

**score** (*self*, *y\_true*, *y\_predicted*, *X=None*, *sample\_weight=None*)

Returns a numerical score indicating performance based on the differences between the predicted and actual values.

#### Parameters

- **y\_predicted** (*pd.Series*) – Predicted values of length [*n\_samples*]
- **y\_true** (*pd.Series*) – Actual class labels of length [*n\_samples*]
- **X** (*pd.DataFrame* or *np.ndarray*) – Extra data of shape [*n\_samples*, *n\_features*] necessary to calculate score

- **sample\_weight** (*pd.DataFrame* or *np.ndarray*) – Sample weights used in computing objective value result

**Returns** score

**validate\_inputs** (*self*, *y\_true*, *y\_predicted*)

Validates the input based on a few simple checks.

**Parameters**

- **y\_predicted** (*pd.Series*, or *pd.DataFrame*) – Predicted values of length [n\_samples]
- **y\_true** (*pd.Series*) – Actual class labels of length [n\_samples]

**Returns** None

**class** evalml.objectives.standard\_metrics.**Recall**

Recall score for binary classification.

**Attributes**

<b>greater_is_better</b>	True
<b>is_bounded_like_percentage</b>	True
<b>name</b>	Recall
<b>perfect_score</b>	1.0
<b>problem_types</b>	[ProblemTypes.BINARY, ProblemTypes.TIME_SERIES_BINARY]
<b>score_needs_proba</b>	False

**Methods**

<i>calculate_percent_difference</i>	Calculate the percent difference between scores.
<i>can_optimize_threshold</i>	Returns a boolean determining if we can optimize the binary classification objective threshold.
<i>decision_function</i>	Apply a learned threshold to predicted probabilities to get predicted classes.
<i>is_defined_for_problem_type</i>	
<i>objective_function</i>	Computes the relative value of the provided predictions compared to the actual labels, according a specified metric
<i>optimize_threshold</i>	Learn a binary classification threshold which optimizes the current objective.
<i>positive_only</i>	If True, this objective is only valid for positive data. Default False.
<i>score</i>	Returns a numerical score indicating performance based on the differences between the predicted and actual values.
<i>validate_inputs</i>	Validates the input based on a few simple checks.

**classmethod** **calculate\_percent\_difference** (*cls*, *score*, *baseline\_score*)

Calculate the percent difference between scores.

**Parameters**

- **score** (*float*) – A score. Output of the score method of this objective.
- **baseline\_score** (*float*) – A score. Output of the score method of this objective. In practice, this is the score achieved on this objective with a baseline estimator.

**Returns**

**The percent difference between the scores. Note that for objectives that can be interpreted** as percentages, this will be the difference between the reference score and score. For all other objectives, the difference will be normalized by the reference score.

**Return type** float

**property can\_optimize\_threshold** (*cls*)

Returns a boolean determining if we can optimize the binary classification objective threshold. This will be false for any objective that works directly with predicted probabilities, like log loss and AUC. Otherwise, it will be true.

**decision\_function** (*self, ypred\_proba, threshold=0.5, X=None*)

Apply a learned threshold to predicted probabilities to get predicted classes.

**Parameters**

- **ypred\_proba** (*pd.Series, np.ndarray*) – The classifier’s predicted probabilities
- **threshold** (*float, optional*) – Threshold used to make a prediction. Defaults to 0.5.
- **X** (*pd.DataFrame, optional*) – Any extra columns that are needed from training data.

**Returns** predictions

**classmethod is\_defined\_for\_problem\_type** (*cls, problem\_type*)

**objective\_function** (*self, y\_true, y\_predicted, X=None, sample\_weight=None*)

Computes the relative value of the provided predictions compared to the actual labels, according a specified metric

**Arguments:** *y\_predicted* (*pd.Series*): Predicted values of length *[n\_samples]* *y\_true* (*pd.Series*): Actual class labels of length *[n\_samples]* *X* (*pd.DataFrame* or *np.ndarray*): Extra data of shape *[n\_samples, n\_features]* necessary to calculate score *sample\_weight* (*pd.DataFrame* or *np.ndarray*): Sample weights used in computing objective value result

**Returns** Numerical value used to calculate score

**optimize\_threshold** (*self, ypred\_proba, y\_true, X=None*)

Learn a binary classification threshold which optimizes the current objective.

**Parameters**

- **ypred\_proba** (*pd.Series*) – The classifier’s predicted probabilities
- **y\_true** (*pd.Series*) – The ground truth for the predictions.
- **X** (*pd.DataFrame, optional*) – Any extra columns that are needed from training data.

**Returns** Optimal threshold for this objective

**positive\_only** (*cls*)

If True, this objective is only valid for positive data. Default False.

**score** (*self*, *y\_true*, *y\_predicted*, *X=None*, *sample\_weight=None*)

Returns a numerical score indicating performance based on the differences between the predicted and actual values.

#### Parameters

- **y\_predicted** (*pd.Series*) – Predicted values of length [n\_samples]
- **y\_true** (*pd.Series*) – Actual class labels of length [n\_samples]
- **X** (*pd.DataFrame* or *np.ndarray*) – Extra data of shape [n\_samples, n\_features] necessary to calculate score
- **sample\_weight** (*pd.DataFrame* or *np.ndarray*) – Sample weights used in computing objective value result

**Returns** score

**validate\_inputs** (*self*, *y\_true*, *y\_predicted*)

Validates the input based on a few simple checks.

#### Parameters

- **y\_predicted** (*pd.Series*, or *pd.DataFrame*) – Predicted values of length [n\_samples]
- **y\_true** (*pd.Series*) – Actual class labels of length [n\_samples]

**Returns** None

**class** evalml.objectives.standard\_metrics.**RecallMacro**

Recall score for multiclass classification using macro averaging.

#### Attributes

<b>greater_is_better</b>	True
<b>is_bounded_like_percentage</b>	True
<b>name</b>	Recall Macro
<b>perfect_score</b>	1.0
<b>problem_types</b>	[ProblemTypes.MULTICLASS, ProblemTypes.TIME_SERIES_MULTICLASS]
<b>score_needs_proba</b>	False

#### Methods

<i>calculate_percent_difference</i>	Calculate the percent difference between scores.
<i>is_defined_for_problem_type</i>	
<i>objective_function</i>	Computes the relative value of the provided predictions compared to the actual labels, according a specified metric
<i>positive_only</i>	If True, this objective is only valid for positive data. Default False.
<i>score</i>	Returns a numerical score indicating performance based on the differences between the predicted and actual values.
<i>validate_inputs</i>	Validates the input based on a few simple checks.

**classmethod** `calculate_percent_difference` (*cls*, *score*, *baseline\_score*)

Calculate the percent difference between scores.

**Parameters**

- **score** (*float*) – A score. Output of the score method of this objective.
- **baseline\_score** (*float*) – A score. Output of the score method of this objective. In practice, this is the score achieved on this objective with a baseline estimator.

**Returns**

**The percent difference between the scores. Note that for objectives that can be interpreted as percentages, this will be the difference between the reference score and score. For all other objectives, the difference will be normalized by the reference score.**

**Return type** `float`

**classmethod** `is_defined_for_problem_type` (*cls*, *problem\_type*)

**objective\_function** (*self*, *y\_true*, *y\_predicted*, *X=None*, *sample\_weight=None*)

Computes the relative value of the provided predictions compared to the actual labels, according a specified metric

**Arguments:** *y\_predicted* (`pd.Series`): Predicted values of length [*n\_samples*] *y\_true* (`pd.Series`): Actual class labels of length [*n\_samples*] *X* (`pd.DataFrame` or `np.ndarray`): Extra data of shape [*n\_samples*, *n\_features*] necessary to calculate score *sample\_weight* (`pd.DataFrame` or `np.ndarray`): Sample weights used in computing objective value result

**Returns** Numerical value used to calculate score

**positive\_only** (*cls*)

If True, this objective is only valid for positive data. Default False.

**score** (*self*, *y\_true*, *y\_predicted*, *X=None*, *sample\_weight=None*)

Returns a numerical score indicating performance based on the differences between the predicted and actual values.

**Parameters**

- **y\_predicted** (`pd.Series`) – Predicted values of length [*n\_samples*]
- **y\_true** (`pd.Series`) – Actual class labels of length [*n\_samples*]
- **X** (`pd.DataFrame` or `np.ndarray`) – Extra data of shape [*n\_samples*, *n\_features*] necessary to calculate score
- **sample\_weight** (`pd.DataFrame` or `np.ndarray`) – Sample weights used in computing objective value result

**Returns** `score`

**validate\_inputs** (*self*, *y\_true*, *y\_predicted*)

Validates the input based on a few simple checks.

**Parameters**

- **y\_predicted** (`pd.Series`, or `pd.DataFrame`) – Predicted values of length [*n\_samples*]
- **y\_true** (`pd.Series`) – Actual class labels of length [*n\_samples*]

**Returns** `None`

**class** evalml.objectives.standard\_metrics.RecallMicro

Recall score for multiclass classification using micro averaging.

#### Attributes

<b>greater_is_better</b>	True
<b>is_bounded_like_percentage</b>	True
<b>name</b>	Recall Micro
<b>perfect_score</b>	1.0
<b>problem_types</b>	[ProblemTypes.MULTICLASS, ProblemTypes.TIME_SERIES_MULTICLASS]
<b>score_needs_proba</b>	False

#### Methods

<i>calculate_percent_difference</i>	Calculate the percent difference between scores.
<i>is_defined_for_problem_type</i>	
<i>objective_function</i>	Computes the relative value of the provided predictions compared to the actual labels, according a specified metric
<i>positive_only</i>	If True, this objective is only valid for positive data. Default False.
<i>score</i>	Returns a numerical score indicating performance based on the differences between the predicted and actual values.
<i>validate_inputs</i>	Validates the input based on a few simple checks.

**classmethod** *calculate\_percent\_difference* (cls, score, baseline\_score)

Calculate the percent difference between scores.

#### Parameters

- **score** (*float*) – A score. Output of the score method of this objective.
- **baseline\_score** (*float*) – A score. Output of the score method of this objective. In practice, this is the score achieved on this objective with a baseline estimator.

#### Returns

**The percent difference between the scores. Note that for objectives that can be interpreted as percentages, this will be the difference between the reference score and score. For all other objectives, the difference will be normalized by the reference score.**

**Return type** float

**classmethod** *is\_defined\_for\_problem\_type* (cls, problem\_type)

**objective\_function** (self, y\_true, y\_predicted, X=None, sample\_weight=None)

Computes the relative value of the provided predictions compared to the actual labels, according a specified metric

**Arguments:** y\_predicted (pd.Series): Predicted values of length [n\_samples] y\_true (pd.Series): Actual class labels of length [n\_samples] X (pd.DataFrame or np.ndarray): Extra data of shape [n\_samples, n\_features] necessary to calculate score sample\_weight (pd.DataFrame or np.ndarray): Sample weights used in computing objective value result

**Returns** Numerical value used to calculate score

**positive\_only** (*cls*)

If True, this objective is only valid for positive data. Default False.

**score** (*self*, *y\_true*, *y\_predicted*, *X=None*, *sample\_weight=None*)

Returns a numerical score indicating performance based on the differences between the predicted and actual values.

**Parameters**

- **y\_predicted** (*pd.Series*) – Predicted values of length [n\_samples]
- **y\_true** (*pd.Series*) – Actual class labels of length [n\_samples]
- **X** (*pd.DataFrame* or *np.ndarray*) – Extra data of shape [n\_samples, n\_features] necessary to calculate score
- **sample\_weight** (*pd.DataFrame* or *np.ndarray*) – Sample weights used in computing objective value result

**Returns** score

**validate\_inputs** (*self*, *y\_true*, *y\_predicted*)

Validates the input based on a few simple checks.

**Parameters**

- **y\_predicted** (*pd.Series*, or *pd.DataFrame*) – Predicted values of length [n\_samples]
- **y\_true** (*pd.Series*) – Actual class labels of length [n\_samples]

**Returns** None

**class** evalml.objectives.standard\_metrics.**RecallWeighted**

Recall score for multiclass classification using weighted averaging.

**Attributes**

<b>greater_is_better</b>	True
<b>is_bounded_like_percentage</b>	True
<b>name</b>	Recall Weighted
<b>perfect_score</b>	1.0
<b>problem_types</b>	[ProblemTypes.MULTICLASS, ProblemTypes.TIME_SERIES_MULTICLASS]
<b>score_needs_proba</b>	False

**Methods**

<code>calculate_percent_difference</code>	Calculate the percent difference between scores.
<code>is_defined_for_problem_type</code>	
<code>objective_function</code>	Computes the relative value of the provided predictions compared to the actual labels, according a specified metric
<code>positive_only</code>	If True, this objective is only valid for positive data. Default False.

continues on next page



Table 239 – continued from previous page

<code>score</code>	Returns a numerical score indicating performance based on the differences between the predicted and actual values.
<code>validate_inputs</code>	Validates the input based on a few simple checks.

**classmethod** `calculate_percent_difference` (*cls*, *score*, *baseline\_score*)

Calculate the percent difference between scores.

#### Parameters

- **score** (*float*) – A score. Output of the score method of this objective.
- **baseline\_score** (*float*) – A score. Output of the score method of this objective. In practice, this is the score achieved on this objective with a baseline estimator.

#### Returns

**The percent difference between the scores. Note that for objectives that can be interpreted as percentages, this will be the difference between the reference score and score. For all other objectives, the difference will be normalized by the reference score.**

**Return type** `float`

**classmethod** `is_defined_for_problem_type` (*cls*, *problem\_type*)

**objective\_function** (*self*, *y\_true*, *y\_predicted*, *X=None*, *sample\_weight=None*)

Computes the relative value of the provided predictions compared to the actual labels, according a specified metric

**Arguments:** *y\_predicted* (*pd.Series*): Predicted values of length [*n\_samples*] *y\_true* (*pd.Series*): Actual class labels of length [*n\_samples*] *X* (*pd.DataFrame* or *np.ndarray*): Extra data of shape [*n\_samples*, *n\_features*] necessary to calculate score *sample\_weight* (*pd.DataFrame* or *np.ndarray*): Sample weights used in computing objective value result

**Returns** Numerical value used to calculate score

**positive\_only** (*cls*)

If True, this objective is only valid for positive data. Default False.

**score** (*self*, *y\_true*, *y\_predicted*, *X=None*, *sample\_weight=None*)

Returns a numerical score indicating performance based on the differences between the predicted and actual values.

#### Parameters

- **y\_predicted** (*pd.Series*) – Predicted values of length [*n\_samples*]
- **y\_true** (*pd.Series*) – Actual class labels of length [*n\_samples*]
- **X** (*pd.DataFrame* or *np.ndarray*) – Extra data of shape [*n\_samples*, *n\_features*] necessary to calculate score
- **sample\_weight** (*pd.DataFrame* or *np.ndarray*) – Sample weights used in computing objective value result

**Returns** `score`

**validate\_inputs** (*self*, *y\_true*, *y\_predicted*)

Validates the input based on a few simple checks.

#### Parameters

- **y\_predicted** (*pd.Series*, or *pd.DataFrame*) – Predicted values of length [n\_samples]
- **y\_true** (*pd.Series*) – Actual class labels of length [n\_samples]

**Returns** None

**class** evalml.objectives.standard\_metrics.**RootMeanSquaredError**

Root mean squared error for regression.

**Attributes**

<b>greater_is_better</b>	False
<b>is_bounded_like_percentage</b>	False
<b>name</b>	Root Mean Squared Error
<b>perfect_score</b>	0.0
<b>problem_types</b>	[ProblemTypes.REGRESSION, ProblemTypes.TIME_SERIES_REGRESSION]
<b>score_needs_proba</b>	False

**Methods**

<i>calculate_percent_difference</i>	Calculate the percent difference between scores.
<i>is_defined_for_problem_type</i>	
<i>objective_function</i>	Computes the relative value of the provided predictions compared to the actual labels, according a specified metric
<i>positive_only</i>	If True, this objective is only valid for positive data. Default False.
<i>score</i>	Returns a numerical score indicating performance based on the differences between the predicted and actual values.
<i>validate_inputs</i>	Validates the input based on a few simple checks.

**classmethod** **calculate\_percent\_difference** (*cls*, *score*, *baseline\_score*)

Calculate the percent difference between scores.

**Parameters**

- **score** (*float*) – A score. Output of the score method of this objective.
- **baseline\_score** (*float*) – A score. Output of the score method of this objective. In practice, this is the score achieved on this objective with a baseline estimator.

**Returns**

**The percent difference between the scores. Note that for objectives that can be interpreted as percentages, this will be the difference between the reference score and score. For all other objectives, the difference will be normalized by the reference score.**

**Return type** float

**classmethod** **is\_defined\_for\_problem\_type** (*cls*, *problem\_type*)

**objective\_function** (*self*, *y\_true*, *y\_predicted*, *X=None*, *sample\_weight=None*)

Computes the relative value of the provided predictions compared to the actual labels, according a specified metric

**Arguments:** `y_predicted` (`pd.Series`): Predicted values of length `[n_samples]` `y_true` (`pd.Series`): Actual class labels of length `[n_samples]` `X` (`pd.DataFrame` or `np.ndarray`): Extra data of shape `[n_samples, n_features]` necessary to calculate score `sample_weight` (`pd.DataFrame` or `np.ndarray`): Sample weights used in computing objective value result

**Returns** Numerical value used to calculate score

**positive\_only** (`cls`)

If True, this objective is only valid for positive data. Default False.

**score** (`self, y_true, y_predicted, X=None, sample_weight=None`)

Returns a numerical score indicating performance based on the differences between the predicted and actual values.

**Parameters**

- **y\_predicted** (`pd.Series`) – Predicted values of length `[n_samples]`
- **y\_true** (`pd.Series`) – Actual class labels of length `[n_samples]`
- **X** (`pd.DataFrame` or `np.ndarray`) – Extra data of shape `[n_samples, n_features]` necessary to calculate score
- **sample\_weight** (`pd.DataFrame` or `np.ndarray`) – Sample weights used in computing objective value result

**Returns** score

**validate\_inputs** (`self, y_true, y_predicted`)

Validates the input based on a few simple checks.

**Parameters**

- **y\_predicted** (`pd.Series`, or `pd.DataFrame`) – Predicted values of length `[n_samples]`
- **y\_true** (`pd.Series`) – Actual class labels of length `[n_samples]`

**Returns** None

**class** `evalml.objectives.standard_metrics.RootMeanSquaredLogError`

Root mean squared log error for regression.

Only valid for nonnegative inputs. Otherwise, will throw a `ValueError`.

**Attributes**

<b>greater_is_better</b>	False
<b>is_bounded_like_percentage</b>	False
<b>name</b>	Root Mean Squared Log Error
<b>perfect_score</b>	0.0
<b>problem_types</b>	[ProblemTypes.REGRESSION, ProblemTypes.TIME_SERIES_REGRESSION]
<b>score_needs_proba</b>	False

**Methods**

<a href="#"><code>calculate_percent_difference</code></a>	Calculate the percent difference between scores.
---	--

continues on next page

Table 241 – continued from previous page

<i>is_defined_for_problem_type</i>	
<i>objective_function</i>	Computes the relative value of the provided predictions compared to the actual labels, according a specified metric
<i>positive_only</i>	If True, this objective is only valid for positive data. Default False.
<i>score</i>	Returns a numerical score indicating performance based on the differences between the predicted and actual values.
<i>validate_inputs</i>	Validates the input based on a few simple checks.

**classmethod** `calculate_percent_difference` (*cls*, *score*, *baseline\_score*)

Calculate the percent difference between scores.

#### Parameters

- **score** (*float*) – A score. Output of the score method of this objective.
- **baseline\_score** (*float*) – A score. Output of the score method of this objective. In practice, this is the score achieved on this objective with a baseline estimator.

#### Returns

**The percent difference between the scores. Note that for objectives that can be interpreted** as percentages, this will be the difference between the reference score and score. For all other objectives, the difference will be normalized by the reference score.

**Return type** `float`

**classmethod** `is_defined_for_problem_type` (*cls*, *problem\_type*)

**objective\_function** (*self*, *y\_true*, *y\_predicted*, *X=None*, *sample\_weight=None*)

Computes the relative value of the provided predictions compared to the actual labels, according a specified metric

**Arguments:** *y\_predicted* (*pd.Series*): Predicted values of length [*n\_samples*] *y\_true* (*pd.Series*): Actual class labels of length [*n\_samples*] *X* (*pd.DataFrame* or *np.ndarray*): Extra data of shape [*n\_samples*, *n\_features*] necessary to calculate score *sample\_weight* (*pd.DataFrame* or *np.ndarray*): Sample weights used in computing objective value result

**Returns** Numerical value used to calculate score

**positive\_only** (*self*)

If True, this objective is only valid for positive data. Default False.

**score** (*self*, *y\_true*, *y\_predicted*, *X=None*, *sample\_weight=None*)

Returns a numerical score indicating performance based on the differences between the predicted and actual values.

#### Parameters

- **y\_predicted** (*pd.Series*) – Predicted values of length [*n\_samples*]
- **y\_true** (*pd.Series*) – Actual class labels of length [*n\_samples*]
- **X** (*pd.DataFrame* or *np.ndarray*) – Extra data of shape [*n\_samples*, *n\_features*] necessary to calculate score

- **sample\_weight** (*pd.DataFrame* or *np.ndarray*) – Sample weights used in computing objective value result

**Returns** score

**validate\_inputs** (*self*, *y\_true*, *y\_predicted*)

Validates the input based on a few simple checks.

**Parameters**

- **y\_predicted** (*pd.Series*, or *pd.DataFrame*) – Predicted values of length [n\_samples]
- **y\_true** (*pd.Series*) – Actual class labels of length [n\_samples]

**Returns** None

## time\_series\_regression\_objective

### Module Contents

### Classes Summary

<i>TimeSeriesRegressionObjective</i>	Base class for all time series regression objectives.
--------------------------------------	---

### Contents

**class** evalml.objectives.time\_series\_regression\_objective.**TimeSeriesRegressionObjective**

Base class for all time series regression objectives.

**Attributes**

<b>problem_types</b>	[ProblemTypes.TIME_SERIES_REGRESSION]
----------------------	---------------------------------------

**Methods**

<i>calculate_percent_difference</i>	Calculate the percent difference between scores.
<i>greater_is_better</i>	Returns a boolean determining if a greater score indicates better model performance.
<i>is_bounded_like_percentage</i>	Returns whether this objective is bounded between 0 and 1, inclusive.
<i>is_defined_for_problem_type</i>	
<i>name</i>	Returns a name describing the objective.
<i>objective_function</i>	Computes the relative value of the provided predictions compared to the actual labels, according a specified metric
<i>perfect_score</i>	Returns the score obtained by evaluating this objective on a perfect model.
<i>positive_only</i>	If True, this objective is only valid for positive data. Default False.

continues on next page

Table 243 – continued from previous page

<code>score</code>	Returns a numerical score indicating performance based on the differences between the predicted and actual values.
<code>score_needs_proba</code>	Returns a boolean determining if the <code>score()</code> method needs probability estimates. This should be true for objectives which work with predicted probabilities, like log loss or AUC, and false for objectives which compare predicted class labels to the actual labels, like F1 or correlation.
<code>validate_inputs</code>	Validates the input based on a few simple checks.

**classmethod** `calculate_percent_difference` (*cls*, *score*, *baseline\_score*)

Calculate the percent difference between scores.

#### Parameters

- **score** (*float*) – A score. Output of the score method of this objective.
- **baseline\_score** (*float*) – A score. Output of the score method of this objective. In practice, this is the score achieved on this objective with a baseline estimator.

#### Returns

**The percent difference between the scores. Note that for objectives that can be interpreted as percentages, this will be the difference between the reference score and score. For all other objectives, the difference will be normalized by the reference score.**

**Return type** `float`

**property** `greater_is_better` (*cls*)

Returns a boolean determining if a greater score indicates better model performance.

**property** `is_bounded_like_percentage` (*cls*)

Returns whether this objective is bounded between 0 and 1, inclusive.

**classmethod** `is_defined_for_problem_type` (*cls*, *problem\_type*)

**property** `name` (*cls*)

Returns a name describing the objective.

**abstract classmethod** `objective_function` (*cls*, *y\_true*, *y\_predicted*, *X=None*, *sample\_weight=None*)

Computes the relative value of the provided predictions compared to the actual labels, according a specified metric

**Arguments:** *y\_predicted* (`pd.Series`): Predicted values of length [*n\_samples*] *y\_true* (`pd.Series`): Actual class labels of length [*n\_samples*] *X* (`pd.DataFrame` or `np.ndarray`): Extra data of shape [*n\_samples*, *n\_features*] necessary to calculate score *sample\_weight* (`pd.DataFrame` or `np.ndarray`): Sample weights used in computing objective value result

**Returns** Numerical value used to calculate score

**property** `perfect_score` (*cls*)

Returns the score obtained by evaluating this objective on a perfect model.

**property** `positive_only` (*cls*)

If True, this objective is only valid for positive data. Default False.

**score** (*self*, *y\_true*, *y\_predicted*, *X=None*, *sample\_weight=None*)

Returns a numerical score indicating performance based on the differences between the predicted and actual values.

#### Parameters

- **y\_predicted** (*pd.Series*) – Predicted values of length [n\_samples]
- **y\_true** (*pd.Series*) – Actual class labels of length [n\_samples]
- **X** (*pd.DataFrame* or *np.ndarray*) – Extra data of shape [n\_samples, n\_features] necessary to calculate score
- **sample\_weight** (*pd.DataFrame* or *np.ndarray*) – Sample weights used in computing objective value result

**Returns** score

**property score\_needs\_proba** (*cls*)

Returns a boolean determining if the score() method needs probability estimates. This should be true for objectives which work with predicted probabilities, like log loss or AUC, and false for objectives which compare predicted class labels to the actual labels, like F1 or correlation.

**validate\_inputs** (*self*, *y\_true*, *y\_predicted*)

Validates the input based on a few simple checks.

#### Parameters

- **y\_predicted** (*pd.Series*, or *pd.DataFrame*) – Predicted values of length [n\_samples]
- **y\_true** (*pd.Series*) – Actual class labels of length [n\_samples]

**Returns** None

## utils

## Module Contents

## Functions

<a href="#"><i>get_all_objective_names</i></a>	Get a list of the names of all objectives.
<a href="#"><i>get_core_objective_names</i></a>	Get a list of all valid core objectives.
<a href="#"><i>get_core_objectives</i></a>	Returns all core objective instances associated with the given problem type.
<a href="#"><i>get_non_core_objectives</i></a>	Get non-core objective classes.
<a href="#"><i>get_objective</i></a>	Returns the Objective class corresponding to a given objective name.

## Contents

`evalml.objectives.utils.get_all_objective_names()`

Get a list of the names of all objectives.

**Returns** Objective names

**Return type** list(str)

`evalml.objectives.utils.get_core_objective_names()`

Get a list of all valid core objectives.

**Returns** Objective names.

**Return type** list[str]

`evalml.objectives.utils.get_core_objectives(problem_type)`

Returns all core objective instances associated with the given problem type.

Core objectives are designed to work out-of-the-box for any dataset.

**Parameters** `problem_type` (str/ProblemTypes) – Type of problem

**Returns** List of ObjectiveBase instances

`evalml.objectives.utils.get_non_core_objectives()`

Get non-core objective classes.

Non-core objectives are objectives that are domain-specific. Users typically need to configure these objectives before using them in AutoMLSearch.

**Returns** List of ObjectiveBase classes

`evalml.objectives.utils.get_objective(objective, return_instance=False, **kwargs)`

Returns the Objective class corresponding to a given objective name.

**Parameters**

- **objective** (str or ObjectiveBase) – Name or instance of the objective class.
- **return\_instance** (bool) – Whether to return an instance of the objective. This only applies if objective is of type str. Note that the instance will be initialized with default arguments.
- **kwargs** (Any) – Any keyword arguments to pass into the objective. Only used when return\_instance=True.

**Returns** ObjectiveBase if the parameter objective is of type ObjectiveBase. If objective is instead a valid objective name, function will return the class corresponding to that name. If return\_instance is True, an instance of that objective will be returned.

## Package Contents

### Classes Summary

---

<code>AccuracyBinary</code>	Accuracy score for binary classification.
<code>AccuracyMulticlass</code>	Accuracy score for multiclass classification.
<code>AUC</code>	AUC score for binary classification.

---

continues on next page



Table 245 – continued from previous page

<i>AUCMacro</i>	AUC score for multiclass classification using macro averaging.
<i>AUCMicro</i>	AUC score for multiclass classification using micro averaging.
<i>AUCWeighted</i>	AUC Score for multiclass classification using weighted averaging.
<i>BalancedAccuracyBinary</i>	Balanced accuracy score for binary classification.
<i>BalancedAccuracyMulticlass</i>	Balanced accuracy score for multiclass classification.
<i>BinaryClassificationObjective</i>	Base class for all binary classification objectives.
<i>CostBenefitMatrix</i>	Score using a cost-benefit matrix. Scores quantify the benefits of a given value, so greater numeric
<i>ExpVariance</i>	Explained variance score for regression.
<i>F1</i>	F1 score for binary classification.
<i>F1Macro</i>	F1 score for multiclass classification using macro averaging.
<i>F1Micro</i>	F1 score for multiclass classification using micro averaging.
<i>F1Weighted</i>	F1 score for multiclass classification using weighted averaging.
<i>FraudCost</i>	Score the percentage of money lost of the total transaction amount process due to fraud.
<i>Gini</i>	Gini coefficient for binary classification.
<i>LeadScoring</i>	Lead scoring.
<i>LogLossBinary</i>	Log Loss for binary classification.
<i>LogLossMulticlass</i>	Log Loss for multiclass classification.
<i>MAE</i>	Mean absolute error for regression.
<i>MAPE</i>	Mean absolute percentage error for time series regression. Scaled by 100 to return a percentage.
<i>MaxError</i>	Maximum residual error for regression.
<i>MCCBinary</i>	Matthews correlation coefficient for binary classification.
<i>MCCMulticlass</i>	Matthews correlation coefficient for multiclass classification.
<i>MeanSquaredLogError</i>	Mean squared log error for regression.
<i>MedianAE</i>	Median absolute error for regression.
<i>MSE</i>	Mean squared error for regression.
<i>MulticlassClassificationObjective</i>	Base class for all multiclass classification objectives.
<i>ObjectiveBase</i>	Base class for all objectives.
<i>Precision</i>	Precision score for binary classification.
<i>PrecisionMacro</i>	Precision score for multiclass classification using macro averaging.
<i>PrecisionMicro</i>	Precision score for multiclass classification using micro averaging.
<i>PrecisionWeighted</i>	Precision score for multiclass classification using weighted averaging.
<i>R2</i>	Coefficient of determination for regression.
<i>Recall</i>	Recall score for binary classification.
<i>RecallMacro</i>	Recall score for multiclass classification using macro averaging.
<i>RecallMicro</i>	Recall score for multiclass classification using micro averaging.

continues on next page

Table 245 – continued from previous page

<i>RecallWeighted</i>	Recall score for multiclass classification using weighted averaging.
<i>RegressionObjective</i>	Base class for all regression objectives.
<i>RootMeanSquaredError</i>	Root mean squared error for regression.
<i>RootMeanSquaredLogError</i>	Root mean squared log error for regression.
<i>SensitivityLowAlert</i>	Base class for all binary classification objectives.

## Functions

<i>get_all_objective_names</i>	Get a list of the names of all objectives.
<i>get_core_objective_names</i>	Get a list of all valid core objectives.
<i>get_core_objectives</i>	Returns all core objective instances associated with the given problem type.
<i>get_non_core_objectives</i>	Get non-core objective classes.
<i>get_objective</i>	Returns the Objective class corresponding to a given objective name.

## Contents

**class** evalml.objectives.**AccuracyBinary**

Accuracy score for binary classification.

### Attributes

<b>greater_is_better</b>	True
<b>is_bounded_like_percentage</b>	True
<b>name</b>	Accuracy Binary
<b>perfect_score</b>	1.0
<b>problem_types</b>	[ProblemTypes.BINARY, ProblemTypes.TIME_SERIES_BINARY]
<b>score_needs_proba</b>	True

### Methods

<i>calculate_percent_difference</i>	Calculate the percent difference between scores.
<i>can_optimize_threshold</i>	Returns a boolean determining if we can optimize the binary classification objective threshold.
<i>decision_function</i>	Apply a learned threshold to predicted probabilities to get predicted classes.
<i>is_defined_for_problem_type</i>	
<i>objective_function</i>	Computes the relative value of the provided predictions compared to the actual labels, according a specified metric
<i>optimize_threshold</i>	Learn a binary classification threshold which optimizes the current objective.

continues on next page

Table 247 – continued from previous page

<code>positive_only</code>	If True, this objective is only valid for positive data. Default False.
<code>score</code>	Returns a numerical score indicating performance based on the differences between the predicted and actual values.
<code>validate_inputs</code>	Validates the input based on a few simple checks.

**classmethod** `calculate_percent_difference` (*cls*, *score*, *baseline\_score*)

Calculate the percent difference between scores.

#### Parameters

- **score** (*float*) – A score. Output of the score method of this objective.
- **baseline\_score** (*float*) – A score. Output of the score method of this objective. In practice, this is the score achieved on this objective with a baseline estimator.

#### Returns

**The percent difference between the scores. Note that for objectives that can be interpreted as percentages, this will be the difference between the reference score and score. For all other objectives, the difference will be normalized by the reference score.**

**Return type** float

**property** `can_optimize_threshold` (*cls*)

Returns a boolean determining if we can optimize the binary classification objective threshold. This will be false for any objective that works directly with predicted probabilities, like log loss and AUC. Otherwise, it will be true.

**decision\_function** (*self*, *ypred\_proba*, *threshold=0.5*, *X=None*)

Apply a learned threshold to predicted probabilities to get predicted classes.

#### Parameters

- **ypred\_proba** (*pd.Series*, *np.ndarray*) – The classifier’s predicted probabilities
- **threshold** (*float*, *optional*) – Threshold used to make a prediction. Defaults to 0.5.
- **X** (*pd.DataFrame*, *optional*) – Any extra columns that are needed from training data.

**Returns** predictions

**classmethod** `is_defined_for_problem_type` (*cls*, *problem\_type*)

**objective\_function** (*self*, *y\_true*, *y\_predicted*, *X=None*, *sample\_weight=None*)

Computes the relative value of the provided predictions compared to the actual labels, according to a specified metric

**Arguments:** *y\_predicted* (*pd.Series*): Predicted values of length [*n\_samples*] *y\_true* (*pd.Series*): Actual class labels of length [*n\_samples*] *X* (*pd.DataFrame* or *np.ndarray*): Extra data of shape [*n\_samples*, *n\_features*] necessary to calculate score *sample\_weight* (*pd.DataFrame* or *np.ndarray*): Sample weights used in computing objective value result

**Returns** Numerical value used to calculate score

**optimize\_threshold** (*self*, *ypred\_proba*, *y\_true*, *X=None*)

Learn a binary classification threshold which optimizes the current objective.

**Parameters**

- **ypred\_proba** (*pd.Series*) – The classifier’s predicted probabilities
- **y\_true** (*pd.Series*) – The ground truth for the predictions.
- **X** (*pd.DataFrame, optional*) – Any extra columns that are needed from training data.

**Returns** Optimal threshold for this objective

**positive\_only** (*cls*)

If True, this objective is only valid for positive data. Default False.

**score** (*self, y\_true, y\_predicted, X=None, sample\_weight=None*)

Returns a numerical score indicating performance based on the differences between the predicted and actual values.

**Parameters**

- **y\_predicted** (*pd.Series*) – Predicted values of length [n\_samples]
- **y\_true** (*pd.Series*) – Actual class labels of length [n\_samples]
- **X** (*pd.DataFrame or np.ndarray*) – Extra data of shape [n\_samples, n\_features] necessary to calculate score
- **sample\_weight** (*pd.DataFrame or np.ndarray*) – Sample weights used in computing objective value result

**Returns** score

**validate\_inputs** (*self, y\_true, y\_predicted*)

Validates the input based on a few simple checks.

**Parameters**

- **y\_predicted** (*pd.Series, or pd.DataFrame*) – Predicted values of length [n\_samples]
- **y\_true** (*pd.Series*) – Actual class labels of length [n\_samples]

**Returns** None

**class** evalml.objectives.**AccuracyMulticlass**

Accuracy score for multiclass classification.

**Attributes**

<b>greater_is_better</b>	True
<b>is_bounded_like_percentage</b>	True
<b>name</b>	Accuracy Multiclass
<b>perfect_score</b>	1.0
<b>problem_types</b>	[ProblemTypes.MULTICLASS, ProblemTypes.TIME_SERIES_MULTICLASS]
<b>score_needs_proba</b>	False

**Methods**

---

<a href="#"><i>calculate_percent_difference</i></a>	Calculate the percent difference between scores.
---	--

---

continues on next page

Table 248 – continued from previous page

<i>is_defined_for_problem_type</i>	
<i>objective_function</i>	Computes the relative value of the provided predictions compared to the actual labels, according a specified metric
<i>positive_only</i>	If True, this objective is only valid for positive data. Default False.
<i>score</i>	Returns a numerical score indicating performance based on the differences between the predicted and actual values.
<i>validate_inputs</i>	Validates the input based on a few simple checks.

**classmethod** `calculate_percent_difference` (*cls*, *score*, *baseline\_score*)

Calculate the percent difference between scores.

#### Parameters

- **score** (*float*) – A score. Output of the score method of this objective.
- **baseline\_score** (*float*) – A score. Output of the score method of this objective. In practice, this is the score achieved on this objective with a baseline estimator.

#### Returns

**The percent difference between the scores. Note that for objectives that can be interpreted** as percentages, this will be the difference between the reference score and score. For all other objectives, the difference will be normalized by the reference score.

**Return type** `float`

**classmethod** `is_defined_for_problem_type` (*cls*, *problem\_type*)

**objective\_function** (*self*, *y\_true*, *y\_predicted*, *X=None*, *sample\_weight=None*)

Computes the relative value of the provided predictions compared to the actual labels, according a specified metric

**Arguments:** *y\_predicted* (*pd.Series*): Predicted values of length [*n\_samples*] *y\_true* (*pd.Series*): Actual class labels of length [*n\_samples*] *X* (*pd.DataFrame* or *np.ndarray*): Extra data of shape [*n\_samples*, *n\_features*] necessary to calculate score *sample\_weight* (*pd.DataFrame* or *np.ndarray*): Sample weights used in computing objective value result

**Returns** Numerical value used to calculate score

**positive\_only** (*cls*)

If True, this objective is only valid for positive data. Default False.

**score** (*self*, *y\_true*, *y\_predicted*, *X=None*, *sample\_weight=None*)

Returns a numerical score indicating performance based on the differences between the predicted and actual values.

#### Parameters

- **y\_predicted** (*pd.Series*) – Predicted values of length [*n\_samples*]
- **y\_true** (*pd.Series*) – Actual class labels of length [*n\_samples*]
- **X** (*pd.DataFrame* or *np.ndarray*) – Extra data of shape [*n\_samples*, *n\_features*] necessary to calculate score

- **sample\_weight** (*pd.DataFrame* or *np.ndarray*) – Sample weights used in computing objective value result

**Returns** score

**validate\_inputs** (*self*, *y\_true*, *y\_predicted*)

Validates the input based on a few simple checks.

**Parameters**

- **y\_predicted** (*pd.Series*, or *pd.DataFrame*) – Predicted values of length [n\_samples]
- **y\_true** (*pd.Series*) – Actual class labels of length [n\_samples]

**Returns** None

**class** evalml.objectives.**AUC**

AUC score for binary classification.

**Attributes**

<b>greater_is_better</b>	True
<b>is_bounded_like_percentage</b>	True
<b>name</b>	AUC
<b>perfect_score</b>	1.0
<b>problem_types</b>	[ProblemTypes.BINARY, ProblemTypes.TIME_SERIES_BINARY]
<b>score_needs_proba</b>	True

**Methods**

<i>calculate_percent_difference</i>	Calculate the percent difference between scores.
<i>can_optimize_threshold</i>	Returns a boolean determining if we can optimize the binary classification objective threshold.
<i>decision_function</i>	Apply a learned threshold to predicted probabilities to get predicted classes.
<i>is_defined_for_problem_type</i>	
<i>objective_function</i>	Computes the relative value of the provided predictions compared to the actual labels, according a specified metric
<i>optimize_threshold</i>	Learn a binary classification threshold which optimizes the current objective.
<i>positive_only</i>	If True, this objective is only valid for positive data. Default False.
<i>score</i>	Returns a numerical score indicating performance based on the differences between the predicted and actual values.
<i>validate_inputs</i>	Validates the input based on a few simple checks.

**classmethod** **calculate\_percent\_difference** (*cls*, *score*, *baseline\_score*)

Calculate the percent difference between scores.

**Parameters**

- **score** (*float*) – A score. Output of the score method of this objective.
- **baseline\_score** (*float*) – A score. Output of the score method of this objective. In practice, this is the score achieved on this objective with a baseline estimator.

**Returns**

**The percent difference between the scores. Note that for objectives that can be interpreted** as percentages, this will be the difference between the reference score and score. For all other objectives, the difference will be normalized by the reference score.

**Return type** float

**property can\_optimize\_threshold** (*cls*)

Returns a boolean determining if we can optimize the binary classification objective threshold. This will be false for any objective that works directly with predicted probabilities, like log loss and AUC. Otherwise, it will be true.

**decision\_function** (*self*, *ypred\_proba*, *threshold=0.5*, *X=None*)

Apply a learned threshold to predicted probabilities to get predicted classes.

**Parameters**

- **ypred\_proba** (*pd.Series*, *np.ndarray*) – The classifier’s predicted probabilities
- **threshold** (*float*, *optional*) – Threshold used to make a prediction. Defaults to 0.5.
- **X** (*pd.DataFrame*, *optional*) – Any extra columns that are needed from training data.

**Returns** predictions

**classmethod is\_defined\_for\_problem\_type** (*cls*, *problem\_type*)

**objective\_function** (*self*, *y\_true*, *y\_predicted*, *X=None*, *sample\_weight=None*)

Computes the relative value of the provided predictions compared to the actual labels, according a specified metric

**Arguments:** *y\_predicted* (*pd.Series*): Predicted values of length [*n\_samples*] *y\_true* (*pd.Series*): Actual class labels of length [*n\_samples*] *X* (*pd.DataFrame* or *np.ndarray*): Extra data of shape [*n\_samples*, *n\_features*] necessary to calculate score *sample\_weight* (*pd.DataFrame* or *np.ndarray*): Sample weights used in computing objective value result

**Returns** Numerical value used to calculate score

**optimize\_threshold** (*self*, *ypred\_proba*, *y\_true*, *X=None*)

Learn a binary classification threshold which optimizes the current objective.

**Parameters**

- **ypred\_proba** (*pd.Series*) – The classifier’s predicted probabilities
- **y\_true** (*pd.Series*) – The ground truth for the predictions.
- **X** (*pd.DataFrame*, *optional*) – Any extra columns that are needed from training data.

**Returns** Optimal threshold for this objective

**positive\_only** (*cls*)

If True, this objective is only valid for positive data. Default False.

**score** (*self*, *y\_true*, *y\_predicted*, *X=None*, *sample\_weight=None*)

Returns a numerical score indicating performance based on the differences between the predicted and actual values.

#### Parameters

- **y\_predicted** (*pd.Series*) – Predicted values of length [n\_samples]
- **y\_true** (*pd.Series*) – Actual class labels of length [n\_samples]
- **X** (*pd.DataFrame* or *np.ndarray*) – Extra data of shape [n\_samples, n\_features] necessary to calculate score
- **sample\_weight** (*pd.DataFrame* or *np.ndarray*) – Sample weights used in computing objective value result

**Returns** score

**validate\_inputs** (*self*, *y\_true*, *y\_predicted*)

Validates the input based on a few simple checks.

#### Parameters

- **y\_predicted** (*pd.Series*, or *pd.DataFrame*) – Predicted values of length [n\_samples]
- **y\_true** (*pd.Series*) – Actual class labels of length [n\_samples]

**Returns** None

**class** evalml.objectives.**AUCMacro**

AUC score for multiclass classification using macro averaging.

#### Attributes

<b>greater_is_better</b>	True
<b>is_bounded_like_percentage</b>	True
<b>name</b>	AUC Macro
<b>perfect_score</b>	1.0
<b>problem_types</b>	[ProblemTypes.MULTICLASS, ProblemTypes.TIME_SERIES_MULTICLASS]
<b>score_needs_proba</b>	True

#### Methods

<i>calculate_percent_difference</i>	Calculate the percent difference between scores.
<i>is_defined_for_problem_type</i>	
<i>objective_function</i>	Computes the relative value of the provided predictions compared to the actual labels, according a specified metric
<i>positive_only</i>	If True, this objective is only valid for positive data. Default False.
<i>score</i>	Returns a numerical score indicating performance based on the differences between the predicted and actual values.
<i>validate_inputs</i>	Validates the input based on a few simple checks.



**classmethod** `calculate_percent_difference` (*cls*, *score*, *baseline\_score*)

Calculate the percent difference between scores.

#### Parameters

- **score** (*float*) – A score. Output of the score method of this objective.
- **baseline\_score** (*float*) – A score. Output of the score method of this objective. In practice, this is the score achieved on this objective with a baseline estimator.

#### Returns

**The percent difference between the scores. Note that for objectives that can be interpreted as percentages, this will be the difference between the reference score and score. For all other objectives, the difference will be normalized by the reference score.**

**Return type** `float`

**classmethod** `is_defined_for_problem_type` (*cls*, *problem\_type*)

**objective\_function** (*self*, *y\_true*, *y\_predicted*, *X=None*, *sample\_weight=None*)

Computes the relative value of the provided predictions compared to the actual labels, according a specified metric

**Arguments:** *y\_predicted* (`pd.Series`): Predicted values of length [*n\_samples*] *y\_true* (`pd.Series`): Actual class labels of length [*n\_samples*] *X* (`pd.DataFrame` or `np.ndarray`): Extra data of shape [*n\_samples*, *n\_features*] necessary to calculate score *sample\_weight* (`pd.DataFrame` or `np.ndarray`): Sample weights used in computing objective value result

**Returns** Numerical value used to calculate score

**positive\_only** (*cls*)

If True, this objective is only valid for positive data. Default False.

**score** (*self*, *y\_true*, *y\_predicted*, *X=None*, *sample\_weight=None*)

Returns a numerical score indicating performance based on the differences between the predicted and actual values.

#### Parameters

- **y\_predicted** (`pd.Series`) – Predicted values of length [*n\_samples*]
- **y\_true** (`pd.Series`) – Actual class labels of length [*n\_samples*]
- **X** (`pd.DataFrame` or `np.ndarray`) – Extra data of shape [*n\_samples*, *n\_features*] necessary to calculate score
- **sample\_weight** (`pd.DataFrame` or `np.ndarray`) – Sample weights used in computing objective value result

**Returns** `score`

**validate\_inputs** (*self*, *y\_true*, *y\_predicted*)

Validates the input based on a few simple checks.

#### Parameters

- **y\_predicted** (`pd.Series`, or `pd.DataFrame`) – Predicted values of length [*n\_samples*]
- **y\_true** (`pd.Series`) – Actual class labels of length [*n\_samples*]

**Returns** `None`

**class** evalml.objectives.AUCMicro

AUC score for multiclass classification using micro averaging.

#### Attributes

<b>greater_is_better</b>	True
<b>is_bounded_like_percentage</b>	True
<b>name</b>	AUC Micro
<b>perfect_score</b>	1.0
<b>problem_types</b>	[ProblemTypes.MULTICLASS, ProblemTypes.TIME_SERIES_MULTICLASS]
<b>score_needs_proba</b>	True

#### Methods

<i>calculate_percent_difference</i>	Calculate the percent difference between scores.
<i>is_defined_for_problem_type</i>	
<i>objective_function</i>	Computes the relative value of the provided predictions compared to the actual labels, according a specified metric
<i>positive_only</i>	If True, this objective is only valid for positive data. Default False.
<i>score</i>	Returns a numerical score indicating performance based on the differences between the predicted and actual values.
<i>validate_inputs</i>	Validates the input based on a few simple checks.

**classmethod** *calculate\_percent\_difference* (*cls*, *score*, *baseline\_score*)

Calculate the percent difference between scores.

#### Parameters

- **score** (*float*) – A score. Output of the score method of this objective.
- **baseline\_score** (*float*) – A score. Output of the score method of this objective. In practice, this is the score achieved on this objective with a baseline estimator.

#### Returns

**The percent difference between the scores. Note that for objectives that can be interpreted as percentages, this will be the difference between the reference score and score. For all other objectives, the difference will be normalized by the reference score.**

**Return type** float

**classmethod** *is\_defined\_for\_problem\_type* (*cls*, *problem\_type*)

**objective\_function** (*self*, *y\_true*, *y\_predicted*, *X=None*, *sample\_weight=None*)

Computes the relative value of the provided predictions compared to the actual labels, according a specified metric

**Arguments:** *y\_predicted* (pd.Series): Predicted values of length [n\_samples] *y\_true* (pd.Series): Actual class labels of length [n\_samples] *X* (pd.DataFrame or np.ndarray): Extra data of shape [n\_samples, n\_features] necessary to calculate score *sample\_weight* (pd.DataFrame or np.ndarray): Sample weights used in computing objective value result

**Returns** Numerical value used to calculate score

**positive\_only** (*cls*)

If True, this objective is only valid for positive data. Default False.

**score** (*self, y\_true, y\_predicted, X=None, sample\_weight=None*)

Returns a numerical score indicating performance based on the differences between the predicted and actual values.

**Parameters**

- **y\_predicted** (*pd.Series*) – Predicted values of length [n\_samples]
- **y\_true** (*pd.Series*) – Actual class labels of length [n\_samples]
- **X** (*pd.DataFrame or np.ndarray*) – Extra data of shape [n\_samples, n\_features] necessary to calculate score
- **sample\_weight** (*pd.DataFrame or np.ndarray*) – Sample weights used in computing objective value result

**Returns** score

**validate\_inputs** (*self, y\_true, y\_predicted*)

Validates the input based on a few simple checks.

**Parameters**

- **y\_predicted** (*pd.Series, or pd.DataFrame*) – Predicted values of length [n\_samples]
- **y\_true** (*pd.Series*) – Actual class labels of length [n\_samples]

**Returns** None

**class** evalml.objectives.**AUCWeighted**

AUC Score for multiclass classification using weighted averaging.

**Attributes**

<b>greater_is_better</b>	True
<b>is_bounded_like_percentage</b>	True
<b>name</b>	AUC Weighted
<b>perfect_score</b>	1.0
<b>problem_types</b>	[ProblemTypes.MULTICLASS, ProblemTypes.TIME_SERIES_MULTICLASS]
<b>score_needs_proba</b>	True

**Methods**

<i>calculate_percent_difference</i>	Calculate the percent difference between scores.
<i>is_defined_for_problem_type</i>	
<i>objective_function</i>	Computes the relative value of the provided predictions compared to the actual labels, according a specified metric
<i>positive_only</i>	If True, this objective is only valid for positive data. Default False.

continues on next page

Table 252 – continued from previous page

<code>score</code>	Returns a numerical score indicating performance based on the differences between the predicted and actual values.
<code>validate_inputs</code>	Validates the input based on a few simple checks.

**classmethod** `calculate_percent_difference` (*cls*, *score*, *baseline\_score*)

Calculate the percent difference between scores.

#### Parameters

- **score** (*float*) – A score. Output of the score method of this objective.
- **baseline\_score** (*float*) – A score. Output of the score method of this objective. In practice, this is the score achieved on this objective with a baseline estimator.

#### Returns

**The percent difference between the scores. Note that for objectives that can be interpreted as percentages, this will be the difference between the reference score and score. For all other objectives, the difference will be normalized by the reference score.**

**Return type** `float`

**classmethod** `is_defined_for_problem_type` (*cls*, *problem\_type*)

**objective\_function** (*self*, *y\_true*, *y\_predicted*, *X=None*, *sample\_weight=None*)

Computes the relative value of the provided predictions compared to the actual labels, according a specified metric

**Arguments:** *y\_predicted* (*pd.Series*): Predicted values of length [*n\_samples*] *y\_true* (*pd.Series*): Actual class labels of length [*n\_samples*] *X* (*pd.DataFrame* or *np.ndarray*): Extra data of shape [*n\_samples*, *n\_features*] necessary to calculate score *sample\_weight* (*pd.DataFrame* or *np.ndarray*): Sample weights used in computing objective value result

**Returns** Numerical value used to calculate score

**positive\_only** (*cls*)

If True, this objective is only valid for positive data. Default False.

**score** (*self*, *y\_true*, *y\_predicted*, *X=None*, *sample\_weight=None*)

Returns a numerical score indicating performance based on the differences between the predicted and actual values.

#### Parameters

- **y\_predicted** (*pd.Series*) – Predicted values of length [*n\_samples*]
- **y\_true** (*pd.Series*) – Actual class labels of length [*n\_samples*]
- **X** (*pd.DataFrame* or *np.ndarray*) – Extra data of shape [*n\_samples*, *n\_features*] necessary to calculate score
- **sample\_weight** (*pd.DataFrame* or *np.ndarray*) – Sample weights used in computing objective value result

**Returns** `score`

**validate\_inputs** (*self*, *y\_true*, *y\_predicted*)

Validates the input based on a few simple checks.

#### Parameters

- **y\_predicted** (*pd.Series*, or *pd.DataFrame*) – Predicted values of length [n\_samples]
- **y\_true** (*pd.Series*) – Actual class labels of length [n\_samples]

**Returns** None

**class** evalml.objectives.BalancedAccuracyBinary

Balanced accuracy score for binary classification.

**Attributes**

<b>greater_is_better</b>	True
<b>is_bounded_like_percentage</b>	True
<b>name</b>	Balanced Accuracy Binary
<b>perfect_score</b>	1.0
<b>problem_types</b>	[ProblemTypes.BINARY, ProblemTypes.TIME_SERIES_BINARY]
<b>score_needs_proba</b>	False

**Methods**

<i>calculate_percent_difference</i>	Calculate the percent difference between scores.
<i>can_optimize_threshold</i>	Returns a boolean determining if we can optimize the binary classification objective threshold.
<i>decision_function</i>	Apply a learned threshold to predicted probabilities to get predicted classes.
<i>is_defined_for_problem_type</i>	
<i>objective_function</i>	Computes the relative value of the provided predictions compared to the actual labels, according a specified metric
<i>optimize_threshold</i>	Learn a binary classification threshold which optimizes the current objective.
<i>positive_only</i>	If True, this objective is only valid for positive data. Default False.
<i>score</i>	Returns a numerical score indicating performance based on the differences between the predicted and actual values.
<i>validate_inputs</i>	Validates the input based on a few simple checks.

**classmethod** **calculate\_percent\_difference** (*cls*, *score*, *baseline\_score*)

Calculate the percent difference between scores.

**Parameters**

- **score** (*float*) – A score. Output of the score method of this objective.
- **baseline\_score** (*float*) – A score. Output of the score method of this objective. In practice, this is the score achieved on this objective with a baseline estimator.

**Returns**

**The percent difference between the scores. Note that for objectives that can be interpreted as percentages, this will be the difference between the reference score and score. For all other objectives, the difference will be normalized by the reference score.**

**Return type** float

**property can\_optimize\_threshold** (*cls*)

Returns a boolean determining if we can optimize the binary classification objective threshold. This will be false for any objective that works directly with predicted probabilities, like log loss and AUC. Otherwise, it will be true.

**decision\_function** (*self*, *ypred\_proba*, *threshold=0.5*, *X=None*)

Apply a learned threshold to predicted probabilities to get predicted classes.

**Parameters**

- **ypred\_proba** (*pd.Series*, *np.ndarray*) – The classifier’s predicted probabilities
- **threshold** (*float*, *optional*) – Threshold used to make a prediction. Defaults to 0.5.
- **X** (*pd.DataFrame*, *optional*) – Any extra columns that are needed from training data.

**Returns** predictions

**classmethod is\_defined\_for\_problem\_type** (*cls*, *problem\_type*)

**objective\_function** (*self*, *y\_true*, *y\_predicted*, *X=None*, *sample\_weight=None*)

Computes the relative value of the provided predictions compared to the actual labels, according a specified metric

**Arguments:** *y\_predicted* (*pd.Series*): Predicted values of length [*n\_samples*] *y\_true* (*pd.Series*): Actual class labels of length [*n\_samples*] *X* (*pd.DataFrame* or *np.ndarray*): Extra data of shape [*n\_samples*, *n\_features*] necessary to calculate score *sample\_weight* (*pd.DataFrame* or *np.ndarray*): Sample weights used in computing objective value result

**Returns** Numerical value used to calculate score

**optimize\_threshold** (*self*, *ypred\_proba*, *y\_true*, *X=None*)

Learn a binary classification threshold which optimizes the current objective.

**Parameters**

- **ypred\_proba** (*pd.Series*) – The classifier’s predicted probabilities
- **y\_true** (*pd.Series*) – The ground truth for the predictions.
- **X** (*pd.DataFrame*, *optional*) – Any extra columns that are needed from training data.

**Returns** Optimal threshold for this objective

**positive\_only** (*cls*)

If True, this objective is only valid for positive data. Default False.

**score** (*self*, *y\_true*, *y\_predicted*, *X=None*, *sample\_weight=None*)

Returns a numerical score indicating performance based on the differences between the predicted and actual values.

**Parameters**

- **y\_predicted** (*pd.Series*) – Predicted values of length [*n\_samples*]
- **y\_true** (*pd.Series*) – Actual class labels of length [*n\_samples*]

- **X** (*pd.DataFrame* or *np.ndarray*) – Extra data of shape [n\_samples, n\_features] necessary to calculate score
- **sample\_weight** (*pd.DataFrame* or *np.ndarray*) – Sample weights used in computing objective value result

**Returns** score

**validate\_inputs** (*self*, *y\_true*, *y\_predicted*)

Validates the input based on a few simple checks.

**Parameters**

- **y\_predicted** (*pd.Series*, or *pd.DataFrame*) – Predicted values of length [n\_samples]
- **y\_true** (*pd.Series*) – Actual class labels of length [n\_samples]

**Returns** None

**class** evalml.objectives.**BalancedAccuracyMulticlass**

Balanced accuracy score for multiclass classification.

**Attributes**

<b>greater_is_better</b>	True
<b>is_bounded_like_percentage</b>	True
<b>name</b>	Balanced Accuracy Multiclass
<b>perfect_score</b>	1.0
<b>problem_types</b>	[ProblemTypes.MULTICLASS, ProblemTypes.TIME_SERIES_MULTICLASS]
<b>score_needs_proba</b>	False

**Methods**

<i>calculate_percent_difference</i>	Calculate the percent difference between scores.
<i>is_defined_for_problem_type</i>	
<i>objective_function</i>	Computes the relative value of the provided predictions compared to the actual labels, according a specified metric
<i>positive_only</i>	If True, this objective is only valid for positive data. Default False.
<i>score</i>	Returns a numerical score indicating performance based on the differences between the predicted and actual values.
<i>validate_inputs</i>	Validates the input based on a few simple checks.

**classmethod** **calculate\_percent\_difference** (*cls*, *score*, *baseline\_score*)

Calculate the percent difference between scores.

**Parameters**

- **score** (*float*) – A score. Output of the score method of this objective.
- **baseline\_score** (*float*) – A score. Output of the score method of this objective. In practice, this is the score achieved on this objective with a baseline estimator.

**Returns**

**The percent difference between the scores.** Note that for objectives that can be interpreted as percentages, this will be the difference between the reference score and score. For all other objectives, the difference will be normalized by the reference score.

**Return type** float

**classmethod** `is_defined_for_problem_type(cls, problem_type)`

**objective\_function** (*self*, *y\_true*, *y\_predicted*, *X=None*, *sample\_weight=None*)

Computes the relative value of the provided predictions compared to the actual labels, according a specified metric

**Arguments:** *y\_predicted* (pd.Series): Predicted values of length [n\_samples] *y\_true* (pd.Series): Actual class labels of length [n\_samples] *X* (pd.DataFrame or np.ndarray): Extra data of shape [n\_samples, n\_features] necessary to calculate score *sample\_weight* (pd.DataFrame or np.ndarray): Sample weights used in computing objective value result

**Returns** Numerical value used to calculate score

**positive\_only** (*cls*)

If True, this objective is only valid for positive data. Default False.

**score** (*self*, *y\_true*, *y\_predicted*, *X=None*, *sample\_weight=None*)

Returns a numerical score indicating performance based on the differences between the predicted and actual values.

**Parameters**

- **y\_predicted** (pd.Series) – Predicted values of length [n\_samples]
- **y\_true** (pd.Series) – Actual class labels of length [n\_samples]
- **X** (pd.DataFrame or np.ndarray) – Extra data of shape [n\_samples, n\_features] necessary to calculate score
- **sample\_weight** (pd.DataFrame or np.ndarray) – Sample weights used in computing objective value result

**Returns** score

**validate\_inputs** (*self*, *y\_true*, *y\_predicted*)

Validates the input based on a few simple checks.

**Parameters**

- **y\_predicted** (pd.Series, or pd.DataFrame) – Predicted values of length [n\_samples]
- **y\_true** (pd.Series) – Actual class labels of length [n\_samples]

**Returns** None

**class** evalml.objectives.**BinaryClassificationObjective**

Base class for all binary classification objectives.

**Attributes**

<b>problem_types</b>	[ProblemTypes.BINARY, ProblemTypes.TIME_SERIES_BINARY]
----------------------	--

**Methods**



<code>calculate_percent_difference</code>	Calculate the percent difference between scores.
<code>can_optimize_threshold</code>	Returns a boolean determining if we can optimize the binary classification objective threshold.
<code>decision_function</code>	Apply a learned threshold to predicted probabilities to get predicted classes.
<code>greater_is_better</code>	Returns a boolean determining if a greater score indicates better model performance.
<code>is_bounded_like_percentage</code>	Returns whether this objective is bounded between 0 and 1, inclusive.
<code>is_defined_for_problem_type</code>	
<code>name</code>	Returns a name describing the objective.
<code>objective_function</code>	Computes the relative value of the provided predictions compared to the actual labels, according a specified metric
<code>optimize_threshold</code>	Learn a binary classification threshold which optimizes the current objective.
<code>perfect_score</code>	Returns the score obtained by evaluating this objective on a perfect model.
<code>positive_only</code>	If True, this objective is only valid for positive data. Default False.
<code>score</code>	Returns a numerical score indicating performance based on the differences between the predicted and actual values.
<code>score_needs_proba</code>	Returns a boolean determining if the score() method needs probability estimates. This should be true for objectives which work with predicted probabilities, like log loss or AUC, and false for objectives which compare predicted class labels to the actual labels, like F1 or correlation.
<code>validate_inputs</code>	Validates the input based on a few simple checks.

**classmethod** `calculate_percent_difference` (*cls*, *score*, *baseline\_score*)

Calculate the percent difference between scores.

#### Parameters

- **score** (*float*) – A score. Output of the score method of this objective.
- **baseline\_score** (*float*) – A score. Output of the score method of this objective. In practice, this is the score achieved on this objective with a baseline estimator.

#### Returns

**The percent difference between the scores. Note that for objectives that can be interpreted as percentages, this will be the difference between the reference score and score. For all other objectives, the difference will be normalized by the reference score.**

**Return type** float

**property** `can_optimize_threshold` (*cls*)

Returns a boolean determining if we can optimize the binary classification objective threshold. This will be false for any objective that works directly with predicted probabilities, like log loss and AUC. Otherwise, it will be true.

**decision\_function** (*self*, *ypred\_proba*, *threshold=0.5*, *X=None*)

Apply a learned threshold to predicted probabilities to get predicted classes.

#### Parameters

- **ypred\_proba** (*pd.Series, np.ndarray*) – The classifier’s predicted probabilities
- **threshold** (*float, optional*) – Threshold used to make a prediction. Defaults to 0.5.
- **X** (*pd.DataFrame, optional*) – Any extra columns that are needed from training data.

**Returns** predictions

**property greater\_is\_better** (*cls*)

Returns a boolean determining if a greater score indicates better model performance.

**property is\_bounded\_like\_percentage** (*cls*)

Returns whether this objective is bounded between 0 and 1, inclusive.

**classmethod is\_defined\_for\_problem\_type** (*cls, problem\_type*)

**property name** (*cls*)

Returns a name describing the objective.

**abstract classmethod objective\_function** (*cls, y\_true, y\_predicted, X=None, sample\_weight=None*)

Computes the relative value of the provided predictions compared to the actual labels, according a specified metric

**Arguments:** *y\_predicted* (*pd.Series*): Predicted values of length *[n\_samples]* *y\_true* (*pd.Series*): Actual class labels of length *[n\_samples]* *X* (*pd.DataFrame* or *np.ndarray*): Extra data of shape *[n\_samples, n\_features]* necessary to calculate score *sample\_weight* (*pd.DataFrame* or *np.ndarray*): Sample weights used in computing objective value result

**Returns** Numerical value used to calculate score

**optimize\_threshold** (*self, ypred\_proba, y\_true, X=None*)

Learn a binary classification threshold which optimizes the current objective.

#### Parameters

- **ypred\_proba** (*pd.Series*) – The classifier’s predicted probabilities
- **y\_true** (*pd.Series*) – The ground truth for the predictions.
- **X** (*pd.DataFrame, optional*) – Any extra columns that are needed from training data.

**Returns** Optimal threshold for this objective

**property perfect\_score** (*cls*)

Returns the score obtained by evaluating this objective on a perfect model.

**positive\_only** (*cls*)

If True, this objective is only valid for positive data. Default False.

**score** (*self, y\_true, y\_predicted, X=None, sample\_weight=None*)

Returns a numerical score indicating performance based on the differences between the predicted and actual values.

#### Parameters

- **y\_predicted** (*pd.Series*) – Predicted values of length *[n\_samples]*

- **y\_true** (*pd.Series*) – Actual class labels of length [n\_samples]
- **X** (*pd.DataFrame* or *np.ndarray*) – Extra data of shape [n\_samples, n\_features] necessary to calculate score
- **sample\_weight** (*pd.DataFrame* or *np.ndarray*) – Sample weights used in computing objective value result

Returns score

**property score\_needs\_proba** (*cls*)

Returns a boolean determining if the score() method needs probability estimates. This should be true for objectives which work with predicted probabilities, like log loss or AUC, and false for objectives which compare predicted class labels to the actual labels, like F1 or correlation.

**validate\_inputs** (*self, y\_true, y\_predicted*)

Validates the input based on a few simple checks.

Parameters

- **y\_predicted** (*pd.Series*, or *pd.DataFrame*) – Predicted values of length [n\_samples]
- **y\_true** (*pd.Series*) – Actual class labels of length [n\_samples]

Returns None

**class** evalml.objectives.**CostBenefitMatrix** (*true\_positive, true\_negative, false\_positive, false\_negative*)

Score using a cost-benefit matrix. Scores quantify the benefits of a given value, so greater numeric scores represents a better score. Costs and scores can be negative, indicating that a value is not beneficial. For example, in the case of monetary profit, a negative cost and/or score represents loss of cash flow.

Parameters

- **true\_positive** (*float*) – Cost associated with true positive predictions
- **true\_negative** (*float*) – Cost associated with true negative predictions
- **false\_positive** (*float*) – Cost associated with false positive predictions
- **false\_negative** (*float*) – Cost associated with false negative predictions

Attributes

<b>greater_is_better</b>	True
<b>is_bounded_like_percentage</b>	False
<b>name</b>	Cost Benefit Matrix
<b>perfect_score</b>	None
<b>problem_types</b>	[ProblemTypes.BINARY, ProblemTypes.TIME_SERIES_BINARY]
<b>score_needs_proba</b>	False

Methods

<i>calculate_percent_difference</i>	Calculate the percent difference between scores.
<i>can_optimize_threshold</i>	Returns a boolean determining if we can optimize the binary classification objective threshold.

continues on next page

Table 256 – continued from previous page

<code>decision_function</code>	Apply a learned threshold to predicted probabilities to get predicted classes.
<code>is_defined_for_problem_type</code>	
<code>objective_function</code>	Calculates cost-benefit of the using the predicted and true values.
<code>optimize_threshold</code>	Learn a binary classification threshold which optimizes the current objective.
<code>positive_only</code>	If True, this objective is only valid for positive data. Default False.
<code>score</code>	Returns a numerical score indicating performance based on the differences between the predicted and actual values.
<code>validate_inputs</code>	Validates the input based on a few simple checks.

**classmethod** `calculate_percent_difference` (*cls*, *score*, *baseline\_score*)

Calculate the percent difference between scores.

#### Parameters

- **score** (*float*) – A score. Output of the score method of this objective.
- **baseline\_score** (*float*) – A score. Output of the score method of this objective. In practice, this is the score achieved on this objective with a baseline estimator.

#### Returns

**The percent difference between the scores. Note that for objectives that can be interpreted** as percentages, this will be the difference between the reference score and score. For all other objectives, the difference will be normalized by the reference score.

**Return type** `float`

**property** `can_optimize_threshold` (*cls*)

Returns a boolean determining if we can optimize the binary classification objective threshold. This will be false for any objective that works directly with predicted probabilities, like log loss and AUC. Otherwise, it will be true.

**decision\_function** (*self*, *ypred\_proba*, *threshold=0.5*, *X=None*)

Apply a learned threshold to predicted probabilities to get predicted classes.

#### Parameters

- **ypred\_proba** (*pd.Series*, *np.ndarray*) – The classifier’s predicted probabilities
- **threshold** (*float*, *optional*) – Threshold used to make a prediction. Defaults to 0.5.
- **X** (*pd.DataFrame*, *optional*) – Any extra columns that are needed from training data.

**Returns** `predictions`

**classmethod** `is_defined_for_problem_type` (*cls*, *problem\_type*)

**objective\_function** (*self*, *y\_true*, *y\_predicted*, *X=None*, *sample\_weight=None*)

Calculates cost-benefit of the using the predicted and true values.

#### Parameters

- **y\_predicted** (*pd.Series*) – Predicted labels
- **y\_true** (*pd.Series*) – True labels
- **X** (*pd.DataFrame*) – Ignored.
- **sample\_weight** (*pd.DataFrame*) – Ignored.

**Returns** Cost-benefit matrix score

**Return type** float

**optimize\_threshold** (*self, ypred\_proba, y\_true, X=None*)

Learn a binary classification threshold which optimizes the current objective.

**Parameters**

- **ypred\_proba** (*pd.Series*) – The classifier’s predicted probabilities
- **y\_true** (*pd.Series*) – The ground truth for the predictions.
- **X** (*pd.DataFrame, optional*) – Any extra columns that are needed from training data.

**Returns** Optimal threshold for this objective

**positive\_only** (*cls*)

If True, this objective is only valid for positive data. Default False.

**score** (*self, y\_true, y\_predicted, X=None, sample\_weight=None*)

Returns a numerical score indicating performance based on the differences between the predicted and actual values.

**Parameters**

- **y\_predicted** (*pd.Series*) – Predicted values of length [n\_samples]
- **y\_true** (*pd.Series*) – Actual class labels of length [n\_samples]
- **X** (*pd.DataFrame or np.ndarray*) – Extra data of shape [n\_samples, n\_features] necessary to calculate score
- **sample\_weight** (*pd.DataFrame or np.ndarray*) – Sample weights used in computing objective value result

**Returns** score

**validate\_inputs** (*self, y\_true, y\_predicted*)

Validates the input based on a few simple checks.

**Parameters**

- **y\_predicted** (*pd.Series, or pd.DataFrame*) – Predicted values of length [n\_samples]
- **y\_true** (*pd.Series*) – Actual class labels of length [n\_samples]

**Returns** None

**class** evalml.objectives.**ExpVariance**

Explained variance score for regression.

**Attributes**

<b>greater_is_better</b>	True
<b>is_bounded_like_percentage</b>	False
<b>name</b>	ExpVariance
<b>perfect_score</b>	1.0
<b>problem_types</b>	[ProblemTypes.REGRESSION, ProblemTypes.TIME_SERIES_REGRESSION]
<b>score_needs_proba</b>	False

## Methods

<i>calculate_percent_difference</i>	Calculate the percent difference between scores.
<i>is_defined_for_problem_type</i>	
<i>objective_function</i>	Computes the relative value of the provided predictions compared to the actual labels, according a specified metric
<i>positive_only</i>	If True, this objective is only valid for positive data. Default False.
<i>score</i>	Returns a numerical score indicating performance based on the differences between the predicted and actual values.
<i>validate_inputs</i>	Validates the input based on a few simple checks.

**classmethod** `calculate_percent_difference` (*cls*, *score*, *baseline\_score*)

Calculate the percent difference between scores.

### Parameters

- **score** (*float*) – A score. Output of the score method of this objective.
- **baseline\_score** (*float*) – A score. Output of the score method of this objective. In practice, this is the score achieved on this objective with a baseline estimator.

### Returns

**The percent difference between the scores. Note that for objectives that can be interpreted as percentages, this will be the difference between the reference score and score. For all other objectives, the difference will be normalized by the reference score.**

**Return type** float

**classmethod** `is_defined_for_problem_type` (*cls*, *problem\_type*)

**objective\_function** (*self*, *y\_true*, *y\_predicted*, *X=None*, *sample\_weight=None*)

Computes the relative value of the provided predictions compared to the actual labels, according a specified metric

**Arguments:** *y\_predicted* (pd.Series): Predicted values of length [n\_samples] *y\_true* (pd.Series): Actual class labels of length [n\_samples] *X* (pd.DataFrame or np.ndarray): Extra data of shape [n\_samples, n\_features] necessary to calculate score *sample\_weight* (pd.DataFrame or np.ndarray): Sample weights used in computing objective value result

**Returns** Numerical value used to calculate score

**positive\_only** (*cls*)

If True, this objective is only valid for positive data. Default False.

**score** (*self, y\_true, y\_predicted, X=None, sample\_weight=None*)

Returns a numerical score indicating performance based on the differences between the predicted and actual values.

#### Parameters

- **y\_predicted** (*pd.Series*) – Predicted values of length [n\_samples]
- **y\_true** (*pd.Series*) – Actual class labels of length [n\_samples]
- **X** (*pd.DataFrame* or *np.ndarray*) – Extra data of shape [n\_samples, n\_features] necessary to calculate score
- **sample\_weight** (*pd.DataFrame* or *np.ndarray*) – Sample weights used in computing objective value result

**Returns** score

**validate\_inputs** (*self, y\_true, y\_predicted*)

Validates the input based on a few simple checks.

#### Parameters

- **y\_predicted** (*pd.Series*, or *pd.DataFrame*) – Predicted values of length [n\_samples]
- **y\_true** (*pd.Series*) – Actual class labels of length [n\_samples]

**Returns** None

**class** evalml.objectives.**F1**

F1 score for binary classification.

#### Attributes

<b>greater_is_better</b>	True
<b>is_bounded_like_percentage</b>	True
<b>name</b>	F1
<b>perfect_score</b>	1.0
<b>problem_types</b>	[ProblemTypes.BINARY, ProblemTypes.TIME_SERIES_BINARY]
<b>score_needs_proba</b>	False

#### Methods

<i>calculate_percent_difference</i>	Calculate the percent difference between scores.
<i>can_optimize_threshold</i>	Returns a boolean determining if we can optimize the binary classification objective threshold.
<i>decision_function</i>	Apply a learned threshold to predicted probabilities to get predicted classes.
<i>is_defined_for_problem_type</i>	
<i>objective_function</i>	Computes the relative value of the provided predictions compared to the actual labels, according a specified metric

continues on next page

Table 258 – continued from previous page

<code>optimize_threshold</code>	Learn a binary classification threshold which optimizes the current objective.
<code>positive_only</code>	If True, this objective is only valid for positive data. Default False.
<code>score</code>	Returns a numerical score indicating performance based on the differences between the predicted and actual values.
<code>validate_inputs</code>	Validates the input based on a few simple checks.

**classmethod** `calculate_percent_difference` (*cls*, *score*, *baseline\_score*)

Calculate the percent difference between scores.

#### Parameters

- **score** (*float*) – A score. Output of the score method of this objective.
- **baseline\_score** (*float*) – A score. Output of the score method of this objective. In practice, this is the score achieved on this objective with a baseline estimator.

#### Returns

**The percent difference between the scores. Note that for objectives that can be interpreted as percentages, this will be the difference between the reference score and score. For all other objectives, the difference will be normalized by the reference score.**

**Return type** float

**property** `can_optimize_threshold` (*cls*)

Returns a boolean determining if we can optimize the binary classification objective threshold. This will be false for any objective that works directly with predicted probabilities, like log loss and AUC. Otherwise, it will be true.

**decision\_function** (*self*, *ypred\_proba*, *threshold=0.5*, *X=None*)

Apply a learned threshold to predicted probabilities to get predicted classes.

#### Parameters

- **ypred\_proba** (*pd.Series*, *np.ndarray*) – The classifier’s predicted probabilities
- **threshold** (*float*, *optional*) – Threshold used to make a prediction. Defaults to 0.5.
- **X** (*pd.DataFrame*, *optional*) – Any extra columns that are needed from training data.

**Returns** predictions

**classmethod** `is_defined_for_problem_type` (*cls*, *problem\_type*)

**objective\_function** (*self*, *y\_true*, *y\_predicted*, *X=None*, *sample\_weight=None*)

Computes the relative value of the provided predictions compared to the actual labels, according a specified metric

**Arguments:** *y\_predicted* (*pd.Series*): Predicted values of length [*n\_samples*] *y\_true* (*pd.Series*): Actual class labels of length [*n\_samples*] *X* (*pd.DataFrame* or *np.ndarray*): Extra data of shape [*n\_samples*, *n\_features*] necessary to calculate score *sample\_weight* (*pd.DataFrame* or *np.ndarray*): Sample weights used in computing objective value result

**Returns** Numerical value used to calculate score



**optimize\_threshold** (*self*, *ypred\_proba*, *y\_true*, *X=None*)

Learn a binary classification threshold which optimizes the current objective.

**Parameters**

- **ypred\_proba** (*pd.Series*) – The classifier’s predicted probabilities
- **y\_true** (*pd.Series*) – The ground truth for the predictions.
- **X** (*pd.DataFrame*, *optional*) – Any extra columns that are needed from training data.

**Returns** Optimal threshold for this objective

**positive\_only** (*cls*)

If True, this objective is only valid for positive data. Default False.

**score** (*self*, *y\_true*, *y\_predicted*, *X=None*, *sample\_weight=None*)

Returns a numerical score indicating performance based on the differences between the predicted and actual values.

**Parameters**

- **y\_predicted** (*pd.Series*) – Predicted values of length [n\_samples]
- **y\_true** (*pd.Series*) – Actual class labels of length [n\_samples]
- **X** (*pd.DataFrame* or *np.ndarray*) – Extra data of shape [n\_samples, n\_features] necessary to calculate score
- **sample\_weight** (*pd.DataFrame* or *np.ndarray*) – Sample weights used in computing objective value result

**Returns** score

**validate\_inputs** (*self*, *y\_true*, *y\_predicted*)

Validates the input based on a few simple checks.

**Parameters**

- **y\_predicted** (*pd.Series*, or *pd.DataFrame*) – Predicted values of length [n\_samples]
- **y\_true** (*pd.Series*) – Actual class labels of length [n\_samples]

**Returns** None

**class** evalml.objectives.**F1Macro**

F1 score for multiclass classification using macro averaging.

**Attributes**

<b>greater_is_better</b>	True
<b>is_bounded_like_percentage</b>	True
<b>name</b>	F1 Macro
<b>perfect_score</b>	1.0
<b>problem_types</b>	[ProblemTypes.MULTICLASS, ProblemTypes.TIME_SERIES_MULTICLASS]
<b>score_needs_proba</b>	False

**Methods**

<code>calculate_percent_difference</code>	Calculate the percent difference between scores.
<code>is_defined_for_problem_type</code>	
<code>objective_function</code>	Computes the relative value of the provided predictions compared to the actual labels, according a specified metric
<code>positive_only</code>	If True, this objective is only valid for positive data. Default False.
<code>score</code>	Returns a numerical score indicating performance based on the differences between the predicted and actual values.
<code>validate_inputs</code>	Validates the input based on a few simple checks.

**classmethod** `calculate_percent_difference` (*cls*, *score*, *baseline\_score*)

Calculate the percent difference between scores.

#### Parameters

- **score** (*float*) – A score. Output of the score method of this objective.
- **baseline\_score** (*float*) – A score. Output of the score method of this objective. In practice, this is the score achieved on this objective with a baseline estimator.

#### Returns

**The percent difference between the scores. Note that for objectives that can be interpreted** as percentages, this will be the difference between the reference score and score. For all other objectives, the difference will be normalized by the reference score.

**Return type** float

**classmethod** `is_defined_for_problem_type` (*cls*, *problem\_type*)

**objective\_function** (*self*, *y\_true*, *y\_predicted*, *X=None*, *sample\_weight=None*)

Computes the relative value of the provided predictions compared to the actual labels, according a specified metric

**Arguments:** *y\_predicted* (*pd.Series*): Predicted values of length [*n\_samples*] *y\_true* (*pd.Series*): Actual class labels of length [*n\_samples*] *X* (*pd.DataFrame* or *np.ndarray*): Extra data of shape [*n\_samples*, *n\_features*] necessary to calculate score *sample\_weight* (*pd.DataFrame* or *np.ndarray*): Sample weights used in computing objective value result

**Returns** Numerical value used to calculate score

**positive\_only** (*cls*)

If True, this objective is only valid for positive data. Default False.

**score** (*self*, *y\_true*, *y\_predicted*, *X=None*, *sample\_weight=None*)

Returns a numerical score indicating performance based on the differences between the predicted and actual values.

#### Parameters

- **y\_predicted** (*pd.Series*) – Predicted values of length [*n\_samples*]
- **y\_true** (*pd.Series*) – Actual class labels of length [*n\_samples*]
- **X** (*pd.DataFrame* or *np.ndarray*) – Extra data of shape [*n\_samples*, *n\_features*] necessary to calculate score

- **sample\_weight** (*pd.DataFrame* or *np.ndarray*) – Sample weights used in computing objective value result

**Returns** score

**validate\_inputs** (*self*, *y\_true*, *y\_predicted*)

Validates the input based on a few simple checks.

**Parameters**

- **y\_predicted** (*pd.Series*, or *pd.DataFrame*) – Predicted values of length [n\_samples]
- **y\_true** (*pd.Series*) – Actual class labels of length [n\_samples]

**Returns** None

**class** evalml.objectives.F1Micro

F1 score for multiclass classification using micro averaging.

**Attributes**

<b>greater_is_better</b>	True
<b>is_bounded_like_percentage</b>	True
<b>name</b>	F1 Micro
<b>perfect_score</b>	1.0
<b>problem_types</b>	[ProblemTypes.MULTICLASS, ProblemTypes.TIME_SERIES_MULTICLASS]
<b>score_needs_proba</b>	False

**Methods**

<i>calculate_percent_difference</i>	Calculate the percent difference between scores.
<i>is_defined_for_problem_type</i>	
<i>objective_function</i>	Computes the relative value of the provided predictions compared to the actual labels, according a specified metric
<i>positive_only</i>	If True, this objective is only valid for positive data. Default False.
<i>score</i>	Returns a numerical score indicating performance based on the differences between the predicted and actual values.
<i>validate_inputs</i>	Validates the input based on a few simple checks.

**classmethod** **calculate\_percent\_difference** (*cls*, *score*, *baseline\_score*)

Calculate the percent difference between scores.

**Parameters**

- **score** (*float*) – A score. Output of the score method of this objective.
- **baseline\_score** (*float*) – A score. Output of the score method of this objective. In practice, this is the score achieved on this objective with a baseline estimator.

**Returns**

The percent difference between the scores. Note that for objectives that can be interpreted

as percentages, this will be the difference between the reference score and score. For all other objectives, the difference will be normalized by the reference score.

**Return type** float

**classmethod** `is_defined_for_problem_type(cls, problem_type)`

**objective\_function** (*self*, *y\_true*, *y\_predicted*, *X=None*, *sample\_weight=None*)

Computes the relative value of the provided predictions compared to the actual labels, according a specified metric

**Arguments:** *y\_predicted* (pd.Series): Predicted values of length [n\_samples] *y\_true* (pd.Series): Actual class labels of length [n\_samples] *X* (pd.DataFrame or np.ndarray): Extra data of shape [n\_samples, n\_features] necessary to calculate score *sample\_weight* (pd.DataFrame or np.ndarray): Sample weights used in computing objective value result

**Returns** Numerical value used to calculate score

**positive\_only** (*cls*)

If True, this objective is only valid for positive data. Default False.

**score** (*self*, *y\_true*, *y\_predicted*, *X=None*, *sample\_weight=None*)

Returns a numerical score indicating performance based on the differences between the predicted and actual values.

**Parameters**

- **y\_predicted** (pd.Series) – Predicted values of length [n\_samples]
- **y\_true** (pd.Series) – Actual class labels of length [n\_samples]
- **X** (pd.DataFrame or np.ndarray) – Extra data of shape [n\_samples, n\_features] necessary to calculate score
- **sample\_weight** (pd.DataFrame or np.ndarray) – Sample weights used in computing objective value result

**Returns** score

**validate\_inputs** (*self*, *y\_true*, *y\_predicted*)

Validates the input based on a few simple checks.

**Parameters**

- **y\_predicted** (pd.Series, or pd.DataFrame) – Predicted values of length [n\_samples]
- **y\_true** (pd.Series) – Actual class labels of length [n\_samples]

**Returns** None

**class** evalml.objectives.F1Weighted

F1 score for multiclass classification using weighted averaging.

**Attributes**

<b>greater_is_better</b>	True
<b>is_bounded_like_percentage</b>	True
<b>name</b>	F1 Weighted
<b>perfect_score</b>	1.0
<b>problem_types</b>	[ProblemTypes.MULTICLASS, ProblemTypes.TIME_SERIES_MULTICLASS]
<b>score_needs_proba</b>	False

## Methods

<i>calculate_percent_difference</i>	Calculate the percent difference between scores.
<i>is_defined_for_problem_type</i>	
<i>objective_function</i>	Computes the relative value of the provided predictions compared to the actual labels, according a specified metric
<i>positive_only</i>	If True, this objective is only valid for positive data. Default False.
<i>score</i>	Returns a numerical score indicating performance based on the differences between the predicted and actual values.
<i>validate_inputs</i>	Validates the input based on a few simple checks.

**classmethod** `calculate_percent_difference` (*cls*, *score*, *baseline\_score*)

Calculate the percent difference between scores.

### Parameters

- **score** (*float*) – A score. Output of the score method of this objective.
- **baseline\_score** (*float*) – A score. Output of the score method of this objective. In practice, this is the score achieved on this objective with a baseline estimator.

### Returns

**The percent difference between the scores. Note that for objectives that can be interpreted as percentages, this will be the difference between the reference score and score. For all other objectives, the difference will be normalized by the reference score.**

**Return type** float

**classmethod** `is_defined_for_problem_type` (*cls*, *problem\_type*)

**objective\_function** (*self*, *y\_true*, *y\_predicted*, *X=None*, *sample\_weight=None*)

Computes the relative value of the provided predictions compared to the actual labels, according a specified metric

**Arguments:** *y\_predicted* (pd.Series): Predicted values of length [n\_samples] *y\_true* (pd.Series): Actual class labels of length [n\_samples] *X* (pd.DataFrame or np.ndarray): Extra data of shape [n\_samples, n\_features] necessary to calculate score *sample\_weight* (pd.DataFrame or np.ndarray): Sample weights used in computing objective value result

**Returns** Numerical value used to calculate score

**positive\_only** (*cls*)

If True, this objective is only valid for positive data. Default False.

**score** (*self, y\_true, y\_predicted, X=None, sample\_weight=None*)

Returns a numerical score indicating performance based on the differences between the predicted and actual values.

#### Parameters

- **y\_predicted** (*pd.Series*) – Predicted values of length [n\_samples]
- **y\_true** (*pd.Series*) – Actual class labels of length [n\_samples]
- **X** (*pd.DataFrame* or *np.ndarray*) – Extra data of shape [n\_samples, n\_features] necessary to calculate score
- **sample\_weight** (*pd.DataFrame* or *np.ndarray*) – Sample weights used in computing objective value result

**Returns** score

**validate\_inputs** (*self, y\_true, y\_predicted*)

Validates the input based on a few simple checks.

#### Parameters

- **y\_predicted** (*pd.Series*, or *pd.DataFrame*) – Predicted values of length [n\_samples]
- **y\_true** (*pd.Series*) – Actual class labels of length [n\_samples]

**Returns** None

**class** evalml.objectives.**FraudCost** (*retry\_percentage=0.5*, *interchange\_fee=0.02*,  
*fraud\_payout\_percentage=1.0, amount\_col='amount'*)

Score the percentage of money lost of the total transaction amount process due to fraud.

#### Parameters

- **retry\_percentage** (*float*) – What percentage of customers that will retry a transaction if it is declined. Between 0 and 1. Defaults to 0.5.
- **interchange\_fee** (*float*) – How much of each successful transaction you pay. Between 0 and 1. Defaults to 0.02.
- **fraud\_payout\_percentage** (*float*) – Percentage of fraud you will not be able to collect. Between 0 and 1. Defaults to 1.0.
- **amount\_col** (*str*) – Name of column in data that contains the amount. Defaults to “amount”.

#### Attributes

<b>greater_is_better</b>	False
<b>is_bounded_like_percentage</b>	True
<b>name</b>	Fraud Cost
<b>perfect_score</b>	0.0
<b>problem_types</b>	[ProblemTypes.BINARY, ProblemTypes.TIME_SERIES_BINARY]
<b>score_needs_proba</b>	False

#### Methods

<code>calculate_percent_difference</code>	Calculate the percent difference between scores.
<code>can_optimize_threshold</code>	Returns a boolean determining if we can optimize the binary classification objective threshold.
<code>decision_function</code>	Apply a learned threshold to predicted probabilities to get predicted classes.
<code>is_defined_for_problem_type</code>	
<code>objective_function</code>	Calculate amount lost to fraud per transaction given predictions, true values, and dataframe with transaction amount.
<code>optimize_threshold</code>	Learn a binary classification threshold which optimizes the current objective.
<code>positive_only</code>	If True, this objective is only valid for positive data. Default False.
<code>score</code>	Returns a numerical score indicating performance based on the differences between the predicted and actual values.
<code>validate_inputs</code>	Validates the input based on a few simple checks.

**classmethod** `calculate_percent_difference` (*cls*, *score*, *baseline\_score*)

Calculate the percent difference between scores.

#### Parameters

- **score** (*float*) – A score. Output of the score method of this objective.
- **baseline\_score** (*float*) – A score. Output of the score method of this objective. In practice, this is the score achieved on this objective with a baseline estimator.

#### Returns

**The percent difference between the scores. Note that for objectives that can be interpreted as percentages, this will be the difference between the reference score and score. For all other objectives, the difference will be normalized by the reference score.**

**Return type** float

**property** `can_optimize_threshold` (*cls*)

Returns a boolean determining if we can optimize the binary classification objective threshold. This will be false for any objective that works directly with predicted probabilities, like log loss and AUC. Otherwise, it will be true.

**decision\_function** (*self*, *ypred\_proba*, *threshold=0.5*, *X=None*)

Apply a learned threshold to predicted probabilities to get predicted classes.

#### Parameters

- **ypred\_proba** (*pd.Series*, *np.ndarray*) – The classifier’s predicted probabilities
- **threshold** (*float*, *optional*) – Threshold used to make a prediction. Defaults to 0.5.
- **X** (*pd.DataFrame*, *optional*) – Any extra columns that are needed from training data.

**Returns** predictions

**classmethod** `is_defined_for_problem_type` (*cls*, *problem\_type*)

**objective\_function** (*self*, *y\_true*, *y\_predicted*, *X*, *sample\_weight=None*)

Calculate amount lost to fraud per transaction given predictions, true values, and dataframe with transaction amount.

**Parameters**

- **y\_predicted** (*pd.Series*) – Predicted fraud labels
- **y\_true** (*pd.Series*) – True fraud labels
- **X** (*pd.DataFrame*) – Data with transaction amounts
- **sample\_weight** (*pd.DataFrame*) – Ignored.

**Returns** Amount lost to fraud per transaction

**Return type** float

**optimize\_threshold** (*self*, *ypred\_proba*, *y\_true*, *X=None*)

Learn a binary classification threshold which optimizes the current objective.

**Parameters**

- **ypred\_proba** (*pd.Series*) – The classifier’s predicted probabilities
- **y\_true** (*pd.Series*) – The ground truth for the predictions.
- **X** (*pd.DataFrame*, *optional*) – Any extra columns that are needed from training data.

**Returns** Optimal threshold for this objective

**positive\_only** (*cls*)

If True, this objective is only valid for positive data. Default False.

**score** (*self*, *y\_true*, *y\_predicted*, *X=None*, *sample\_weight=None*)

Returns a numerical score indicating performance based on the differences between the predicted and actual values.

**Parameters**

- **y\_predicted** (*pd.Series*) – Predicted values of length [n\_samples]
- **y\_true** (*pd.Series*) – Actual class labels of length [n\_samples]
- **X** (*pd.DataFrame* or *np.ndarray*) – Extra data of shape [n\_samples, n\_features] necessary to calculate score
- **sample\_weight** (*pd.DataFrame* or *np.ndarray*) – Sample weights used in computing objective value result

**Returns** score

**validate\_inputs** (*self*, *y\_true*, *y\_predicted*)

Validates the input based on a few simple checks.

**Parameters**

- **y\_predicted** (*pd.Series*, or *pd.DataFrame*) – Predicted values of length [n\_samples]
- **y\_true** (*pd.Series*) – Actual class labels of length [n\_samples]

**Returns** None

**evalml.objectives.get\_all\_objective\_names** ()

Get a list of the names of all objectives.



**Returns** Objective names

**Return type** list (str)

```
evalml.objectives.get_core_objective_names()
```

Get a list of all valid core objectives.

**Returns** Objective names.

**Return type** list[str]

```
evalml.objectives.get_core_objectives(problem_type)
```

Returns all core objective instances associated with the given problem type.

Core objectives are designed to work out-of-the-box for any dataset.

**Parameters** `problem_type` (str/*ProblemTypes*) – Type of problem

**Returns** List of ObjectiveBase instances

```
evalml.objectives.get_non_core_objectives()
```

Get non-core objective classes.

Non-core objectives are objectives that are domain-specific. Users typically need to configure these objectives before using them in AutoMLSearch.

**Returns** List of ObjectiveBase classes

```
evalml.objectives.get_objective(objective, return_instance=False, **kwargs)
```

Returns the Objective class corresponding to a given objective name.

**Parameters**

- **objective** (str or ObjectiveBase) – Name or instance of the objective class.
- **return\_instance** (bool) – Whether to return an instance of the objective. This only applies if objective is of type str. Note that the instance will be initialized with default arguments.
- **kwargs** (Any) – Any keyword arguments to pass into the objective. Only used when return\_instance=True.

**Returns** ObjectiveBase if the parameter objective is of type ObjectiveBase. If objective is instead a valid objective name, function will return the class corresponding to that name. If return\_instance is True, an instance of that objective will be returned.

```
class evalml.objectives.Gini
```

Gini coefficient for binary classification.

**Attributes**

<b>greater_is_better</b>	True
<b>is_bounded_like_percentage</b>	False
<b>name</b>	Gini
<b>perfect_score</b>	1.0
<b>problem_types</b>	[ProblemTypes.BINARY, ProblemTypes.TIME_SERIES_BINARY]
<b>score_needs_proba</b>	True

**Methods**

<code>calculate_percent_difference</code>	Calculate the percent difference between scores.
<code>can_optimize_threshold</code>	Returns a boolean determining if we can optimize the binary classification objective threshold.
<code>decision_function</code>	Apply a learned threshold to predicted probabilities to get predicted classes.
<code>is_defined_for_problem_type</code>	
<code>objective_function</code>	Computes the relative value of the provided predictions compared to the actual labels, according a specified metric
<code>optimize_threshold</code>	Learn a binary classification threshold which optimizes the current objective.
<code>positive_only</code>	If True, this objective is only valid for positive data. Default False.
<code>score</code>	Returns a numerical score indicating performance based on the differences between the predicted and actual values.
<code>validate_inputs</code>	Validates the input based on a few simple checks.

**classmethod** `calculate_percent_difference` (*cls*, *score*, *baseline\_score*)

Calculate the percent difference between scores.

#### Parameters

- **score** (*float*) – A score. Output of the score method of this objective.
- **baseline\_score** (*float*) – A score. Output of the score method of this objective. In practice, this is the score achieved on this objective with a baseline estimator.

#### Returns

**The percent difference between the scores. Note that for objectives that can be interpreted as percentages, this will be the difference between the reference score and score. For all other objectives, the difference will be normalized by the reference score.**

**Return type** float

**property** `can_optimize_threshold` (*cls*)

Returns a boolean determining if we can optimize the binary classification objective threshold. This will be false for any objective that works directly with predicted probabilities, like log loss and AUC. Otherwise, it will be true.

**decision\_function** (*self*, *ypred\_proba*, *threshold=0.5*, *X=None*)

Apply a learned threshold to predicted probabilities to get predicted classes.

#### Parameters

- **ypred\_proba** (*pd.Series*, *np.ndarray*) – The classifier’s predicted probabilities
- **threshold** (*float*, *optional*) – Threshold used to make a prediction. Defaults to 0.5.
- **X** (*pd.DataFrame*, *optional*) – Any extra columns that are needed from training data.

**Returns** predictions

**classmethod** `is_defined_for_problem_type` (*cls*, *problem\_type*)

**objective\_function** (*self*, *y\_true*, *y\_predicted*, *X=None*, *sample\_weight=None*)

Computes the relative value of the provided predictions compared to the actual labels, according a specified metric

**Arguments:** *y\_predicted* (*pd.Series*): Predicted values of length [*n\_samples*] *y\_true* (*pd.Series*): Actual class labels of length [*n\_samples*] *X* (*pd.DataFrame* or *np.ndarray*): Extra data of shape [*n\_samples*, *n\_features*] necessary to calculate score *sample\_weight* (*pd.DataFrame* or *np.ndarray*): Sample weights used in computing objective value result

**Returns** Numerical value used to calculate score

**optimize\_threshold** (*self*, *ypred\_proba*, *y\_true*, *X=None*)

Learn a binary classification threshold which optimizes the current objective.

**Parameters**

- **ypred\_proba** (*pd.Series*) – The classifier’s predicted probabilities
- **y\_true** (*pd.Series*) – The ground truth for the predictions.
- **X** (*pd.DataFrame*, *optional*) – Any extra columns that are needed from training data.

**Returns** Optimal threshold for this objective

**positive\_only** (*cls*)

If True, this objective is only valid for positive data. Default False.

**score** (*self*, *y\_true*, *y\_predicted*, *X=None*, *sample\_weight=None*)

Returns a numerical score indicating performance based on the differences between the predicted and actual values.

**Parameters**

- **y\_predicted** (*pd.Series*) – Predicted values of length [*n\_samples*]
- **y\_true** (*pd.Series*) – Actual class labels of length [*n\_samples*]
- **X** (*pd.DataFrame* or *np.ndarray*) – Extra data of shape [*n\_samples*, *n\_features*] necessary to calculate score
- **sample\_weight** (*pd.DataFrame* or *np.ndarray*) – Sample weights used in computing objective value result

**Returns** score

**validate\_inputs** (*self*, *y\_true*, *y\_predicted*)

Validates the input based on a few simple checks.

**Parameters**

- **y\_predicted** (*pd.Series*, or *pd.DataFrame*) – Predicted values of length [*n\_samples*]
- **y\_true** (*pd.Series*) – Actual class labels of length [*n\_samples*]

**Returns** None

**class** evalml.objectives.**LeadScoring** (*true\_positives=1*, *false\_positives=-1*)

Lead scoring.

**Parameters**

- **true\_positives** (*int*) – Reward for a true positive. Defaults to 1.

- **false\_positives** (*int*) – Cost for a false positive. Should be negative. Defaults to -1.

#### Attributes

<b>greater_is_better</b>	True
<b>is_bounded_like_percentage</b>	False
<b>name</b>	Lead Scoring
<b>perfect_score</b>	None
<b>problem_types</b>	[ProblemTypes.BINARY, ProblemTypes.TIME_SERIES_BINARY]
<b>score_needs_proba</b>	True

#### Methods

<i>calculate_percent_difference</i>	Calculate the percent difference between scores.
<i>can_optimize_threshold</i>	Returns a boolean determining if we can optimize the binary classification objective threshold.
<i>decision_function</i>	Apply a learned threshold to predicted probabilities to get predicted classes.
<i>is_defined_for_problem_type</i>	
<i>objective_function</i>	Calculate the profit per lead.
<i>optimize_threshold</i>	Learn a binary classification threshold which optimizes the current objective.
<i>positive_only</i>	If True, this objective is only valid for positive data. Default False.
<i>score</i>	Returns a numerical score indicating performance based on the differences between the predicted and actual values.
<i>validate_inputs</i>	Validates the input based on a few simple checks.

**classmethod** **calculate\_percent\_difference** (*cls*, *score*, *baseline\_score*)

Calculate the percent difference between scores.

#### Parameters

- **score** (*float*) – A score. Output of the score method of this objective.
- **baseline\_score** (*float*) – A score. Output of the score method of this objective. In practice, this is the score achieved on this objective with a baseline estimator.

#### Returns

**The percent difference between the scores. Note that for objectives that can be interpreted as percentages, this will be the difference between the reference score and score. For all other objectives, the difference will be normalized by the reference score.**

**Return type** float

**property** **can\_optimize\_threshold** (*cls*)

Returns a boolean determining if we can optimize the binary classification objective threshold. This will be false for any objective that works directly with predicted probabilities, like log loss and AUC. Otherwise, it will be true.

**decision\_function** (*self*, *ypred\_proba*, *threshold=0.5*, *X=None*)

Apply a learned threshold to predicted probabilities to get predicted classes.

**Parameters**

- **ypred\_proba** (*pd.Series, np.ndarray*) – The classifier’s predicted probabilities
- **threshold** (*float, optional*) – Threshold used to make a prediction. Defaults to 0.5.
- **X** (*pd.DataFrame, optional*) – Any extra columns that are needed from training data.

**Returns** predictions**classmethod is\_defined\_for\_problem\_type** (*cls, problem\_type*)**objective\_function** (*self, y\_true, y\_predicted, X=None, sample\_weight=None*)

Calculate the profit per lead.

**Parameters**

- **y\_predicted** (*pd.Series*) – Predicted labels
- **y\_true** (*pd.Series*) – True labels
- **X** (*pd.DataFrame*) – Ignored.
- **sample\_weight** (*pd.DataFrame*) – Ignored.

**Returns** Profit per lead**Return type** float**optimize\_threshold** (*self, ypred\_proba, y\_true, X=None*)

Learn a binary classification threshold which optimizes the current objective.

**Parameters**

- **ypred\_proba** (*pd.Series*) – The classifier’s predicted probabilities
- **y\_true** (*pd.Series*) – The ground truth for the predictions.
- **X** (*pd.DataFrame, optional*) – Any extra columns that are needed from training data.

**Returns** Optimal threshold for this objective**positive\_only** (*cls*)

If True, this objective is only valid for positive data. Default False.

**score** (*self, y\_true, y\_predicted, X=None, sample\_weight=None*)

Returns a numerical score indicating performance based on the differences between the predicted and actual values.

**Parameters**

- **y\_predicted** (*pd.Series*) – Predicted values of length [n\_samples]
- **y\_true** (*pd.Series*) – Actual class labels of length [n\_samples]
- **X** (*pd.DataFrame or np.ndarray*) – Extra data of shape [n\_samples, n\_features] necessary to calculate score
- **sample\_weight** (*pd.DataFrame or np.ndarray*) – Sample weights used in computing objective value result

**Returns** score

**validate\_inputs** (*self*, *y\_true*, *y\_predicted*)

Validates the input based on a few simple checks.

#### Parameters

- **y\_predicted** (*pd.Series*, or *pd.DataFrame*) – Predicted values of length [n\_samples]
- **y\_true** (*pd.Series*) – Actual class labels of length [n\_samples]

**Returns** None

**class** evalml.objectives.**LogLossBinary**

Log Loss for binary classification.

#### Attributes

<b>greater_is_better</b>	False
<b>is_bounded_like_percentage</b>	True
<b>name</b>	Log Loss Binary
<b>perfect_score</b>	0.0
<b>problem_types</b>	[ProblemTypes.BINARY, ProblemTypes.TIME_SERIES_BINARY]
<b>score_needs_proba</b>	True

#### Methods

<i>calculate_percent_difference</i>	Calculate the percent difference between scores.
<i>can_optimize_threshold</i>	Returns a boolean determining if we can optimize the binary classification objective threshold.
<i>decision_function</i>	Apply a learned threshold to predicted probabilities to get predicted classes.
<i>is_defined_for_problem_type</i>	
<i>objective_function</i>	Computes the relative value of the provided predictions compared to the actual labels, according a specified metric
<i>optimize_threshold</i>	Learn a binary classification threshold which optimizes the current objective.
<i>positive_only</i>	If True, this objective is only valid for positive data. Default False.
<i>score</i>	Returns a numerical score indicating performance based on the differences between the predicted and actual values.
<i>validate_inputs</i>	Validates the input based on a few simple checks.

**classmethod** **calculate\_percent\_difference** (*cls*, *score*, *baseline\_score*)

Calculate the percent difference between scores.

#### Parameters

- **score** (*float*) – A score. Output of the score method of this objective.
- **baseline\_score** (*float*) – A score. Output of the score method of this objective. In practice, this is the score achieved on this objective with a baseline estimator.

**Returns**

**The percent difference between the scores.** Note that for objectives that can be interpreted as percentages, this will be the difference between the reference score and score. For all other objectives, the difference will be normalized by the reference score.

**Return type** float

**property can\_optimize\_threshold** (*cls*)

Returns a boolean determining if we can optimize the binary classification objective threshold. This will be false for any objective that works directly with predicted probabilities, like log loss and AUC. Otherwise, it will be true.

**decision\_function** (*self*, *ypred\_proba*, *threshold=0.5*, *X=None*)

Apply a learned threshold to predicted probabilities to get predicted classes.

**Parameters**

- **ypred\_proba** (*pd.Series*, *np.ndarray*) – The classifier’s predicted probabilities
- **threshold** (*float*, *optional*) – Threshold used to make a prediction. Defaults to 0.5.
- **X** (*pd.DataFrame*, *optional*) – Any extra columns that are needed from training data.

**Returns** predictions

**classmethod is\_defined\_for\_problem\_type** (*cls*, *problem\_type*)

**objective\_function** (*self*, *y\_true*, *y\_predicted*, *X=None*, *sample\_weight=None*)

Computes the relative value of the provided predictions compared to the actual labels, according a specified metric

**Arguments:** *y\_predicted* (*pd.Series*): Predicted values of length [*n\_samples*] *y\_true* (*pd.Series*): Actual class labels of length [*n\_samples*] *X* (*pd.DataFrame* or *np.ndarray*): Extra data of shape [*n\_samples*, *n\_features*] necessary to calculate score *sample\_weight* (*pd.DataFrame* or *np.ndarray*): Sample weights used in computing objective value result

**Returns** Numerical value used to calculate score

**optimize\_threshold** (*self*, *ypred\_proba*, *y\_true*, *X=None*)

Learn a binary classification threshold which optimizes the current objective.

**Parameters**

- **ypred\_proba** (*pd.Series*) – The classifier’s predicted probabilities
- **y\_true** (*pd.Series*) – The ground truth for the predictions.
- **X** (*pd.DataFrame*, *optional*) – Any extra columns that are needed from training data.

**Returns** Optimal threshold for this objective

**positive\_only** (*cls*)

If True, this objective is only valid for positive data. Default False.

**score** (*self*, *y\_true*, *y\_predicted*, *X=None*, *sample\_weight=None*)

Returns a numerical score indicating performance based on the differences between the predicted and actual values.

**Parameters**

- **y\_predicted** (*pd.Series*) – Predicted values of length [n\_samples]
- **y\_true** (*pd.Series*) – Actual class labels of length [n\_samples]
- **x** (*pd.DataFrame* or *np.ndarray*) – Extra data of shape [n\_samples, n\_features] necessary to calculate score
- **sample\_weight** (*pd.DataFrame* or *np.ndarray*) – Sample weights used in computing objective value result

**Returns** score

**validate\_inputs** (*self, y\_true, y\_predicted*)

Validates the input based on a few simple checks.

**Parameters**

- **y\_predicted** (*pd.Series*, or *pd.DataFrame*) – Predicted values of length [n\_samples]
- **y\_true** (*pd.Series*) – Actual class labels of length [n\_samples]

**Returns** None

**class** evalml.objectives.**LogLossMulticlass**

Log Loss for multiclass classification.

**Attributes**

<b>greater_is_better</b>	False
<b>is_bounded_like_percentage</b>	False
<b>name</b>	Log Loss Multiclass
<b>perfect_score</b>	0.0
<b>problem_types</b>	[ProblemTypes.MULTICLASS, ProblemTypes.TIME_SERIES_MULTICLASS]
<b>score_needs_proba</b>	True

**Methods**

<i>calculate_percent_difference</i>	Calculate the percent difference between scores.
<i>is_defined_for_problem_type</i>	
<i>objective_function</i>	Computes the relative value of the provided predictions compared to the actual labels, according a specified metric
<i>positive_only</i>	If True, this objective is only valid for positive data. Default False.
<i>score</i>	Returns a numerical score indicating performance based on the differences between the predicted and actual values.
<i>validate_inputs</i>	Validates the input based on a few simple checks.

**classmethod** **calculate\_percent\_difference** (*cls, score, baseline\_score*)

Calculate the percent difference between scores.

**Parameters**

- **score** (*float*) – A score. Output of the score method of this objective.



- **baseline\_score** (*float*) – A score. Output of the score method of this objective. In practice, this is the score achieved on this objective with a baseline estimator.

#### Returns

**The percent difference between the scores.** Note that for objectives that can be interpreted as percentages, this will be the difference between the reference score and score. For all other objectives, the difference will be normalized by the reference score.

**Return type** float

**classmethod** **is\_defined\_for\_problem\_type** (*cls, problem\_type*)

**objective\_function** (*self, y\_true, y\_predicted, X=None, sample\_weight=None*)

Computes the relative value of the provided predictions compared to the actual labels, according a specified metric

**Arguments:** *y\_predicted* (*pd.Series*): Predicted values of length [*n\_samples*] *y\_true* (*pd.Series*): Actual class labels of length [*n\_samples*] *X* (*pd.DataFrame* or *np.ndarray*): Extra data of shape [*n\_samples*, *n\_features*] necessary to calculate score *sample\_weight* (*pd.DataFrame* or *np.ndarray*): Sample weights used in computing objective value result

**Returns** Numerical value used to calculate score

**positive\_only** (*cls*)

If True, this objective is only valid for positive data. Default False.

**score** (*self, y\_true, y\_predicted, X=None, sample\_weight=None*)

Returns a numerical score indicating performance based on the differences between the predicted and actual values.

#### Parameters

- **y\_predicted** (*pd.Series*) – Predicted values of length [*n\_samples*]
- **y\_true** (*pd.Series*) – Actual class labels of length [*n\_samples*]
- **X** (*pd.DataFrame* or *np.ndarray*) – Extra data of shape [*n\_samples*, *n\_features*] necessary to calculate score
- **sample\_weight** (*pd.DataFrame* or *np.ndarray*) – Sample weights used in computing objective value result

**Returns** score

**validate\_inputs** (*self, y\_true, y\_predicted*)

Validates the input based on a few simple checks.

#### Parameters

- **y\_predicted** (*pd.Series*, or *pd.DataFrame*) – Predicted values of length [*n\_samples*]
- **y\_true** (*pd.Series*) – Actual class labels of length [*n\_samples*]

**Returns** None

**class** evalml.objectives.**MAE**

Mean absolute error for regression.

#### Attributes

<b>greater_is_better</b>	False
<b>is_bounded_like_percentage</b>	True
<b>name</b>	MAE
<b>perfect_score</b>	0.0
<b>problem_types</b>	[ProblemTypes.REGRESSION, ProblemTypes.TIME_SERIES_REGRESSION]
<b>score_needs_positive</b>	False

## Methods

<i>calculate_percent_difference</i>	Calculate the percent difference between scores.
<i>is_defined_for_problem_type</i>	
<i>objective_function</i>	Computes the relative value of the provided predictions compared to the actual labels, according a specified metric
<i>positive_only</i>	If True, this objective is only valid for positive data. Default False.
<i>score</i>	Returns a numerical score indicating performance based on the differences between the predicted and actual values.
<i>validate_inputs</i>	Validates the input based on a few simple checks.

**classmethod** `calculate_percent_difference` (*cls*, *score*, *baseline\_score*)

Calculate the percent difference between scores.

### Parameters

- **score** (*float*) – A score. Output of the score method of this objective.
- **baseline\_score** (*float*) – A score. Output of the score method of this objective. In practice, this is the score achieved on this objective with a baseline estimator.

### Returns

**The percent difference between the scores. Note that for objectives that can be interpreted as percentages, this will be the difference between the reference score and score. For all other objectives, the difference will be normalized by the reference score.**

**Return type** float

**classmethod** `is_defined_for_problem_type` (*cls*, *problem\_type*)

**objective\_function** (*self*, *y\_true*, *y\_predicted*, *X=None*, *sample\_weight=None*)

Computes the relative value of the provided predictions compared to the actual labels, according a specified metric

**Arguments:** *y\_predicted* (pd.Series): Predicted values of length [n\_samples] *y\_true* (pd.Series): Actual class labels of length [n\_samples] *X* (pd.DataFrame or np.ndarray): Extra data of shape [n\_samples, n\_features] necessary to calculate score *sample\_weight* (pd.DataFrame or np.ndarray): Sample weights used in computing objective value result

**Returns** Numerical value used to calculate score

**positive\_only** (*cls*)

If True, this objective is only valid for positive data. Default False.

**score** (*self, y\_true, y\_predicted, X=None, sample\_weight=None*)

Returns a numerical score indicating performance based on the differences between the predicted and actual values.

#### Parameters

- **y\_predicted** (*pd.Series*) – Predicted values of length [n\_samples]
- **y\_true** (*pd.Series*) – Actual class labels of length [n\_samples]
- **X** (*pd.DataFrame or np.ndarray*) – Extra data of shape [n\_samples, n\_features] necessary to calculate score
- **sample\_weight** (*pd.DataFrame or np.ndarray*) – Sample weights used in computing objective value result

**Returns** score

**validate\_inputs** (*self, y\_true, y\_predicted*)

Validates the input based on a few simple checks.

#### Parameters

- **y\_predicted** (*pd.Series, or pd.DataFrame*) – Predicted values of length [n\_samples]
- **y\_true** (*pd.Series*) – Actual class labels of length [n\_samples]

**Returns** None

**class** evalml.objectives.**MAPE**

Mean absolute percentage error for time series regression. Scaled by 100 to return a percentage.

Only valid for nonzero inputs. Otherwise, will throw a ValueError

#### Attributes

<b>greater_is_better</b>	False
<b>is_bounded_like_percentage</b>	True
<b>name</b>	Mean Absolute Percentage Error
<b>perfect_score</b>	0.0
<b>problem_types</b>	[ProblemTypes.TIME_SERIES_REGRESSION]
<b>score_needs_problem</b>	True

#### Methods

<a href="#"><i>calculate_percent_difference</i></a>	Calculate the percent difference between scores.
<a href="#"><i>is_defined_for_problem_type</i></a>	
<a href="#"><i>objective_function</i></a>	Computes the relative value of the provided predictions compared to the actual labels, according a specified metric
<a href="#"><i>positive_only</i></a>	If True, this objective is only valid for positive data. Default False.

continues on next page

Table 268 – continued from previous page

<code>score</code>	Returns a numerical score indicating performance based on the differences between the predicted and actual values.
<code>validate_inputs</code>	Validates the input based on a few simple checks.

**classmethod** `calculate_percent_difference` (*cls*, *score*, *baseline\_score*)

Calculate the percent difference between scores.

#### Parameters

- **score** (*float*) – A score. Output of the score method of this objective.
- **baseline\_score** (*float*) – A score. Output of the score method of this objective. In practice, this is the score achieved on this objective with a baseline estimator.

#### Returns

**The percent difference between the scores. Note that for objectives that can be interpreted as percentages, this will be the difference between the reference score and score. For all other objectives, the difference will be normalized by the reference score.**

**Return type** float

**classmethod** `is_defined_for_problem_type` (*cls*, *problem\_type*)

**objective\_function** (*self*, *y\_true*, *y\_predicted*, *X=None*, *sample\_weight=None*)

Computes the relative value of the provided predictions compared to the actual labels, according a specified metric

**Arguments:** *y\_predicted* (*pd.Series*): Predicted values of length [*n\_samples*] *y\_true* (*pd.Series*): Actual class labels of length [*n\_samples*] *X* (*pd.DataFrame* or *np.ndarray*): Extra data of shape [*n\_samples*, *n\_features*] necessary to calculate score *sample\_weight* (*pd.DataFrame* or *np.ndarray*): Sample weights used in computing objective value result

**Returns** Numerical value used to calculate score

**positive\_only** (*self*)

If True, this objective is only valid for positive data. Default False.

**score** (*self*, *y\_true*, *y\_predicted*, *X=None*, *sample\_weight=None*)

Returns a numerical score indicating performance based on the differences between the predicted and actual values.

#### Parameters

- **y\_predicted** (*pd.Series*) – Predicted values of length [*n\_samples*]
- **y\_true** (*pd.Series*) – Actual class labels of length [*n\_samples*]
- **X** (*pd.DataFrame* or *np.ndarray*) – Extra data of shape [*n\_samples*, *n\_features*] necessary to calculate score
- **sample\_weight** (*pd.DataFrame* or *np.ndarray*) – Sample weights used in computing objective value result

**Returns** score

**validate\_inputs** (*self*, *y\_true*, *y\_predicted*)

Validates the input based on a few simple checks.

#### Parameters

- **y\_predicted** (*pd.Series*, or *pd.DataFrame*) – Predicted values of length [n\_samples]
- **y\_true** (*pd.Series*) – Actual class labels of length [n\_samples]

**Returns** None

**class** evalml.objectives.**MaxError**

Maximum residual error for regression.

**Attributes**

<b>greater_is_better</b>	False
<b>is_bounded_like_percentage</b>	False
<b>name</b>	MaxError
<b>perfect_score</b>	0.0
<b>problem_types</b>	[ProblemTypes.REGRESSION, ProblemTypes.TIME_SERIES_REGRESSION]
<b>score_needs_proba</b>	False

**Methods**

<i>calculate_percent_difference</i>	Calculate the percent difference between scores.
<i>is_defined_for_problem_type</i>	
<i>objective_function</i>	Computes the relative value of the provided predictions compared to the actual labels, according a specified metric
<i>positive_only</i>	If True, this objective is only valid for positive data. Default False.
<i>score</i>	Returns a numerical score indicating performance based on the differences between the predicted and actual values.
<i>validate_inputs</i>	Validates the input based on a few simple checks.

**classmethod** **calculate\_percent\_difference** (*cls*, *score*, *baseline\_score*)

Calculate the percent difference between scores.

**Parameters**

- **score** (*float*) – A score. Output of the score method of this objective.
- **baseline\_score** (*float*) – A score. Output of the score method of this objective. In practice, this is the score achieved on this objective with a baseline estimator.

**Returns**

**The percent difference between the scores. Note that for objectives that can be interpreted as percentages, this will be the difference between the reference score and score. For all other objectives, the difference will be normalized by the reference score.**

**Return type** float

**classmethod** **is\_defined\_for\_problem\_type** (*cls*, *problem\_type*)

**objective\_function** (*self*, *y\_true*, *y\_predicted*, *X=None*, *sample\_weight=None*)

Computes the relative value of the provided predictions compared to the actual labels, according a specified metric

**Arguments:** `y_predicted` (pd.Series): Predicted values of length `[n_samples]` `y_true` (pd.Series): Actual class labels of length `[n_samples]` `X` (pd.DataFrame or np.ndarray): Extra data of shape `[n_samples, n_features]` necessary to calculate score `sample_weight` (pd.DataFrame or np.ndarray): Sample weights used in computing objective value result

**Returns** Numerical value used to calculate score

**positive\_only** (cls)

If True, this objective is only valid for positive data. Default False.

**score** (self, y\_true, y\_predicted, X=None, sample\_weight=None)

Returns a numerical score indicating performance based on the differences between the predicted and actual values.

**Parameters**

- **y\_predicted** (pd.Series) – Predicted values of length `[n_samples]`
- **y\_true** (pd.Series) – Actual class labels of length `[n_samples]`
- **X** (pd.DataFrame or np.ndarray) – Extra data of shape `[n_samples, n_features]` necessary to calculate score
- **sample\_weight** (pd.DataFrame or np.ndarray) – Sample weights used in computing objective value result

**Returns** score

**validate\_inputs** (self, y\_true, y\_predicted)

Validates the input based on a few simple checks.

**Parameters**

- **y\_predicted** (pd.Series, or pd.DataFrame) – Predicted values of length `[n_samples]`
- **y\_true** (pd.Series) – Actual class labels of length `[n_samples]`

**Returns** None

**class** evalml.objectives.MCCBinary

Matthews correlation coefficient for binary classification.

**Attributes**

<b>greater_is_better</b>	True
<b>is_bounded_like_percentage</b>	False
<b>name</b>	MCC Binary
<b>perfect_score</b>	1.0
<b>problem_types</b>	[ProblemTypes.BINARY, ProblemTypes.TIME_SERIES_BINARY]
<b>score_needs_proba</b>	False

**Methods**

<code>calculate_percent_difference</code>	Calculate the percent difference between scores.
<code>can_optimize_threshold</code>	Returns a boolean determining if we can optimize the binary classification objective threshold.

continues on next page

Table 270 – continued from previous page

<code>decision_function</code>	Apply a learned threshold to predicted probabilities to get predicted classes.
<code>is_defined_for_problem_type</code>	
<code>objective_function</code>	Computes the relative value of the provided predictions compared to the actual labels, according a specified metric
<code>optimize_threshold</code>	Learn a binary classification threshold which optimizes the current objective.
<code>positive_only</code>	If True, this objective is only valid for positive data. Default False.
<code>score</code>	Returns a numerical score indicating performance based on the differences between the predicted and actual values.
<code>validate_inputs</code>	Validates the input based on a few simple checks.

**classmethod** `calculate_percent_difference` (*cls*, *score*, *baseline\_score*)

Calculate the percent difference between scores.

#### Parameters

- **score** (*float*) – A score. Output of the score method of this objective.
- **baseline\_score** (*float*) – A score. Output of the score method of this objective. In practice, this is the score achieved on this objective with a baseline estimator.

#### Returns

**The percent difference between the scores. Note that for objectives that can be interpreted** as percentages, this will be the difference between the reference score and score. For all other objectives, the difference will be normalized by the reference score.

**Return type** `float`

**property** `can_optimize_threshold` (*cls*)

Returns a boolean determining if we can optimize the binary classification objective threshold. This will be false for any objective that works directly with predicted probabilities, like log loss and AUC. Otherwise, it will be true.

**decision\_function** (*self*, *ypred\_proba*, *threshold=0.5*, *X=None*)

Apply a learned threshold to predicted probabilities to get predicted classes.

#### Parameters

- **ypred\_proba** (*pd.Series*, *np.ndarray*) – The classifier’s predicted probabilities
- **threshold** (*float*, *optional*) – Threshold used to make a prediction. Defaults to 0.5.
- **X** (*pd.DataFrame*, *optional*) – Any extra columns that are needed from training data.

**Returns** `predictions`

**classmethod** `is_defined_for_problem_type` (*cls*, *problem\_type*)

**objective\_function** (*self*, *y\_true*, *y\_predicted*, *X=None*, *sample\_weight=None*)

Computes the relative value of the provided predictions compared to the actual labels, according a specified metric

**Arguments:** `y_predicted` (`pd.Series`): Predicted values of length `[n_samples]` `y_true` (`pd.Series`): Actual class labels of length `[n_samples]` `X` (`pd.DataFrame` or `np.ndarray`): Extra data of shape `[n_samples, n_features]` necessary to calculate score `sample_weight` (`pd.DataFrame` or `np.ndarray`): Sample weights used in computing objective value result

**Returns** Numerical value used to calculate score

**`optimize_threshold`** (*self*, *ypred\_proba*, *y\_true*, *X=None*)

Learn a binary classification threshold which optimizes the current objective.

**Parameters**

- **`ypred_proba`** (*pd.Series*) – The classifier’s predicted probabilities
- **`y_true`** (*pd.Series*) – The ground truth for the predictions.
- **`X`** (*pd.DataFrame*, *optional*) – Any extra columns that are needed from training data.

**Returns** Optimal threshold for this objective

**`positive_only`** (*cls*)

If True, this objective is only valid for positive data. Default False.

**`score`** (*self*, *y\_true*, *y\_predicted*, *X=None*, *sample\_weight=None*)

Returns a numerical score indicating performance based on the differences between the predicted and actual values.

**Parameters**

- **`y_predicted`** (*pd.Series*) – Predicted values of length `[n_samples]`
- **`y_true`** (*pd.Series*) – Actual class labels of length `[n_samples]`
- **`X`** (*pd.DataFrame* or *np.ndarray*) – Extra data of shape `[n_samples, n_features]` necessary to calculate score
- **`sample_weight`** (*pd.DataFrame* or *np.ndarray*) – Sample weights used in computing objective value result

**Returns** score

**`validate_inputs`** (*self*, *y\_true*, *y\_predicted*)

Validates the input based on a few simple checks.

**Parameters**

- **`y_predicted`** (*pd.Series*, or *pd.DataFrame*) – Predicted values of length `[n_samples]`
- **`y_true`** (*pd.Series*) – Actual class labels of length `[n_samples]`

**Returns** None

**`class evalml.objectives.MCCMulticlass`**

Matthews correlation coefficient for multiclass classification.

**Attributes**



<b>greater_is_better</b>	True
<b>is_bounded_like_percentage</b>	False
<b>name</b>	MCC Multiclass
<b>perfect_score</b>	1.0
<b>problem_types</b>	[ProblemTypes.MULTICLASS, ProblemTypes.TIME_SERIES_MULTICLASS]
<b>score_needs_proba</b>	True

## Methods

<code>calculate_percent_difference</code>	Calculate the percent difference between scores.
<code>is_defined_for_problem_type</code>	
<code>objective_function</code>	Computes the relative value of the provided predictions compared to the actual labels, according a specified metric
<code>positive_only</code>	If True, this objective is only valid for positive data. Default False.
<code>score</code>	Returns a numerical score indicating performance based on the differences between the predicted and actual values.
<code>validate_inputs</code>	Validates the input based on a few simple checks.

**classmethod** `calculate_percent_difference` (*cls*, *score*, *baseline\_score*)

Calculate the percent difference between scores.

### Parameters

- **score** (*float*) – A score. Output of the score method of this objective.
- **baseline\_score** (*float*) – A score. Output of the score method of this objective. In practice, this is the score achieved on this objective with a baseline estimator.

### Returns

**The percent difference between the scores. Note that for objectives that can be interpreted as percentages, this will be the difference between the reference score and score. For all other objectives, the difference will be normalized by the reference score.**

**Return type** float

**classmethod** `is_defined_for_problem_type` (*cls*, *problem\_type*)

**objective\_function** (*self*, *y\_true*, *y\_predicted*, *X=None*, *sample\_weight=None*)

Computes the relative value of the provided predictions compared to the actual labels, according a specified metric

**Arguments:** *y\_predicted* (pd.Series): Predicted values of length [n\_samples] *y\_true* (pd.Series): Actual class labels of length [n\_samples] *X* (pd.DataFrame or np.ndarray): Extra data of shape [n\_samples, n\_features] necessary to calculate score *sample\_weight* (pd.DataFrame or np.ndarray): Sample weights used in computing objective value result

**Returns** Numerical value used to calculate score

**positive\_only** (*cls*)

If True, this objective is only valid for positive data. Default False.

**score** (*self, y\_true, y\_predicted, X=None, sample\_weight=None*)

Returns a numerical score indicating performance based on the differences between the predicted and actual values.

#### Parameters

- **y\_predicted** (*pd.Series*) – Predicted values of length [n\_samples]
- **y\_true** (*pd.Series*) – Actual class labels of length [n\_samples]
- **X** (*pd.DataFrame or np.ndarray*) – Extra data of shape [n\_samples, n\_features] necessary to calculate score
- **sample\_weight** (*pd.DataFrame or np.ndarray*) – Sample weights used in computing objective value result

**Returns** score

**validate\_inputs** (*self, y\_true, y\_predicted*)

Validates the input based on a few simple checks.

#### Parameters

- **y\_predicted** (*pd.Series, or pd.DataFrame*) – Predicted values of length [n\_samples]
- **y\_true** (*pd.Series*) – Actual class labels of length [n\_samples]

**Returns** None

**class** evalml.objectives.**MeanSquaredLogError**

Mean squared log error for regression.

Only valid for nonnegative inputs. Otherwise, will throw a ValueError

#### Attributes

<b>greater_is_better</b>	False
<b>is_bounded_like_percentage</b>	False
<b>name</b>	Mean Squared Log Error
<b>perfect_score</b>	0.0
<b>problem_types</b>	[ProblemTypes.REGRESSION, ProblemTypes.TIME_SERIES_REGRESSION]
<b>score_needs_problem</b>	True

#### Methods

<a href="#"><i>calculate_percent_difference</i></a>	Calculate the percent difference between scores.
<a href="#"><i>is_defined_for_problem_type</i></a>	
<a href="#"><i>objective_function</i></a>	Computes the relative value of the provided predictions compared to the actual labels, according a specified metric
<a href="#"><i>positive_only</i></a>	If True, this objective is only valid for positive data. Default False.

continues on next page

Table 272 – continued from previous page

<code>score</code>	Returns a numerical score indicating performance based on the differences between the predicted and actual values.
<code>validate_inputs</code>	Validates the input based on a few simple checks.

**classmethod** `calculate_percent_difference` (*cls*, *score*, *baseline\_score*)

Calculate the percent difference between scores.

#### Parameters

- **score** (*float*) – A score. Output of the score method of this objective.
- **baseline\_score** (*float*) – A score. Output of the score method of this objective. In practice, this is the score achieved on this objective with a baseline estimator.

#### Returns

**The percent difference between the scores. Note that for objectives that can be interpreted as percentages, this will be the difference between the reference score and score. For all other objectives, the difference will be normalized by the reference score.**

**Return type** `float`

**classmethod** `is_defined_for_problem_type` (*cls*, *problem\_type*)

**objective\_function** (*self*, *y\_true*, *y\_predicted*, *X=None*, *sample\_weight=None*)

Computes the relative value of the provided predictions compared to the actual labels, according a specified metric

**Arguments:** *y\_predicted* (*pd.Series*): Predicted values of length [*n\_samples*] *y\_true* (*pd.Series*): Actual class labels of length [*n\_samples*] *X* (*pd.DataFrame* or *np.ndarray*): Extra data of shape [*n\_samples*, *n\_features*] necessary to calculate score *sample\_weight* (*pd.DataFrame* or *np.ndarray*): Sample weights used in computing objective value result

**Returns** Numerical value used to calculate score

**positive\_only** (*self*)

If True, this objective is only valid for positive data. Default False.

**score** (*self*, *y\_true*, *y\_predicted*, *X=None*, *sample\_weight=None*)

Returns a numerical score indicating performance based on the differences between the predicted and actual values.

#### Parameters

- **y\_predicted** (*pd.Series*) – Predicted values of length [*n\_samples*]
- **y\_true** (*pd.Series*) – Actual class labels of length [*n\_samples*]
- **X** (*pd.DataFrame* or *np.ndarray*) – Extra data of shape [*n\_samples*, *n\_features*] necessary to calculate score
- **sample\_weight** (*pd.DataFrame* or *np.ndarray*) – Sample weights used in computing objective value result

**Returns** `score`

**validate\_inputs** (*self*, *y\_true*, *y\_predicted*)

Validates the input based on a few simple checks.

#### Parameters

- **y\_predicted** (*pd.Series*, or *pd.DataFrame*) – Predicted values of length [n\_samples]
- **y\_true** (*pd.Series*) – Actual class labels of length [n\_samples]

**Returns** None

**class** evalml.objectives.**MedianAE**

Median absolute error for regression.

**Attributes**

<b>greater_is_better</b>	False
<b>is_bounded_like_percentage</b>	False
<b>name</b>	MedianAE
<b>perfect_score</b>	0.0
<b>problem_types</b>	[ProblemTypes.REGRESSION, ProblemTypes.TIME_SERIES_REGRESSION]
<b>score_needs_proba</b>	False

**Methods**

<i>calculate_percent_difference</i>	Calculate the percent difference between scores.
<i>is_defined_for_problem_type</i>	
<i>objective_function</i>	Computes the relative value of the provided predictions compared to the actual labels, according a specified metric
<i>positive_only</i>	If True, this objective is only valid for positive data. Default False.
<i>score</i>	Returns a numerical score indicating performance based on the differences between the predicted and actual values.
<i>validate_inputs</i>	Validates the input based on a few simple checks.

**classmethod** **calculate\_percent\_difference** (*cls*, *score*, *baseline\_score*)

Calculate the percent difference between scores.

**Parameters**

- **score** (*float*) – A score. Output of the score method of this objective.
- **baseline\_score** (*float*) – A score. Output of the score method of this objective. In practice, this is the score achieved on this objective with a baseline estimator.

**Returns**

**The percent difference between the scores. Note that for objectives that can be interpreted as percentages, this will be the difference between the reference score and score. For all other objectives, the difference will be normalized by the reference score.**

**Return type** float

**classmethod** **is\_defined\_for\_problem\_type** (*cls*, *problem\_type*)

**objective\_function** (*self*, *y\_true*, *y\_predicted*, *X=None*, *sample\_weight=None*)

Computes the relative value of the provided predictions compared to the actual labels, according a specified metric

**Arguments:** `y_predicted` (`pd.Series`): Predicted values of length `[n_samples]` `y_true` (`pd.Series`): Actual class labels of length `[n_samples]` `X` (`pd.DataFrame` or `np.ndarray`): Extra data of shape `[n_samples, n_features]` necessary to calculate score `sample_weight` (`pd.DataFrame` or `np.ndarray`): Sample weights used in computing objective value result

**Returns** Numerical value used to calculate score

**positive\_only** (`cls`)

If True, this objective is only valid for positive data. Default False.

**score** (`self, y_true, y_predicted, X=None, sample_weight=None`)

Returns a numerical score indicating performance based on the differences between the predicted and actual values.

**Parameters**

- **y\_predicted** (`pd.Series`) – Predicted values of length `[n_samples]`
- **y\_true** (`pd.Series`) – Actual class labels of length `[n_samples]`
- **X** (`pd.DataFrame` or `np.ndarray`) – Extra data of shape `[n_samples, n_features]` necessary to calculate score
- **sample\_weight** (`pd.DataFrame` or `np.ndarray`) – Sample weights used in computing objective value result

**Returns** score

**validate\_inputs** (`self, y_true, y_predicted`)

Validates the input based on a few simple checks.

**Parameters**

- **y\_predicted** (`pd.Series`, or `pd.DataFrame`) – Predicted values of length `[n_samples]`
- **y\_true** (`pd.Series`) – Actual class labels of length `[n_samples]`

**Returns** None

**class** `evalml.objectives.MSE`

Mean squared error for regression.

**Attributes**

<b>greater_is_better</b>	False
<b>is_bounded_like_percentage</b>	False
<b>name</b>	MSE
<b>perfect_score</b>	0.0
<b>problem_types</b>	[ProblemTypes.REGRESSION, ProblemTypes.TIME_SERIES_REGRESSION]
<b>score_needs_problem</b>	True

**Methods**

<code>calculate_percent_difference</code>	Calculate the percent difference between scores.
<code>is_defined_for_problem_type</code>	

continues on next page

Table 274 – continued from previous page

<i>objective_function</i>	Computes the relative value of the provided predictions compared to the actual labels, according a specified metric
<i>positive_only</i>	If True, this objective is only valid for positive data. Default False.
<i>score</i>	Returns a numerical score indicating performance based on the differences between the predicted and actual values.
<i>validate_inputs</i>	Validates the input based on a few simple checks.

**classmethod** `calculate_percent_difference` (*cls*, *score*, *baseline\_score*)

Calculate the percent difference between scores.

#### Parameters

- **score** (*float*) – A score. Output of the score method of this objective.
- **baseline\_score** (*float*) – A score. Output of the score method of this objective. In practice, this is the score achieved on this objective with a baseline estimator.

#### Returns

**The percent difference between the scores. Note that for objectives that can be interpreted as percentages, this will be the difference between the reference score and score. For all other objectives, the difference will be normalized by the reference score.**

**Return type** `float`

**classmethod** `is_defined_for_problem_type` (*cls*, *problem\_type*)

**objective\_function** (*self*, *y\_true*, *y\_predicted*, *X=None*, *sample\_weight=None*)

Computes the relative value of the provided predictions compared to the actual labels, according a specified metric

**Arguments:** *y\_predicted* (*pd.Series*): Predicted values of length [*n\_samples*] *y\_true* (*pd.Series*): Actual class labels of length [*n\_samples*] *X* (*pd.DataFrame* or *np.ndarray*): Extra data of shape [*n\_samples*, *n\_features*] necessary to calculate score *sample\_weight* (*pd.DataFrame* or *np.ndarray*): Sample weights used in computing objective value result

**Returns** Numerical value used to calculate score

**positive\_only** (*cls*)

If True, this objective is only valid for positive data. Default False.

**score** (*self*, *y\_true*, *y\_predicted*, *X=None*, *sample\_weight=None*)

Returns a numerical score indicating performance based on the differences between the predicted and actual values.

#### Parameters

- **y\_predicted** (*pd.Series*) – Predicted values of length [*n\_samples*]
- **y\_true** (*pd.Series*) – Actual class labels of length [*n\_samples*]
- **X** (*pd.DataFrame* or *np.ndarray*) – Extra data of shape [*n\_samples*, *n\_features*] necessary to calculate score
- **sample\_weight** (*pd.DataFrame* or *np.ndarray*) – Sample weights used in computing objective value result

**Returns** `score`

**validate\_inputs** (*self*, *y\_true*, *y\_predicted*)

Validates the input based on a few simple checks.

#### Parameters

- **y\_predicted** (*pd.Series*, or *pd.DataFrame*) – Predicted values of length [n\_samples]
- **y\_true** (*pd.Series*) – Actual class labels of length [n\_samples]

**Returns** None

**class** evalml.objectives.**MulticlassClassificationObjective**

Base class for all multiclass classification objectives.

#### Attributes

<b>problem_types</b>	[ProblemTypes.MULTICLASS, ProblemTypes.TIME_SERIES_MULTICLASS]
----------------------	--

#### Methods

<i>calculate_percent_difference</i>	Calculate the percent difference between scores.
<i>greater_is_better</i>	Returns a boolean determining if a greater score indicates better model performance.
<i>is_bounded_like_percentage</i>	Returns whether this objective is bounded between 0 and 1, inclusive.
<i>is_defined_for_problem_type</i>	
<i>name</i>	Returns a name describing the objective.
<i>objective_function</i>	Computes the relative value of the provided predictions compared to the actual labels, according a specified metric
<i>perfect_score</i>	Returns the score obtained by evaluating this objective on a perfect model.
<i>positive_only</i>	If True, this objective is only valid for positive data. Default False.
<i>score</i>	Returns a numerical score indicating performance based on the differences between the predicted and actual values.
<i>score_needs_proba</i>	Returns a boolean determining if the score() method needs probability estimates. This should be true for objectives which work with predicted probabilities, like log loss or AUC, and false for objectives which compare predicted class labels to the actual labels, like F1 or correlation.
<i>validate_inputs</i>	Validates the input based on a few simple checks.

**classmethod** **calculate\_percent\_difference** (*cls*, *score*, *baseline\_score*)

Calculate the percent difference between scores.

#### Parameters

- **score** (*float*) – A score. Output of the score method of this objective.
- **baseline\_score** (*float*) – A score. Output of the score method of this objective. In practice, this is the score achieved on this objective with a baseline estimator.

**Returns**

**The percent difference between the scores. Note that for objectives that can be interpreted as percentages, this will be the difference between the reference score and score. For all other objectives, the difference will be normalized by the reference score.**

**Return type** float

**property greater\_is\_better** (*cls*)

Returns a boolean determining if a greater score indicates better model performance.

**property is\_bounded\_like\_percentage** (*cls*)

Returns whether this objective is bounded between 0 and 1, inclusive.

**classmethod is\_defined\_for\_problem\_type** (*cls, problem\_type*)

**property name** (*cls*)

Returns a name describing the objective.

**abstract classmethod objective\_function** (*cls, y\_true, y\_predicted, X=None, sample\_weight=None*)

Computes the relative value of the provided predictions compared to the actual labels, according a specified metric

**Arguments:** *y\_predicted* (pd.Series): Predicted values of length [n\_samples] *y\_true* (pd.Series): Actual class labels of length [n\_samples] *X* (pd.DataFrame or np.ndarray): Extra data of shape [n\_samples, n\_features] necessary to calculate score *sample\_weight* (pd.DataFrame or np.ndarray): Sample weights used in computing objective value result

**Returns** Numerical value used to calculate score

**property perfect\_score** (*cls*)

Returns the score obtained by evaluating this objective on a perfect model.

**positive\_only** (*cls*)

If True, this objective is only valid for positive data. Default False.

**score** (*self, y\_true, y\_predicted, X=None, sample\_weight=None*)

Returns a numerical score indicating performance based on the differences between the predicted and actual values.

**Parameters**

- **y\_predicted** (*pd.Series*) – Predicted values of length [n\_samples]
- **y\_true** (*pd.Series*) – Actual class labels of length [n\_samples]
- **X** (*pd.DataFrame or np.ndarray*) – Extra data of shape [n\_samples, n\_features] necessary to calculate score
- **sample\_weight** (*pd.DataFrame or np.ndarray*) – Sample weights used in computing objective value result

**Returns** score

**property score\_needs\_proba** (*cls*)

Returns a boolean determining if the score() method needs probability estimates. This should be true for objectives which work with predicted probabilities, like log loss or AUC, and false for objectives which compare predicted class labels to the actual labels, like F1 or correlation.

**validate\_inputs** (*self, y\_true, y\_predicted*)

Validates the input based on a few simple checks.

**Parameters**



- **y\_predicted** (*pd.Series*, or *pd.DataFrame*) – Predicted values of length [n\_samples]
- **y\_true** (*pd.Series*) – Actual class labels of length [n\_samples]

**Returns** None

**class** evalml.objectives.**ObjectiveBase**

Base class for all objectives.

**Attributes**

<b>problem_types</b>	None
----------------------	------

**Methods**

<i>calculate_percent_difference</i>	Calculate the percent difference between scores.
<i>greater_is_better</i>	Returns a boolean determining if a greater score indicates better model performance.
<i>is_bounded_like_percentage</i>	Returns whether this objective is bounded between 0 and 1, inclusive.
<i>is_defined_for_problem_type</i>	
<i>name</i>	Returns a name describing the objective.
<i>objective_function</i>	Computes the relative value of the provided predictions compared to the actual labels, according a specified metric
<i>perfect_score</i>	Returns the score obtained by evaluating this objective on a perfect model.
<i>positive_only</i>	If True, this objective is only valid for positive data. Default False.
<i>score</i>	Returns a numerical score indicating performance based on the differences between the predicted and actual values.
<i>score_needs_proba</i>	Returns a boolean determining if the score() method needs probability estimates. This should be true for objectives which work with predicted probabilities, like log loss or AUC, and false for objectives which compare predicted class labels to the actual labels, like F1 or correlation.
<i>validate_inputs</i>	Validates the input based on a few simple checks.

**classmethod** **calculate\_percent\_difference** (*cls*, *score*, *baseline\_score*)

Calculate the percent difference between scores.

**Parameters**

- **score** (*float*) – A score. Output of the score method of this objective.
- **baseline\_score** (*float*) – A score. Output of the score method of this objective. In practice, this is the score achieved on this objective with a baseline estimator.

**Returns**

**The percent difference between the scores. Note that for objectives that can be interpreted as percentages, this will be the difference between the reference score and score. For all**

other objectives, the difference will be normalized by the reference score.

**Return type** float

**property greater\_is\_better** (*cls*)

Returns a boolean determining if a greater score indicates better model performance.

**property is\_bounded\_like\_percentage** (*cls*)

Returns whether this objective is bounded between 0 and 1, inclusive.

**classmethod is\_defined\_for\_problem\_type** (*cls*, *problem\_type*)

**property name** (*cls*)

Returns a name describing the objective.

**abstract classmethod objective\_function** (*cls*, *y\_true*, *y\_predicted*, *X=None*, *sample\_weight=None*)

Computes the relative value of the provided predictions compared to the actual labels, according a specified metric

**Arguments:** *y\_predicted* (pd.Series): Predicted values of length [n\_samples] *y\_true* (pd.Series): Actual class labels of length [n\_samples] *X* (pd.DataFrame or np.ndarray): Extra data of shape [n\_samples, n\_features] necessary to calculate score *sample\_weight* (pd.DataFrame or np.ndarray): Sample weights used in computing objective value result

**Returns** Numerical value used to calculate score

**property perfect\_score** (*cls*)

Returns the score obtained by evaluating this objective on a perfect model.

**positive\_only** (*cls*)

If True, this objective is only valid for positive data. Default False.

**score** (*self*, *y\_true*, *y\_predicted*, *X=None*, *sample\_weight=None*)

Returns a numerical score indicating performance based on the differences between the predicted and actual values.

**Parameters**

- **y\_predicted** (*pd.Series*) – Predicted values of length [n\_samples]
- **y\_true** (*pd.Series*) – Actual class labels of length [n\_samples]
- **X** (*pd.DataFrame* or *np.ndarray*) – Extra data of shape [n\_samples, n\_features] necessary to calculate score
- **sample\_weight** (*pd.DataFrame* or *np.ndarray*) – Sample weights used in computing objective value result

**Returns** score

**property score\_needs\_proba** (*cls*)

Returns a boolean determining if the score() method needs probability estimates. This should be true for objectives which work with predicted probabilities, like log loss or AUC, and false for objectives which compare predicted class labels to the actual labels, like F1 or correlation.

**validate\_inputs** (*self*, *y\_true*, *y\_predicted*)

Validates the input based on a few simple checks.

**Parameters**

- **y\_predicted** (*pd.Series*, or *pd.DataFrame*) – Predicted values of length [n\_samples]

- **y\_true** (*pd.Series*) – Actual class labels of length [n\_samples]

**Returns** None

**class** evalml.objectives.Precision

Precision score for binary classification.

**Attributes**

<b>greater_is_better</b>	True
<b>is_bounded_like_percentage</b>	True
<b>name</b>	Precision
<b>perfect_score</b>	1.0
<b>problem_types</b>	[ProblemTypes.BINARY, ProblemTypes.TIME_SERIES_BINARY]
<b>score_needs_proba</b>	True

**Methods**

<i>calculate_percent_difference</i>	Calculate the percent difference between scores.
<i>can_optimize_threshold</i>	Returns a boolean determining if we can optimize the binary classification objective threshold.
<i>decision_function</i>	Apply a learned threshold to predicted probabilities to get predicted classes.
<i>is_defined_for_problem_type</i>	
<i>objective_function</i>	Computes the relative value of the provided predictions compared to the actual labels, according a specified metric
<i>optimize_threshold</i>	Learn a binary classification threshold which optimizes the current objective.
<i>positive_only</i>	If True, this objective is only valid for positive data. Default False.
<i>score</i>	Returns a numerical score indicating performance based on the differences between the predicted and actual values.
<i>validate_inputs</i>	Validates the input based on a few simple checks.

**classmethod** **calculate\_percent\_difference** (*cls*, *score*, *baseline\_score*)

Calculate the percent difference between scores.

**Parameters**

- **score** (*float*) – A score. Output of the score method of this objective.
- **baseline\_score** (*float*) – A score. Output of the score method of this objective. In practice, this is the score achieved on this objective with a baseline estimator.

**Returns**

**The percent difference between the scores. Note that for objectives that can be interpreted as percentages, this will be the difference between the reference score and score. For all other objectives, the difference will be normalized by the reference score.**

**Return type** float

**property can\_optimize\_threshold** (*cls*)

Returns a boolean determining if we can optimize the binary classification objective threshold. This will be false for any objective that works directly with predicted probabilities, like log loss and AUC. Otherwise, it will be true.

**decision\_function** (*self*, *ypred\_proba*, *threshold=0.5*, *X=None*)

Apply a learned threshold to predicted probabilities to get predicted classes.

**Parameters**

- **ypred\_proba** (*pd.Series*, *np.ndarray*) – The classifier’s predicted probabilities
- **threshold** (*float*, *optional*) – Threshold used to make a prediction. Defaults to 0.5.
- **X** (*pd.DataFrame*, *optional*) – Any extra columns that are needed from training data.

**Returns** predictions

**classmethod is\_defined\_for\_problem\_type** (*cls*, *problem\_type*)**objective\_function** (*self*, *y\_true*, *y\_predicted*, *X=None*, *sample\_weight=None*)

Computes the relative value of the provided predictions compared to the actual labels, according a specified metric

**Arguments:** *y\_predicted* (*pd.Series*): Predicted values of length [n\_samples] *y\_true* (*pd.Series*): Actual class labels of length [n\_samples] *X* (*pd.DataFrame* or *np.ndarray*): Extra data of shape [n\_samples, n\_features] necessary to calculate score *sample\_weight* (*pd.DataFrame* or *np.ndarray*): Sample weights used in computing objective value result

**Returns** Numerical value used to calculate score

**optimize\_threshold** (*self*, *ypred\_proba*, *y\_true*, *X=None*)

Learn a binary classification threshold which optimizes the current objective.

**Parameters**

- **ypred\_proba** (*pd.Series*) – The classifier’s predicted probabilities
- **y\_true** (*pd.Series*) – The ground truth for the predictions.
- **X** (*pd.DataFrame*, *optional*) – Any extra columns that are needed from training data.

**Returns** Optimal threshold for this objective

**positive\_only** (*cls*)

If True, this objective is only valid for positive data. Default False.

**score** (*self*, *y\_true*, *y\_predicted*, *X=None*, *sample\_weight=None*)

Returns a numerical score indicating performance based on the differences between the predicted and actual values.

**Parameters**

- **y\_predicted** (*pd.Series*) – Predicted values of length [n\_samples]
- **y\_true** (*pd.Series*) – Actual class labels of length [n\_samples]
- **X** (*pd.DataFrame* or *np.ndarray*) – Extra data of shape [n\_samples, n\_features] necessary to calculate score

- **sample\_weight** (*pd.DataFrame* or *np.ndarray*) – Sample weights used in computing objective value result

**Returns** score

**validate\_inputs** (*self*, *y\_true*, *y\_predicted*)

Validates the input based on a few simple checks.

**Parameters**

- **y\_predicted** (*pd.Series*, or *pd.DataFrame*) – Predicted values of length [n\_samples]
- **y\_true** (*pd.Series*) – Actual class labels of length [n\_samples]

**Returns** None

**class** evalml.objectives.**PrecisionMacro**

Precision score for multiclass classification using macro averaging.

**Attributes**

<b>greater_is_better</b>	True
<b>is_bounded_like_percentage</b>	True
<b>name</b>	Precision Macro
<b>perfect_score</b>	1.0
<b>problem_types</b>	[ProblemTypes.MULTICLASS, ProblemTypes.TIME_SERIES_MULTICLASS]
<b>score_needs_proba</b>	True

**Methods**

<i>calculate_percent_difference</i>	Calculate the percent difference between scores.
<i>is_defined_for_problem_type</i>	
<i>objective_function</i>	Computes the relative value of the provided predictions compared to the actual labels, according a specified metric
<i>positive_only</i>	If True, this objective is only valid for positive data. Default False.
<i>score</i>	Returns a numerical score indicating performance based on the differences between the predicted and actual values.
<i>validate_inputs</i>	Validates the input based on a few simple checks.

**classmethod** **calculate\_percent\_difference** (*cls*, *score*, *baseline\_score*)

Calculate the percent difference between scores.

**Parameters**

- **score** (*float*) – A score. Output of the score method of this objective.
- **baseline\_score** (*float*) – A score. Output of the score method of this objective. In practice, this is the score achieved on this objective with a baseline estimator.

**Returns**

The percent difference between the scores. Note that for objectives that can be interpreted

as percentages, this will be the difference between the reference score and score. For all other objectives, the difference will be normalized by the reference score.

**Return type** float

**classmethod** `is_defined_for_problem_type(cls, problem_type)`

**objective\_function** (*self*, *y\_true*, *y\_predicted*, *X=None*, *sample\_weight=None*)

Computes the relative value of the provided predictions compared to the actual labels, according a specified metric

**Arguments:** *y\_predicted* (pd.Series): Predicted values of length [n\_samples] *y\_true* (pd.Series): Actual class labels of length [n\_samples] *X* (pd.DataFrame or np.ndarray): Extra data of shape [n\_samples, n\_features] necessary to calculate score *sample\_weight* (pd.DataFrame or np.ndarray): Sample weights used in computing objective value result

**Returns** Numerical value used to calculate score

**positive\_only** (*cls*)

If True, this objective is only valid for positive data. Default False.

**score** (*self*, *y\_true*, *y\_predicted*, *X=None*, *sample\_weight=None*)

Returns a numerical score indicating performance based on the differences between the predicted and actual values.

**Parameters**

- **y\_predicted** (pd.Series) – Predicted values of length [n\_samples]
- **y\_true** (pd.Series) – Actual class labels of length [n\_samples]
- **X** (pd.DataFrame or np.ndarray) – Extra data of shape [n\_samples, n\_features] necessary to calculate score
- **sample\_weight** (pd.DataFrame or np.ndarray) – Sample weights used in computing objective value result

**Returns** score

**validate\_inputs** (*self*, *y\_true*, *y\_predicted*)

Validates the input based on a few simple checks.

**Parameters**

- **y\_predicted** (pd.Series, or pd.DataFrame) – Predicted values of length [n\_samples]
- **y\_true** (pd.Series) – Actual class labels of length [n\_samples]

**Returns** None

**class** evalml.objectives.PrecisionMicro

Precision score for multiclass classification using micro averaging.

**Attributes**

<b>greater_is_better</b>	True
<b>is_bounded_like_percentage</b>	True
<b>name</b>	Precision Micro
<b>perfect_score</b>	1.0
<b>problem_types</b>	[ProblemTypes.MULTICLASS, ProblemTypes.TIME_SERIES_MULTICLASS]
<b>score_needs_proba</b>	True

## Methods

<i>calculate_percent_difference</i>	Calculate the percent difference between scores.
<i>is_defined_for_problem_type</i>	
<i>objective_function</i>	Computes the relative value of the provided predictions compared to the actual labels, according a specified metric
<i>positive_only</i>	If True, this objective is only valid for positive data. Default False.
<i>score</i>	Returns a numerical score indicating performance based on the differences between the predicted and actual values.
<i>validate_inputs</i>	Validates the input based on a few simple checks.

**classmethod** `calculate_percent_difference` (*cls*, *score*, *baseline\_score*)

Calculate the percent difference between scores.

### Parameters

- **score** (*float*) – A score. Output of the score method of this objective.
- **baseline\_score** (*float*) – A score. Output of the score method of this objective. In practice, this is the score achieved on this objective with a baseline estimator.

### Returns

**The percent difference between the scores. Note that for objectives that can be interpreted as percentages, this will be the difference between the reference score and score. For all other objectives, the difference will be normalized by the reference score.**

**Return type** float

**classmethod** `is_defined_for_problem_type` (*cls*, *problem\_type*)

**objective\_function** (*self*, *y\_true*, *y\_predicted*, *X=None*, *sample\_weight=None*)

Computes the relative value of the provided predictions compared to the actual labels, according a specified metric

**Arguments:** *y\_predicted* (pd.Series): Predicted values of length [n\_samples] *y\_true* (pd.Series): Actual class labels of length [n\_samples] *X* (pd.DataFrame or np.ndarray): Extra data of shape [n\_samples, n\_features] necessary to calculate score *sample\_weight* (pd.DataFrame or np.ndarray): Sample weights used in computing objective value result

**Returns** Numerical value used to calculate score

**positive\_only** (*cls*)

If True, this objective is only valid for positive data. Default False.

**score** (*self*, *y\_true*, *y\_predicted*, *X=None*, *sample\_weight=None*)

Returns a numerical score indicating performance based on the differences between the predicted and actual values.

#### Parameters

- **y\_predicted** (*pd.Series*) – Predicted values of length [n\_samples]
- **y\_true** (*pd.Series*) – Actual class labels of length [n\_samples]
- **X** (*pd.DataFrame* or *np.ndarray*) – Extra data of shape [n\_samples, n\_features] necessary to calculate score
- **sample\_weight** (*pd.DataFrame* or *np.ndarray*) – Sample weights used in computing objective value result

**Returns** score

**validate\_inputs** (*self*, *y\_true*, *y\_predicted*)

Validates the input based on a few simple checks.

#### Parameters

- **y\_predicted** (*pd.Series*, or *pd.DataFrame*) – Predicted values of length [n\_samples]
- **y\_true** (*pd.Series*) – Actual class labels of length [n\_samples]

**Returns** None

**class** evalml.objectives.PrecisionWeighted

Precision score for multiclass classification using weighted averaging.

#### Attributes

<b>greater_is_better</b>	True
<b>is_bounded_like_percentage</b>	True
<b>name</b>	Precision Weighted
<b>perfect_score</b>	1.0
<b>problem_types</b>	[ProblemTypes.MULTICLASS, ProblemTypes.TIME_SERIES_MULTICLASS]
<b>score_needs_proba</b>	False

#### Methods

<i>calculate_percent_difference</i>	Calculate the percent difference between scores.
<i>is_defined_for_problem_type</i>	
<i>objective_function</i>	Computes the relative value of the provided predictions compared to the actual labels, according a specified metric
<i>positive_only</i>	If True, this objective is only valid for positive data. Default False.

continues on next page



Table 280 – continued from previous page

<code>score</code>	Returns a numerical score indicating performance based on the differences between the predicted and actual values.
<code>validate_inputs</code>	Validates the input based on a few simple checks.

**classmethod** `calculate_percent_difference` (*cls*, *score*, *baseline\_score*)

Calculate the percent difference between scores.

#### Parameters

- **score** (*float*) – A score. Output of the score method of this objective.
- **baseline\_score** (*float*) – A score. Output of the score method of this objective. In practice, this is the score achieved on this objective with a baseline estimator.

#### Returns

**The percent difference between the scores. Note that for objectives that can be interpreted as percentages, this will be the difference between the reference score and score. For all other objectives, the difference will be normalized by the reference score.**

**Return type** `float`

**classmethod** `is_defined_for_problem_type` (*cls*, *problem\_type*)

**objective\_function** (*self*, *y\_true*, *y\_predicted*, *X=None*, *sample\_weight=None*)

Computes the relative value of the provided predictions compared to the actual labels, according a specified metric

**Arguments:** *y\_predicted* (*pd.Series*): Predicted values of length [*n\_samples*] *y\_true* (*pd.Series*): Actual class labels of length [*n\_samples*] *X* (*pd.DataFrame* or *np.ndarray*): Extra data of shape [*n\_samples*, *n\_features*] necessary to calculate score *sample\_weight* (*pd.DataFrame* or *np.ndarray*): Sample weights used in computing objective value result

**Returns** Numerical value used to calculate score

**positive\_only** (*cls*)

If True, this objective is only valid for positive data. Default False.

**score** (*self*, *y\_true*, *y\_predicted*, *X=None*, *sample\_weight=None*)

Returns a numerical score indicating performance based on the differences between the predicted and actual values.

#### Parameters

- **y\_predicted** (*pd.Series*) – Predicted values of length [*n\_samples*]
- **y\_true** (*pd.Series*) – Actual class labels of length [*n\_samples*]
- **X** (*pd.DataFrame* or *np.ndarray*) – Extra data of shape [*n\_samples*, *n\_features*] necessary to calculate score
- **sample\_weight** (*pd.DataFrame* or *np.ndarray*) – Sample weights used in computing objective value result

**Returns** `score`

**validate\_inputs** (*self*, *y\_true*, *y\_predicted*)

Validates the input based on a few simple checks.

#### Parameters

- **y\_predicted** (*pd.Series*, or *pd.DataFrame*) – Predicted values of length [n\_samples]
- **y\_true** (*pd.Series*) – Actual class labels of length [n\_samples]

**Returns** None

**class** evalml.objectives.**R2**

Coefficient of determination for regression.

**Attributes**

<b>greater_is_better</b>	True
<b>is_bounded_like_percentage</b>	False
<b>name</b>	R2
<b>perfect_score</b>	1
<b>problem_types</b>	[ProblemTypes.REGRESSION, ProblemTypes.TIME_SERIES_REGRESSION]
<b>score_needs_proba</b>	False

**Methods**

<i>calculate_percent_difference</i>	Calculate the percent difference between scores.
<i>is_defined_for_problem_type</i>	
<i>objective_function</i>	Computes the relative value of the provided predictions compared to the actual labels, according a specified metric
<i>positive_only</i>	If True, this objective is only valid for positive data. Default False.
<i>score</i>	Returns a numerical score indicating performance based on the differences between the predicted and actual values.
<i>validate_inputs</i>	Validates the input based on a few simple checks.

**classmethod** **calculate\_percent\_difference** (*cls*, *score*, *baseline\_score*)

Calculate the percent difference between scores.

**Parameters**

- **score** (*float*) – A score. Output of the score method of this objective.
- **baseline\_score** (*float*) – A score. Output of the score method of this objective. In practice, this is the score achieved on this objective with a baseline estimator.

**Returns**

**The percent difference between the scores. Note that for objectives that can be interpreted as percentages, this will be the difference between the reference score and score. For all other objectives, the difference will be normalized by the reference score.**

**Return type** float

**classmethod** **is\_defined\_for\_problem\_type** (*cls*, *problem\_type*)

**objective\_function** (*self*, *y\_true*, *y\_predicted*, *X=None*, *sample\_weight=None*)

Computes the relative value of the provided predictions compared to the actual labels, according a specified metric

**Arguments:** `y_predicted` (`pd.Series`): Predicted values of length `[n_samples]` `y_true` (`pd.Series`): Actual class labels of length `[n_samples]` `X` (`pd.DataFrame` or `np.ndarray`): Extra data of shape `[n_samples, n_features]` necessary to calculate score `sample_weight` (`pd.DataFrame` or `np.ndarray`): Sample weights used in computing objective value result

**Returns** Numerical value used to calculate score

**positive\_only** (`cls`)

If True, this objective is only valid for positive data. Default False.

**score** (`self, y_true, y_predicted, X=None, sample_weight=None`)

Returns a numerical score indicating performance based on the differences between the predicted and actual values.

**Parameters**

- **y\_predicted** (`pd.Series`) – Predicted values of length `[n_samples]`
- **y\_true** (`pd.Series`) – Actual class labels of length `[n_samples]`
- **X** (`pd.DataFrame` or `np.ndarray`) – Extra data of shape `[n_samples, n_features]` necessary to calculate score
- **sample\_weight** (`pd.DataFrame` or `np.ndarray`) – Sample weights used in computing objective value result

**Returns** score

**validate\_inputs** (`self, y_true, y_predicted`)

Validates the input based on a few simple checks.

**Parameters**

- **y\_predicted** (`pd.Series`, or `pd.DataFrame`) – Predicted values of length `[n_samples]`
- **y\_true** (`pd.Series`) – Actual class labels of length `[n_samples]`

**Returns** None

**class** `evalml.objectives.Recall`

Recall score for binary classification.

**Attributes**

<b>greater_is_better</b>	True
<b>is_bounded_like_percentage</b>	True
<b>name</b>	Recall
<b>perfect_score</b>	1.0
<b>problem_types</b>	[ProblemTypes.BINARY, ProblemTypes.TIME_SERIES_BINARY]
<b>score_needs_proba</b>	False

**Methods**

<code>calculate_percent_difference</code>	Calculate the percent difference between scores.
<code>can_optimize_threshold</code>	Returns a boolean determining if we can optimize the binary classification objective threshold.

continues on next page

Table 282 – continued from previous page

<code>decision_function</code>	Apply a learned threshold to predicted probabilities to get predicted classes.
<code>is_defined_for_problem_type</code>	
<code>objective_function</code>	Computes the relative value of the provided predictions compared to the actual labels, according a specified metric
<code>optimize_threshold</code>	Learn a binary classification threshold which optimizes the current objective.
<code>positive_only</code>	If True, this objective is only valid for positive data. Default False.
<code>score</code>	Returns a numerical score indicating performance based on the differences between the predicted and actual values.
<code>validate_inputs</code>	Validates the input based on a few simple checks.

**classmethod** `calculate_percent_difference` (*cls*, *score*, *baseline\_score*)

Calculate the percent difference between scores.

#### Parameters

- **score** (*float*) – A score. Output of the score method of this objective.
- **baseline\_score** (*float*) – A score. Output of the score method of this objective. In practice, this is the score achieved on this objective with a baseline estimator.

#### Returns

**The percent difference between the scores. Note that for objectives that can be interpreted** as percentages, this will be the difference between the reference score and score. For all other objectives, the difference will be normalized by the reference score.

**Return type** `float`

**property** `can_optimize_threshold` (*cls*)

Returns a boolean determining if we can optimize the binary classification objective threshold. This will be false for any objective that works directly with predicted probabilities, like log loss and AUC. Otherwise, it will be true.

**decision\_function** (*self*, *ypred\_proba*, *threshold=0.5*, *X=None*)

Apply a learned threshold to predicted probabilities to get predicted classes.

#### Parameters

- **ypred\_proba** (*pd.Series*, *np.ndarray*) – The classifier’s predicted probabilities
- **threshold** (*float*, *optional*) – Threshold used to make a prediction. Defaults to 0.5.
- **X** (*pd.DataFrame*, *optional*) – Any extra columns that are needed from training data.

**Returns** `predictions`

**classmethod** `is_defined_for_problem_type` (*cls*, *problem\_type*)

**objective\_function** (*self*, *y\_true*, *y\_predicted*, *X=None*, *sample\_weight=None*)

Computes the relative value of the provided predictions compared to the actual labels, according a specified metric

**Arguments:** `y_predicted` (`pd.Series`): Predicted values of length `[n_samples]` `y_true` (`pd.Series`): Actual class labels of length `[n_samples]` `X` (`pd.DataFrame` or `np.ndarray`): Extra data of shape `[n_samples, n_features]` necessary to calculate score `sample_weight` (`pd.DataFrame` or `np.ndarray`): Sample weights used in computing objective value result

**Returns** Numerical value used to calculate score

**optimize\_threshold** (*self*, *ypred\_proba*, *y\_true*, *X=None*)

Learn a binary classification threshold which optimizes the current objective.

**Parameters**

- **ypred\_proba** (`pd.Series`) – The classifier’s predicted probabilities
- **y\_true** (`pd.Series`) – The ground truth for the predictions.
- **X** (`pd.DataFrame`, *optional*) – Any extra columns that are needed from training data.

**Returns** Optimal threshold for this objective

**positive\_only** (*cls*)

If True, this objective is only valid for positive data. Default False.

**score** (*self*, *y\_true*, *y\_predicted*, *X=None*, *sample\_weight=None*)

Returns a numerical score indicating performance based on the differences between the predicted and actual values.

**Parameters**

- **y\_predicted** (`pd.Series`) – Predicted values of length `[n_samples]`
- **y\_true** (`pd.Series`) – Actual class labels of length `[n_samples]`
- **X** (`pd.DataFrame` or `np.ndarray`) – Extra data of shape `[n_samples, n_features]` necessary to calculate score
- **sample\_weight** (`pd.DataFrame` or `np.ndarray`) – Sample weights used in computing objective value result

**Returns** score

**validate\_inputs** (*self*, *y\_true*, *y\_predicted*)

Validates the input based on a few simple checks.

**Parameters**

- **y\_predicted** (`pd.Series`, or `pd.DataFrame`) – Predicted values of length `[n_samples]`
- **y\_true** (`pd.Series`) – Actual class labels of length `[n_samples]`

**Returns** None

**class** `evalml.objectives.RecallMacro`

Recall score for multiclass classification using macro averaging.

**Attributes**

<b>greater_is_better</b>	True
<b>is_bounded_like_percentage</b>	True
<b>name</b>	Recall Macro
<b>perfect_score</b>	1.0
<b>problem_types</b>	[ProblemTypes.MULTICLASS, ProblemTypes.TIME_SERIES_MULTICLASS]
<b>score_needs_proba</b>	False

## Methods

<i>calculate_percent_difference</i>	Calculate the percent difference between scores.
<i>is_defined_for_problem_type</i>	
<i>objective_function</i>	Computes the relative value of the provided predictions compared to the actual labels, according a specified metric
<i>positive_only</i>	If True, this objective is only valid for positive data. Default False.
<i>score</i>	Returns a numerical score indicating performance based on the differences between the predicted and actual values.
<i>validate_inputs</i>	Validates the input based on a few simple checks.

**classmethod** `calculate_percent_difference` (*cls*, *score*, *baseline\_score*)

Calculate the percent difference between scores.

### Parameters

- **score** (*float*) – A score. Output of the score method of this objective.
- **baseline\_score** (*float*) – A score. Output of the score method of this objective. In practice, this is the score achieved on this objective with a baseline estimator.

### Returns

**The percent difference between the scores. Note that for objectives that can be interpreted as percentages, this will be the difference between the reference score and score. For all other objectives, the difference will be normalized by the reference score.**

**Return type** float

**classmethod** `is_defined_for_problem_type` (*cls*, *problem\_type*)

**objective\_function** (*self*, *y\_true*, *y\_predicted*, *X=None*, *sample\_weight=None*)

Computes the relative value of the provided predictions compared to the actual labels, according a specified metric

**Arguments:** *y\_predicted* (pd.Series): Predicted values of length [n\_samples] *y\_true* (pd.Series): Actual class labels of length [n\_samples] *X* (pd.DataFrame or np.ndarray): Extra data of shape [n\_samples, n\_features] necessary to calculate score *sample\_weight* (pd.DataFrame or np.ndarray): Sample weights used in computing objective value result

**Returns** Numerical value used to calculate score

**positive\_only** (*cls*)

If True, this objective is only valid for positive data. Default False.

**score** (*self, y\_true, y\_predicted, X=None, sample\_weight=None*)

Returns a numerical score indicating performance based on the differences between the predicted and actual values.

#### Parameters

- **y\_predicted** (*pd.Series*) – Predicted values of length [n\_samples]
- **y\_true** (*pd.Series*) – Actual class labels of length [n\_samples]
- **X** (*pd.DataFrame or np.ndarray*) – Extra data of shape [n\_samples, n\_features] necessary to calculate score
- **sample\_weight** (*pd.DataFrame or np.ndarray*) – Sample weights used in computing objective value result

**Returns** score

**validate\_inputs** (*self, y\_true, y\_predicted*)

Validates the input based on a few simple checks.

#### Parameters

- **y\_predicted** (*pd.Series, or pd.DataFrame*) – Predicted values of length [n\_samples]
- **y\_true** (*pd.Series*) – Actual class labels of length [n\_samples]

**Returns** None

**class** evalml.objectives.**RecallMicro**

Recall score for multiclass classification using micro averaging.

#### Attributes

<b>greater_is_better</b>	True
<b>is_bounded_like_percentage</b>	True
<b>name</b>	Recall Micro
<b>perfect_score</b>	1.0
<b>problem_types</b>	[ProblemTypes.MULTICLASS, ProblemTypes.TIME_SERIES_MULTICLASS]
<b>score_needs_proba</b>	False

#### Methods

<i>calculate_percent_difference</i>	Calculate the percent difference between scores.
<i>is_defined_for_problem_type</i>	
<i>objective_function</i>	Computes the relative value of the provided predictions compared to the actual labels, according a specified metric
<i>positive_only</i>	If True, this objective is only valid for positive data. Default False.

continues on next page

Table 284 – continued from previous page

<code>score</code>	Returns a numerical score indicating performance based on the differences between the predicted and actual values.
<code>validate_inputs</code>	Validates the input based on a few simple checks.

**classmethod** `calculate_percent_difference` (*cls*, *score*, *baseline\_score*)

Calculate the percent difference between scores.

#### Parameters

- **score** (*float*) – A score. Output of the score method of this objective.
- **baseline\_score** (*float*) – A score. Output of the score method of this objective. In practice, this is the score achieved on this objective with a baseline estimator.

#### Returns

**The percent difference between the scores. Note that for objectives that can be interpreted as percentages, this will be the difference between the reference score and score. For all other objectives, the difference will be normalized by the reference score.**

**Return type** float

**classmethod** `is_defined_for_problem_type` (*cls*, *problem\_type*)

**objective\_function** (*self*, *y\_true*, *y\_predicted*, *X=None*, *sample\_weight=None*)

Computes the relative value of the provided predictions compared to the actual labels, according a specified metric

**Arguments:** *y\_predicted* (*pd.Series*): Predicted values of length [*n\_samples*] *y\_true* (*pd.Series*): Actual class labels of length [*n\_samples*] *X* (*pd.DataFrame* or *np.ndarray*): Extra data of shape [*n\_samples*, *n\_features*] necessary to calculate score *sample\_weight* (*pd.DataFrame* or *np.ndarray*): Sample weights used in computing objective value result

**Returns** Numerical value used to calculate score

**positive\_only** (*cls*)

If True, this objective is only valid for positive data. Default False.

**score** (*self*, *y\_true*, *y\_predicted*, *X=None*, *sample\_weight=None*)

Returns a numerical score indicating performance based on the differences between the predicted and actual values.

#### Parameters

- **y\_predicted** (*pd.Series*) – Predicted values of length [*n\_samples*]
- **y\_true** (*pd.Series*) – Actual class labels of length [*n\_samples*]
- **X** (*pd.DataFrame* or *np.ndarray*) – Extra data of shape [*n\_samples*, *n\_features*] necessary to calculate score
- **sample\_weight** (*pd.DataFrame* or *np.ndarray*) – Sample weights used in computing objective value result

**Returns** score

**validate\_inputs** (*self*, *y\_true*, *y\_predicted*)

Validates the input based on a few simple checks.

#### Parameters



- **y\_predicted** (*pd.Series*, or *pd.DataFrame*) – Predicted values of length [n\_samples]
- **y\_true** (*pd.Series*) – Actual class labels of length [n\_samples]

**Returns** None

**class** evalml.objectives.RecallWeighted

Recall score for multiclass classification using weighted averaging.

**Attributes**

<b>greater_is_better</b>	True
<b>is_bounded_like_percentage</b>	True
<b>name</b>	Recall Weighted
<b>perfect_score</b>	1.0
<b>problem_types</b>	[ProblemTypes.MULTICLASS, ProblemTypes.TIME_SERIES_MULTICLASS]
<b>score_needs_proba</b>	True

**Methods**

<i>calculate_percent_difference</i>	Calculate the percent difference between scores.
<i>is_defined_for_problem_type</i>	
<i>objective_function</i>	Computes the relative value of the provided predictions compared to the actual labels, according a specified metric
<i>positive_only</i>	If True, this objective is only valid for positive data. Default False.
<i>score</i>	Returns a numerical score indicating performance based on the differences between the predicted and actual values.
<i>validate_inputs</i>	Validates the input based on a few simple checks.

**classmethod** **calculate\_percent\_difference** (*cls*, *score*, *baseline\_score*)

Calculate the percent difference between scores.

**Parameters**

- **score** (*float*) – A score. Output of the score method of this objective.
- **baseline\_score** (*float*) – A score. Output of the score method of this objective. In practice, this is the score achieved on this objective with a baseline estimator.

**Returns**

**The percent difference between the scores. Note that for objectives that can be interpreted as percentages, this will be the difference between the reference score and score. For all other objectives, the difference will be normalized by the reference score.**

**Return type** float

**classmethod** **is\_defined\_for\_problem\_type** (*cls*, *problem\_type*)

**objective\_function** (*self*, *y\_true*, *y\_predicted*, *X=None*, *sample\_weight=None*)

Computes the relative value of the provided predictions compared to the actual labels, according a specified metric

**Arguments:** `y_predicted` (`pd.Series`): Predicted values of length `[n_samples]` `y_true` (`pd.Series`): Actual class labels of length `[n_samples]` `X` (`pd.DataFrame` or `np.ndarray`): Extra data of shape `[n_samples, n_features]` necessary to calculate score `sample_weight` (`pd.DataFrame` or `np.ndarray`): Sample weights used in computing objective value result

**Returns** Numerical value used to calculate score

**positive\_only** (*cls*)

If True, this objective is only valid for positive data. Default False.

**score** (*self*, *y\_true*, *y\_predicted*, *X=None*, *sample\_weight=None*)

Returns a numerical score indicating performance based on the differences between the predicted and actual values.

**Parameters**

- **y\_predicted** (*pd.Series*) – Predicted values of length `[n_samples]`
- **y\_true** (*pd.Series*) – Actual class labels of length `[n_samples]`
- **X** (*pd.DataFrame* or *np.ndarray*) – Extra data of shape `[n_samples, n_features]` necessary to calculate score
- **sample\_weight** (*pd.DataFrame* or *np.ndarray*) – Sample weights used in computing objective value result

**Returns** score

**validate\_inputs** (*self*, *y\_true*, *y\_predicted*)

Validates the input based on a few simple checks.

**Parameters**

- **y\_predicted** (*pd.Series*, or *pd.DataFrame*) – Predicted values of length `[n_samples]`
- **y\_true** (*pd.Series*) – Actual class labels of length `[n_samples]`

**Returns** None

**class** `evalml.objectives.RegressionObjective`

Base class for all regression objectives.

**Attributes**

<b>problem_types</b>	[ProblemTypes.REGRESSION, ProblemTypes.TIME_SERIES_REGRESSION]
----------------------	--

**Methods**

<code>calculate_percent_difference</code>	Calculate the percent difference between scores.
<code>greater_is_better</code>	Returns a boolean determining if a greater score indicates better model performance.
<code>is_bounded_like_percentage</code>	Returns whether this objective is bounded between 0 and 1, inclusive.
<code>is_defined_for_problem_type</code>	
<code>name</code>	Returns a name describing the objective.

continues on next page

Table 286 – continued from previous page

<i>objective_function</i>	Computes the relative value of the provided predictions compared to the actual labels, according a specified metric
<i>perfect_score</i>	Returns the score obtained by evaluating this objective on a perfect model.
<i>positive_only</i>	If True, this objective is only valid for positive data. Default False.
<i>score</i>	Returns a numerical score indicating performance based on the differences between the predicted and actual values.
<i>score_needs_proba</i>	Returns a boolean determining if the score() method needs probability estimates. This should be true for objectives which work with predicted probabilities, like log loss or AUC, and false for objectives which compare predicted class labels to the actual labels, like F1 or correlation.
<i>validate_inputs</i>	Validates the input based on a few simple checks.

**classmethod** `calculate_percent_difference` (*cls*, *score*, *baseline\_score*)

Calculate the percent difference between scores.

#### Parameters

- **score** (*float*) – A score. Output of the score method of this objective.
- **baseline\_score** (*float*) – A score. Output of the score method of this objective. In practice, this is the score achieved on this objective with a baseline estimator.

#### Returns

**The percent difference between the scores. Note that for objectives that can be interpreted** as percentages, this will be the difference between the reference score and score. For all other objectives, the difference will be normalized by the reference score.

**Return type** float

**property** `greater_is_better` (*cls*)

Returns a boolean determining if a greater score indicates better model performance.

**property** `is_bounded_like_percentage` (*cls*)

Returns whether this objective is bounded between 0 and 1, inclusive.

**classmethod** `is_defined_for_problem_type` (*cls*, *problem\_type*)

**property** `name` (*cls*)

Returns a name describing the objective.

**abstract classmethod** `objective_function` (*cls*, *y\_true*, *y\_predicted*, *X=None*, *sample\_weight=None*)

Computes the relative value of the provided predictions compared to the actual labels, according a specified metric

**Arguments:** *y\_predicted* (pd.Series): Predicted values of length [*n\_samples*] *y\_true* (pd.Series): Actual class labels of length [*n\_samples*] *X* (pd.DataFrame or np.ndarray): Extra data of shape [*n\_samples*, *n\_features*] necessary to calculate score *sample\_weight* (pd.DataFrame or np.ndarray): Sample weights used in computing objective value result

**Returns** Numerical value used to calculate score

**property** `perfect_score` (*cls*)

Returns the score obtained by evaluating this objective on a perfect model.

**property** `positive_only` (*cls*)

If True, this objective is only valid for positive data. Default False.

**score** (*self*, *y\_true*, *y\_predicted*, *X=None*, *sample\_weight=None*)

Returns a numerical score indicating performance based on the differences between the predicted and actual values.

#### Parameters

- **y\_predicted** (*pd.Series*) – Predicted values of length [n\_samples]
- **y\_true** (*pd.Series*) – Actual class labels of length [n\_samples]
- **X** (*pd.DataFrame* or *np.ndarray*) – Extra data of shape [n\_samples, n\_features] necessary to calculate score
- **sample\_weight** (*pd.DataFrame* or *np.ndarray*) – Sample weights used in computing objective value result

**Returns** score

**property** `score_needs_proba` (*cls*)

Returns a boolean determining if the `score()` method needs probability estimates. This should be true for objectives which work with predicted probabilities, like log loss or AUC, and false for objectives which compare predicted class labels to the actual labels, like F1 or correlation.

**validate\_inputs** (*self*, *y\_true*, *y\_predicted*)

Validates the input based on a few simple checks.

#### Parameters

- **y\_predicted** (*pd.Series*, or *pd.DataFrame*) – Predicted values of length [n\_samples]
- **y\_true** (*pd.Series*) – Actual class labels of length [n\_samples]

**Returns** None

**class** `evalml.objectives.RootMeanSquaredError`

Root mean squared error for regression.

#### Attributes

<b>greater_is_better</b>	False
<b>is_bounded_like_percentage</b>	False
<b>name</b>	Root Mean Squared Error
<b>perfect_score</b>	0.0
<b>problem_types</b>	[ProblemTypes.REGRESSION, ProblemTypes.TIME_SERIES_REGRESSION]
<b>score_needs_proba</b>	False

#### Methods

<code>calculate_percent_difference</code>	Calculate the percent difference between scores.
<code>is_defined_for_problem_type</code>	

---

continues on next page

Table 287 – continued from previous page

<i>objective_function</i>	Computes the relative value of the provided predictions compared to the actual labels, according a specified metric
<i>positive_only</i>	If True, this objective is only valid for positive data. Default False.
<i>score</i>	Returns a numerical score indicating performance based on the differences between the predicted and actual values.
<i>validate_inputs</i>	Validates the input based on a few simple checks.

**classmethod** `calculate_percent_difference` (*cls*, *score*, *baseline\_score*)

Calculate the percent difference between scores.

#### Parameters

- **score** (*float*) – A score. Output of the score method of this objective.
- **baseline\_score** (*float*) – A score. Output of the score method of this objective. In practice, this is the score achieved on this objective with a baseline estimator.

#### Returns

**The percent difference between the scores. Note that for objectives that can be interpreted** as percentages, this will be the difference between the reference score and score. For all other objectives, the difference will be normalized by the reference score.

**Return type** float

**classmethod** `is_defined_for_problem_type` (*cls*, *problem\_type*)

**objective\_function** (*self*, *y\_true*, *y\_predicted*, *X=None*, *sample\_weight=None*)

Computes the relative value of the provided predictions compared to the actual labels, according a specified metric

**Arguments:** *y\_predicted* (*pd.Series*): Predicted values of length [*n\_samples*] *y\_true* (*pd.Series*): Actual class labels of length [*n\_samples*] *X* (*pd.DataFrame* or *np.ndarray*): Extra data of shape [*n\_samples*, *n\_features*] necessary to calculate score *sample\_weight* (*pd.DataFrame* or *np.ndarray*): Sample weights used in computing objective value result

**Returns** Numerical value used to calculate score

**positive\_only** (*cls*)

If True, this objective is only valid for positive data. Default False.

**score** (*self*, *y\_true*, *y\_predicted*, *X=None*, *sample\_weight=None*)

Returns a numerical score indicating performance based on the differences between the predicted and actual values.

#### Parameters

- **y\_predicted** (*pd.Series*) – Predicted values of length [*n\_samples*]
- **y\_true** (*pd.Series*) – Actual class labels of length [*n\_samples*]
- **X** (*pd.DataFrame* or *np.ndarray*) – Extra data of shape [*n\_samples*, *n\_features*] necessary to calculate score
- **sample\_weight** (*pd.DataFrame* or *np.ndarray*) – Sample weights used in computing objective value result

**Returns** score

**validate\_inputs** (*self*, *y\_true*, *y\_predicted*)

Validates the input based on a few simple checks.

**Parameters**

- **y\_predicted** (*pd.Series*, or *pd.DataFrame*) – Predicted values of length [n\_samples]
- **y\_true** (*pd.Series*) – Actual class labels of length [n\_samples]

**Returns** None

**class** evalml.objectives.**RootMeanSquaredLogError**

Root mean squared log error for regression.

Only valid for nonnegative inputs. Otherwise, will throw a ValueError.

**Attributes**

<b>greater_is_better</b>	False
<b>is_bounded_like_percentage</b>	False
<b>name</b>	Root Mean Squared Log Error
<b>perfect_score</b>	0.0
<b>problem_types</b>	[ProblemTypes.REGRESSION, ProblemTypes.TIME_SERIES_REGRESSION]
<b>score_needs_positive</b>	True

**Methods**

<i>calculate_percent_difference</i>	Calculate the percent difference between scores.
<i>is_defined_for_problem_type</i>	
<i>objective_function</i>	Computes the relative value of the provided predictions compared to the actual labels, according a specified metric
<i>positive_only</i>	If True, this objective is only valid for positive data. Default False.
<i>score</i>	Returns a numerical score indicating performance based on the differences between the predicted and actual values.
<i>validate_inputs</i>	Validates the input based on a few simple checks.

**classmethod** **calculate\_percent\_difference** (*cls*, *score*, *baseline\_score*)

Calculate the percent difference between scores.

**Parameters**

- **score** (*float*) – A score. Output of the score method of this objective.
- **baseline\_score** (*float*) – A score. Output of the score method of this objective. In practice, this is the score achieved on this objective with a baseline estimator.

**Returns**

**The percent difference between the scores. Note that for objectives that can be interpreted as percentages, this will be the difference between the reference score and score. For all other objectives, the difference will be normalized by the reference score.**

**Return type** float

**classmethod** `is_defined_for_problem_type(cls, problem_type)`

**objective\_function** (*self*, *y\_true*, *y\_predicted*, *X=None*, *sample\_weight=None*)

Computes the relative value of the provided predictions compared to the actual labels, according a specified metric

**Arguments:** *y\_predicted* (pd.Series): Predicted values of length [n\_samples] *y\_true* (pd.Series): Actual class labels of length [n\_samples] *X* (pd.DataFrame or np.ndarray): Extra data of shape [n\_samples, n\_features] necessary to calculate score *sample\_weight* (pd.DataFrame or np.ndarray): Sample weights used in computing objective value result

**Returns** Numerical value used to calculate score

**positive\_only** (*self*)

If True, this objective is only valid for positive data. Default False.

**score** (*self*, *y\_true*, *y\_predicted*, *X=None*, *sample\_weight=None*)

Returns a numerical score indicating performance based on the differences between the predicted and actual values.

**Parameters**

- **y\_predicted** (pd.Series) – Predicted values of length [n\_samples]
- **y\_true** (pd.Series) – Actual class labels of length [n\_samples]
- **X** (pd.DataFrame or np.ndarray) – Extra data of shape [n\_samples, n\_features] necessary to calculate score
- **sample\_weight** (pd.DataFrame or np.ndarray) – Sample weights used in computing objective value result

**Returns** score

**validate\_inputs** (*self*, *y\_true*, *y\_predicted*)

Validates the input based on a few simple checks.

**Parameters**

- **y\_predicted** (pd.Series, or pd.DataFrame) – Predicted values of length [n\_samples]
- **y\_true** (pd.Series) – Actual class labels of length [n\_samples]

**Returns** None

**class** evalml.objectives.**SensitivityLowAlert** (*alert\_rate=0.01*)

Base class for all binary classification objectives.

**Attributes**

<b>greater_is_better</b>	True
<b>is_bounded_like_percentage</b>	True
<b>name</b>	Sensitivity at Low Alert Rates
<b>perfect_score</b>	1.0
<b>problem_types</b>	[ProblemTypes.BINARY, ProblemTypes.TIME_SERIES_BINARY]
<b>score_needs_proba</b>	False

**Methods**

<code>calculate_percent_difference</code>	Calculate the percent difference between scores.
<code>can_optimize_threshold</code>	Returns a boolean determining if we can optimize the binary classification objective threshold.
<code>decision_function</code>	Determine if an observation is high risk given an alert rate
<code>is_defined_for_problem_type</code>	
<code>objective_function</code>	Calculate sensitivity across all predictions, using the top alert_rate percent of observations as the predicted positive class
<code>optimize_threshold</code>	Learn a binary classification threshold which optimizes the current objective.
<code>positive_only</code>	If True, this objective is only valid for positive data. Default False.
<code>score</code>	Returns a numerical score indicating performance based on the differences between the predicted and actual values.
<code>validate_inputs</code>	Validates the input based on a few simple checks.

**classmethod** `calculate_percent_difference` (*cls*, *score*, *baseline\_score*)

Calculate the percent difference between scores.

**Parameters**

- **score** (*float*) – A score. Output of the score method of this objective.
- **baseline\_score** (*float*) – A score. Output of the score method of this objective. In practice, this is the score achieved on this objective with a baseline estimator.

**Returns**

**The percent difference between the scores. Note that for objectives that can be interpreted as percentages, this will be the difference between the reference score and score. For all other objectives, the difference will be normalized by the reference score.**

**Return type** float

**property** `can_optimize_threshold` (*cls*)

Returns a boolean determining if we can optimize the binary classification objective threshold. This will be false for any objective that works directly with predicted probabilities, like log loss and AUC. Otherwise, it will be true.

**decision\_function** (*self*, *ypred\_proba*, *\*\*kwargs*)

Determine if an observation is high risk given an alert rate

**Parameters** *ypred\_proba* (*pd.Series*) – Predicted probabilities

**classmethod** `is_defined_for_problem_type` (*cls*, *problem\_type*)

**objective\_function** (*self*, *y\_true*, *y\_predicted*, *\*\*kwargs*)

Calculate sensitivity across all predictions, using the top alert\_rate percent of observations as the predicted positive class

**Parameters**

- **y\_true** (*pd.Series*) – True labels
- **y\_predicted** (*pd.Series*) – Predicted labels based on alert\_rate



**Returns** sensitivity using the observations with the top scores as the predicted positive class

**Return type** float

**optimize\_threshold** (*self*, *ypred\_proba*, *y\_true*, *X=None*)

Learn a binary classification threshold which optimizes the current objective.

**Parameters**

- **ypred\_proba** (*pd.Series*) – The classifier’s predicted probabilities
- **y\_true** (*pd.Series*) – The ground truth for the predictions.
- **X** (*pd.DataFrame*, *optional*) – Any extra columns that are needed from training data.

**Returns** Optimal threshold for this objective

**positive\_only** (*cls*)

If True, this objective is only valid for positive data. Default False.

**score** (*self*, *y\_true*, *y\_predicted*, *X=None*, *sample\_weight=None*)

Returns a numerical score indicating performance based on the differences between the predicted and actual values.

**Parameters**

- **y\_predicted** (*pd.Series*) – Predicted values of length [n\_samples]
- **y\_true** (*pd.Series*) – Actual class labels of length [n\_samples]
- **X** (*pd.DataFrame* or *np.ndarray*) – Extra data of shape [n\_samples, n\_features] necessary to calculate score
- **sample\_weight** (*pd.DataFrame* or *np.ndarray*) – Sample weights used in computing objective value result

**Returns** score

**validate\_inputs** (*self*, *y\_true*, *y\_predicted*)

Validates the input based on a few simple checks.

**Parameters**

- **y\_predicted** (*pd.Series*, or *pd.DataFrame*) – Predicted values of length [n\_samples]
- **y\_true** (*pd.Series*) – Actual class labels of length [n\_samples]

**Returns** None

## Pipelines

## Subpackages

## components

## Subpackages

## ensemble

## Submodules

### sklearn\_stacked\_ensemble\_base

## Module Contents

## Classes Summary

---

<i>SklearnStackedEnsembleBase</i>	Stacked Ensemble Base Class.
-----------------------------------	------------------------------

---

## Contents

**class** evalml.pipelines.components.ensemble.sklearn\_stacked\_ensemble\_base.**SklearnStackedEnse**

Stacked Ensemble Base Class.

### Parameters

- **input\_pipelines** (*list(PipelineBase or subclass obj)*) – List of pipeline instances to use as the base estimators. This must not be None or an empty list or else EnsembleMissingPipelinesError will be raised.
- **final\_estimator** (*Estimator or subclass*) – The estimator used to combine the base estimators.
- **cv** (*int, cross-validation generator or an iterable*) – Determines the cross-validation splitting strategy used to train final\_estimator. For int/None inputs, if the estimator is a classifier and y is either binary or multiclass, StratifiedKFold is used. In all other cases, KFold is used. Possible inputs for cv are:
  - None: 5-fold cross validation
  - int: the number of folds in a (Stratified) KFold
  - An scikit-learn cross-validation generator object
  - An iterable yielding (train, test) splits
- **n\_jobs** (*int or None*) – Non-negative integer describing level of parallelism used for pipelines. None and 1 are equivalent. If set to -1, all CPUs are used. For n\_jobs below -1, (n\_cpus + 1 + n\_jobs) are used. Defaults to -1. - Note: there could be some multi-process errors thrown for values of *n\_jobs*  $\neq 1$ . If this is the case, please use *n\_jobs* = 1.
- **random\_seed** (*int*) – Seed for the random number generator. Defaults to 0.

### Attributes

<b>model_family</b>	ModelFamily.ENSEMBLE
<b>modifies_features</b>	True
<b>modifies_target</b>	False
<b>pre-dict_uses_y</b>	False

## Methods

<i>clone</i>	Constructs a new component with the same parameters and random state.
<i>default_parameters</i>	Returns the default parameters for stacked ensemble classes.
<i>describe</i>	Describe a component and its parameters
<i>feature_importance</i>	Not implemented for SklearnStackedEnsembleClassifier and SklearnStackedEnsembleRegressor
<i>fit</i>	Fits component to data
<i>load</i>	Loads component at file path
<i>name</i>	Returns string name of this component
<i>needs_fitting</i>	Returns boolean determining if component needs fitting before
<i>parameters</i>	Returns the parameters which were used to initialize the component
<i>predict</i>	Make predictions using selected features.
<i>predict_proba</i>	Make probability estimates for labels.
<i>save</i>	Saves component at file path
<i>supported_problem_types</i>	Problem types this estimator supports

### **clone** (*self*)

Constructs a new component with the same parameters and random state.

**Returns** A new instance of this component with identical parameters and random state.

### **default\_parameters** (*cls*)

Returns the default parameters for stacked ensemble classes.

**Returns** default parameters for this component.

**Return type** dict

### **describe** (*self*, *print\_name=False*, *return\_dict=False*)

Describe a component and its parameters

#### Parameters

- **print\_name** (*bool*, *optional*) – whether to print name of component
- **return\_dict** (*bool*, *optional*) – whether to return description as dictionary in the format {"name": name, "parameters": parameters}

**Returns** prints and returns dictionary

**Return type** None or dict

### **property feature\_importance** (*self*)

Not implemented for SklearnStackedEnsembleClassifier and SklearnStackedEnsembleRegressor

**fit** (*self*, *X*, *y=None*)

Fits component to data

**Parameters**

- **X** (*list*, *pd.DataFrame* or *np.ndarray*) – The input training data of shape [n\_samples, n\_features]
- **y** (*list*, *pd.Series*, *np.ndarray*, *optional*) – The target training data of length [n\_samples]

**Returns** *self*

**static load** (*file\_path*)

Loads component at file path

**Parameters** **file\_path** (*str*) – Location to load file

**Returns** ComponentBase object

**property name** (*cls*)

Returns string name of this component

**needs\_fitting** (*self*)

Returns boolean determining if component needs fitting before calling predict, predict\_proba, transform, or feature\_importances. This can be overridden to False for components that do not need to be fit or whose fit methods do nothing.

**property parameters** (*self*)

Returns the parameters which were used to initialize the component

**predict** (*self*, *X*)

Make predictions using selected features.

**Parameters** **X** (*pd.DataFrame*, *np.ndarray*) – Data of shape [n\_samples, n\_features]

**Returns** Predicted values

**Return type** *pd.Series*

**predict\_proba** (*self*, *X*)

Make probability estimates for labels.

**Parameters** **X** (*pd.DataFrame*, or *np.ndarray*) – Features

**Returns** Probability estimates

**Return type** *pd.Series*

**save** (*self*, *file\_path*, *pickle\_protocol=cloudpickle.DEFAULT\_PROTOCOL*)

Saves component at file path

**Parameters**

- **file\_path** (*str*) – Location to save file
- **pickle\_protocol** (*int*) – The pickle data stream format.

**Returns** None

**property supported\_problem\_types** (*cls*)

Problem types this estimator supports

## sklearn\_stacked\_ensemble\_classifier

### Module Contents

### Classes Summary

---

<i>SklearnStackedEnsembleClassifier</i>	Scikit-learn Stacked Ensemble Classifier.
---	---

---

### Contents

**class** evalml.pipelines.components.ensemble.sklearn\_stacked\_ensemble\_classifier.**SklearnStackedEnsembleClassifier**

Scikit-learn Stacked Ensemble Classifier.

#### Parameters

- **input\_pipelines** (*list(PipelineBase or subclass obj)*) – List of pipeline instances to use as the base estimators. This must not be None or an empty list or else EnsembleMissingPipelinesError will be raised.
- **final\_estimator** (*Estimator or subclass*) – The classifier used to combine the base estimators. If None, uses LogisticRegressionClassifier.
- **cv** (*int, cross-validation generator or an iterable*) – Determines the cross-validation splitting strategy used to train final\_estimator. For int/None inputs, if the estimator is a classifier and y is either binary or multiclass, StratifiedKFold is used. Defaults to None. Possible inputs for cv are:
  - None: 3-fold cross validation
  - int: the number of folds in a (Stratified) KFold
  - An scikit-learn cross-validation generator object
  - An iterable yielding (train, test) splits
- **n\_jobs** (*int or None*) – Non-negative integer describing level of parallelism used for pipelines. None and 1 are equivalent. If set to -1, all CPUs are used. For n\_jobs below -1, (n\_cpus + 1 + n\_jobs) are used. Defaults to -1. - Note: there could be some multi-process errors thrown for values of *n\_jobs* != 1. If this is the case, please use *n\_jobs* = 1.
- **random\_seed** (*int*) – Seed for the random number generator. Defaults to 0.

#### Attributes

<b>hyper-parameter_ranges</b>	{}
<b>model_family</b>	ModelFamily.ENSEMBLE
<b>modifies_features</b>	True
<b>modifies_target</b>	False
<b>name</b>	Sklearn Stacked Ensemble Classifier
<b>pre-dict_uses_y</b>	False
<b>supported_problem_types</b>	[ProblemTypes.BINARY, ProblemTypes.MULTICLASS, ProblemTypes.TIME_SERIES_BINARY, ProblemTypes.TIME_SERIES_MULTICLASS,]

## Methods

<i>clone</i>	Constructs a new component with the same parameters and random state.
<i>default_parameters</i>	Returns the default parameters for stacked ensemble classes.
<i>describe</i>	Describe a component and its parameters
<i>feature_importance</i>	Not implemented for SklearnStackedEnsembleClassifier and SklearnStackedEnsembleRegressor
<i>fit</i>	Fits component to data
<i>load</i>	Loads component at file path
<i>needs_fitting</i>	Returns boolean determining if component needs fitting before
<i>parameters</i>	Returns the parameters which were used to initialize the component
<i>predict</i>	Make predictions using selected features.
<i>predict_proba</i>	Make probability estimates for labels.
<i>save</i>	Saves component at file path

### **clone** (*self*)

Constructs a new component with the same parameters and random state.

**Returns** A new instance of this component with identical parameters and random state.

### **default\_parameters** (*cls*)

Returns the default parameters for stacked ensemble classes.

**Returns** default parameters for this component.

**Return type** dict

### **describe** (*self*, *print\_name=False*, *return\_dict=False*)

Describe a component and its parameters

#### Parameters

- **print\_name** (*bool*, *optional*) – whether to print name of component
- **return\_dict** (*bool*, *optional*) – whether to return description as dictionary in the format {"name": name, "parameters": parameters}

**Returns** prints and returns dictionary

**Return type** None or dict

**property feature\_importance** (*self*)

Not implemented for SklearnStackedEnsembleClassifier and SklearnStackedEnsembleRegressor

**fit** (*self*, *X*, *y=None*)

Fits component to data

**Parameters**

- **X** (*list*, *pd.DataFrame* or *np.ndarray*) – The input training data of shape [n\_samples, n\_features]
- **y** (*list*, *pd.Series*, *np.ndarray*, *optional*) – The target training data of length [n\_samples]

**Returns** *self*

**static load** (*file\_path*)

Loads component at file path

**Parameters** **file\_path** (*str*) – Location to load file

**Returns** ComponentBase object

**needs\_fitting** (*self*)

Returns boolean determining if component needs fitting before calling predict, predict\_proba, transform, or feature\_importances. This can be overridden to False for components that do not need to be fit or whose fit methods do nothing.

**property parameters** (*self*)

Returns the parameters which were used to initialize the component

**predict** (*self*, *X*)

Make predictions using selected features.

**Parameters** **X** (*pd.DataFrame*, *np.ndarray*) – Data of shape [n\_samples, n\_features]

**Returns** Predicted values

**Return type** *pd.Series*

**predict\_proba** (*self*, *X*)

Make probability estimates for labels.

**Parameters** **X** (*pd.DataFrame*, or *np.ndarray*) – Features

**Returns** Probability estimates

**Return type** *pd.Series*

**save** (*self*, *file\_path*, *pickle\_protocol=cloudpickle.DEFAULT\_PROTOCOL*)

Saves component at file path

**Parameters**

- **file\_path** (*str*) – Location to save file
- **pickle\_protocol** (*int*) – The pickle data stream format.

**Returns** None

## sklearn\_stacked\_ensemble\_regressor

### Module Contents

### Classes Summary

---

<a href="#"><i>SklearnStackedEnsembleRegressor</i></a>	Scikit-learn Stacked Ensemble Regressor.
--	--

---

### Contents

**class** evalml.pipelines.components.ensemble.sklearn\_stacked\_ensemble\_regressor.**SklearnStackedEnsembleRegressor**

Scikit-learn Stacked Ensemble Regressor.

#### Parameters

- **input\_pipelines** (*list(PipelineBase or subclass obj)*) – List of pipeline instances to use as the base estimators. This must not be None or an empty list or else EnsembleMissingPipelinesError will be raised.
- **final\_estimator** (*Estimator or subclass*) – The regressor used to combine the base estimators. If None, uses LinearRegressor.
- **cv** (*int, cross-validation generator or an iterable*) – Determines the cross-validation splitting strategy used to train final\_estimator. For int/None inputs, KFold is used. Defaults to None. Possible inputs for cv are:
  - None: 3-fold cross validation
  - int: the number of folds in a (Stratified) KFold
  - An scikit-learn cross-validation generator object
  - An iterable yielding (train, test) splits
- **n\_jobs** (*int or None*) – Non-negative integer describing level of parallelism used for pipelines. None and 1 are equivalent. If set to -1, all CPUs are used. For n\_jobs below -1, (n\_cpus + 1 + n\_jobs) are used. Defaults to -1. - Note: there could be some multi-process errors thrown for values of *n\_jobs* != 1. If this is the case, please use *n\_jobs* = 1.
- **random\_seed** (*int*) – Seed for the random number generator. Defaults to 0.

#### Attributes



<b>hyper-parameter_ranges</b>	{}
<b>model_family</b>	ModelFamily.ENSEMBLE
<b>modifies_features</b>	True
<b>modifies_target</b>	False
<b>name</b>	Sklearn Stacked Ensemble Regressor
<b>pre-dict_uses_y</b>	False
<b>supported_problem_types</b>	[ ProblemTypes.REGRESSION, ProblemTypes.TIME_SERIES_REGRESSION,]

## Methods

<i>clone</i>	Constructs a new component with the same parameters and random state.
<i>default_parameters</i>	Returns the default parameters for stacked ensemble classes.
<i>describe</i>	Describe a component and its parameters
<i>feature_importance</i>	Not implemented for SklearnStackedEnsembleClassifier and SklearnStackedEnsembleRegressor
<i>fit</i>	Fits component to data
<i>load</i>	Loads component at file path
<i>needs_fitting</i>	Returns boolean determining if component needs fitting before
<i>parameters</i>	Returns the parameters which were used to initialize the component
<i>predict</i>	Make predictions using selected features.
<i>predict_proba</i>	Make probability estimates for labels.
<i>save</i>	Saves component at file path

### **clone** (*self*)

Constructs a new component with the same parameters and random state.

**Returns** A new instance of this component with identical parameters and random state.

### **default\_parameters** (*cls*)

Returns the default parameters for stacked ensemble classes.

**Returns** default parameters for this component.

**Return type** dict

### **describe** (*self*, *print\_name=False*, *return\_dict=False*)

Describe a component and its parameters

#### Parameters

- **print\_name** (*bool*, *optional*) – whether to print name of component
- **return\_dict** (*bool*, *optional*) – whether to return description as dictionary in the format {"name": name, "parameters": parameters}

**Returns** prints and returns dictionary

**Return type** None or dict

**property feature\_importance** (*self*)

Not implemented for SklearnStackedEnsembleClassifier and SklearnStackedEnsembleRegressor

**fit** (*self*, *X*, *y=None*)

Fits component to data

**Parameters**

- **X** (*list*, *pd.DataFrame* or *np.ndarray*) – The input training data of shape [n\_samples, n\_features]
- **y** (*list*, *pd.Series*, *np.ndarray*, *optional*) – The target training data of length [n\_samples]

**Returns** *self*

**static load** (*file\_path*)

Loads component at file path

**Parameters** **file\_path** (*str*) – Location to load file

**Returns** ComponentBase object

**needs\_fitting** (*self*)

Returns boolean determining if component needs fitting before calling predict, predict\_proba, transform, or feature\_importances. This can be overridden to False for components that do not need to be fit or whose fit methods do nothing.

**property parameters** (*self*)

Returns the parameters which were used to initialize the component

**predict** (*self*, *X*)

Make predictions using selected features.

**Parameters** **X** (*pd.DataFrame*, *np.ndarray*) – Data of shape [n\_samples, n\_features]

**Returns** Predicted values

**Return type** *pd.Series*

**predict\_proba** (*self*, *X*)

Make probability estimates for labels.

**Parameters** **X** (*pd.DataFrame*, or *np.ndarray*) – Features

**Returns** Probability estimates

**Return type** *pd.Series*

**save** (*self*, *file\_path*, *pickle\_protocol=cloudpickle.DEFAULT\_PROTOCOL*)

Saves component at file path

**Parameters**

- **file\_path** (*str*) – Location to save file
- **pickle\_protocol** (*int*) – The pickle data stream format.

**Returns** None

## Package Contents

### Classes Summary

<i>SklearnStackedEnsembleBase</i>	Stacked Ensemble Base Class.
<i>SklearnStackedEnsembleClassifier</i>	Scikit-learn Stacked Ensemble Classifier.
<i>SklearnStackedEnsembleRegressor</i>	Scikit-learn Stacked Ensemble Regressor.

### Contents

**class** evalml.pipelines.components.ensemble.**SklearnStackedEnsembleBase** (*input\_pipelines=None, final\_estimator=None, cv=None, n\_jobs=-1, random\_seed=0, \*\*kwargs*)

Stacked Ensemble Base Class.

#### Parameters

- **input\_pipelines** (*list (PipelineBase or subclass obj)*) – List of pipeline instances to use as the base estimators. This must not be None or an empty list or else EnsembleMissingPipelinesError will be raised.
- **final\_estimator** (*Estimator or subclass*) – The estimator used to combine the base estimators.
- **cv** (*int, cross-validation generator or an iterable*) – Determines the cross-validation splitting strategy used to train final\_estimator. For int/None inputs, if the estimator is a classifier and y is either binary or multiclass, StratifiedKFold is used. In all other cases, KFold is used. Possible inputs for cv are:
  - None: 5-fold cross validation
  - int: the number of folds in a (Stratified) KFold
  - An scikit-learn cross-validation generator object
  - An iterable yielding (train, test) splits
- **n\_jobs** (*int or None*) – Non-negative integer describing level of parallelism used for pipelines. None and 1 are equivalent. If set to -1, all CPUs are used. For n\_jobs below -1, (n\_cpus + 1 + n\_jobs) are used. Defaults to -1. - Note: there could be some multi-process errors thrown for values of *n\_jobs* != 1. If this is the case, please use *n\_jobs* = 1.
- **random\_seed** (*int*) – Seed for the random number generator. Defaults to 0.

#### Attributes

<b>model_family</b>	ModelFamily.ENSEMBLE
<b>modifies_features</b>	True
<b>modifies_target</b>	False
<b>pre_dict_uses_y</b>	False

## Methods

<i>clone</i>	Constructs a new component with the same parameters and random state.
<i>default_parameters</i>	Returns the default parameters for stacked ensemble classes.
<i>describe</i>	Describe a component and its parameters
<i>feature_importance</i>	Not implemented for SklearnStackedEnsembleClassifier and SklearnStackedEnsembleRegressor
<i>fit</i>	Fits component to data
<i>load</i>	Loads component at file path
<i>name</i>	Returns string name of this component
<i>needs_fitting</i>	Returns boolean determining if component needs fitting before
<i>parameters</i>	Returns the parameters which were used to initialize the component
<i>predict</i>	Make predictions using selected features.
<i>predict_proba</i>	Make probability estimates for labels.
<i>save</i>	Saves component at file path
<i>supported_problem_types</i>	Problem types this estimator supports

### **clone** (*self*)

Constructs a new component with the same parameters and random state.

**Returns** A new instance of this component with identical parameters and random state.

### **default\_parameters** (*cls*)

Returns the default parameters for stacked ensemble classes.

**Returns** default parameters for this component.

**Return type** dict

### **describe** (*self*, *print\_name=False*, *return\_dict=False*)

Describe a component and its parameters

#### **Parameters**

- **print\_name** (*bool*, *optional*) – whether to print name of component
- **return\_dict** (*bool*, *optional*) – whether to return description as dictionary in the format {"name": name, "parameters": parameters}

**Returns** prints and returns dictionary

**Return type** None or dict

### **property feature\_importance** (*self*)

Not implemented for SklearnStackedEnsembleClassifier and SklearnStackedEnsembleRegressor

**fit** (*self*, *X*, *y=None*)

Fits component to data

**Parameters**

- **X** (*list*, *pd.DataFrame* or *np.ndarray*) – The input training data of shape [n\_samples, n\_features]
- **y** (*list*, *pd.Series*, *np.ndarray*, *optional*) – The target training data of length [n\_samples]

**Returns** *self*

**static load** (*file\_path*)

Loads component at file path

**Parameters** **file\_path** (*str*) – Location to load file

**Returns** ComponentBase object

**property name** (*cls*)

Returns string name of this component

**needs\_fitting** (*self*)

Returns boolean determining if component needs fitting before calling predict, predict\_proba, transform, or feature\_importances. This can be overridden to False for components that do not need to be fit or whose fit methods do nothing.

**property parameters** (*self*)

Returns the parameters which were used to initialize the component

**predict** (*self*, *X*)

Make predictions using selected features.

**Parameters** **X** (*pd.DataFrame*, *np.ndarray*) – Data of shape [n\_samples, n\_features]

**Returns** Predicted values

**Return type** *pd.Series*

**predict\_proba** (*self*, *X*)

Make probability estimates for labels.

**Parameters** **X** (*pd.DataFrame*, or *np.ndarray*) – Features

**Returns** Probability estimates

**Return type** *pd.Series*

**save** (*self*, *file\_path*, *pickle\_protocol=cloudpickle.DEFAULT\_PROTOCOL*)

Saves component at file path

**Parameters**

- **file\_path** (*str*) – Location to save file
- **pickle\_protocol** (*int*) – The pickle data stream format.

**Returns** None

**property supported\_problem\_types** (*cls*)

Problem types this estimator supports

```

class evalml.pipelines.components.ensemble.SklearnStackedEnsembleClassifier (input_pipelines=None,
final_estimator=None,
cv=None,
n_jobs=-1,
random_seed=0,
**kwargs)

```

Scikit-learn Stacked Ensemble Classifier.

### Parameters

- **input\_pipelines** (*list(PipelineBase or subclass obj)*) – List of pipeline instances to use as the base estimators. This must not be None or an empty list or else EnsembleMissingPipelinesError will be raised.
- **final\_estimator** (*Estimator or subclass*) – The classifier used to combine the base estimators. If None, uses LogisticRegressionClassifier.
- **cv** (*int, cross-validation generator or an iterable*) – Determines the cross-validation splitting strategy used to train final\_estimator. For int/None inputs, if the estimator is a classifier and y is either binary or multiclass, StratifiedKFold is used. Defaults to None. Possible inputs for cv are:
  - None: 3-fold cross validation
  - int: the number of folds in a (Stratified) KFold
  - An scikit-learn cross-validation generator object
  - An iterable yielding (train, test) splits
- **n\_jobs** (*int or None*) – Non-negative integer describing level of parallelism used for pipelines. None and 1 are equivalent. If set to -1, all CPUs are used. For n\_jobs below -1, (n\_cpus + 1 + n\_jobs) are used. Defaults to -1. - Note: there could be some multi-process errors thrown for values of *n\_jobs* != 1. If this is the case, please use *n\_jobs* = 1.
- **random\_seed** (*int*) – Seed for the random number generator. Defaults to 0.

### Attributes

hyper-parameter_ranges	{}
model_family	ModelFamily.ENSEMBLE
modifies_features	True
modifies_target	False
name	Sklearn Stacked Ensemble Classifier
pre-dict_uses_y	False
supported_problem_types	[ ProblemTypes.BINARY, ProblemTypes.MULTICLASS, ProblemTypes.TIME_SERIES_BINARY, ProblemTypes.TIME_SERIES_MULTICLASS,]

### Methods

<code>clone</code>	Constructs a new component with the same parameters and random state.
<code>default_parameters</code>	Returns the default parameters for stacked ensemble classes.
<code>describe</code>	Describe a component and its parameters
<code>feature_importance</code>	Not implemented for SklearnStackedEnsembleClassifier and SklearnStackedEnsembleRegressor
<code>fit</code>	Fits component to data
<code>load</code>	Loads component at file path
<code>needs_fitting</code>	Returns boolean determining if component needs fitting before
<code>parameters</code>	Returns the parameters which were used to initialize the component
<code>predict</code>	Make predictions using selected features.
<code>predict_proba</code>	Make probability estimates for labels.
<code>save</code>	Saves component at file path

**clone** (*self*)

Constructs a new component with the same parameters and random state.

**Returns** A new instance of this component with identical parameters and random state.

**default\_parameters** (*cls*)

Returns the default parameters for stacked ensemble classes.

**Returns** default parameters for this component.

**Return type** dict

**describe** (*self*, *print\_name=False*, *return\_dict=False*)

Describe a component and its parameters

**Parameters**

- **print\_name** (*bool*, *optional*) – whether to print name of component
- **return\_dict** (*bool*, *optional*) – whether to return description as dictionary in the format {"name": name, "parameters": parameters}

**Returns** prints and returns dictionary

**Return type** None or dict

**property feature\_importance** (*self*)

Not implemented for SklearnStackedEnsembleClassifier and SklearnStackedEnsembleRegressor

**fit** (*self*, *X*, *y=None*)

Fits component to data

**Parameters**

- **X** (*list*, *pd.DataFrame* or *np.ndarray*) – The input training data of shape [n\_samples, n\_features]
- **y** (*list*, *pd.Series*, *np.ndarray*, *optional*) – The target training data of length [n\_samples]

**Returns** self

**static load** (*file\_path*)

Loads component at file path

**Parameters** `file_path` (*str*) – Location to load file

**Returns** ComponentBase object

**needs\_fitting** (*self*)

Returns boolean determining if component needs fitting before calling predict, predict\_proba, transform, or feature\_importances. This can be overridden to False for components that do not need to be fit or whose fit methods do nothing.

**property parameters** (*self*)

Returns the parameters which were used to initialize the component

**predict** (*self*, *X*)

Make predictions using selected features.

**Parameters** *X* (*pd.DataFrame*, *np.ndarray*) – Data of shape [n\_samples, n\_features]

**Returns** Predicted values

**Return type** *pd.Series*

**predict\_proba** (*self*, *X*)

Make probability estimates for labels.

**Parameters** *X* (*pd.DataFrame*, or *np.ndarray*) – Features

**Returns** Probability estimates

**Return type** *pd.Series*

**save** (*self*, *file\_path*, *pickle\_protocol=cloudpickle.DEFAULT\_PROTOCOL*)

Saves component at file path

**Parameters**

- **file\_path** (*str*) – Location to save file
- **pickle\_protocol** (*int*) – The pickle data stream format.

**Returns** None

```
class evalml.pipelines.components.ensemble.SklearnStackedEnsembleRegressor (input_pipelines=None,  
                                                                    fi-  
                                                                    nal_estimator=None,  
                                                                    cv=None,  
                                                                    n_jobs=-  
  
                                                                    1,  
                                                                    ran-  
                                                                    dom_seed=0,  
                                                                    **kwargs)
```

Scikit-learn Stacked Ensemble Regressor.

**Parameters**

- **input\_pipelines** (*list(PipelineBase or subclass obj)*) – List of pipeline instances to use as the base estimators. This must not be None or an empty list or else EnsembleMissingPipelinesError will be raised.
- **final\_estimator** (*Estimator or subclass*) – The regressor used to combine the base estimators. If None, uses LinearRegressor.
- **cv** (*int, cross-validation generator or an iterable*) – Determines the cross-validation splitting strategy used to train final\_estimator. For int/None inputs, KFold is used. Defaults to None. Possible inputs for cv are:



- None: 3-fold cross validation
- int: the number of folds in a (Stratified) KFold
- An scikit-learn cross-validation generator object
- An iterable yielding (train, test) splits
- **n\_jobs** (*int or None*) – Non-negative integer describing level of parallelism used for pipelines. None and 1 are equivalent. If set to -1, all CPUs are used. For n\_jobs below -1, (n\_cpus + 1 + n\_jobs) are used. Defaults to -1. - Note: there could be some multi-process errors thrown for values of *n\_jobs*  $\neq 1$ . If this is the case, please use *n\_jobs* = 1.
- **random\_seed** (*int*) – Seed for the random number generator. Defaults to 0.

#### Attributes

<b>hyper-parameter_ranges</b>	{}
<b>model_family</b>	ModelFamily.ENSEMBLE
<b>modifies_features</b>	True
<b>modifies_target</b>	False
<b>name</b>	Sklearn Stacked Ensemble Regressor
<b>predict_uses_y</b>	False
<b>supported_problem_types</b>	[ ProblemTypes.REGRESSION, ProblemTypes.TIME_SERIES_REGRESSION,]

#### Methods

<i>clone</i>	Constructs a new component with the same parameters and random state.
<i>default_parameters</i>	Returns the default parameters for stacked ensemble classes.
<i>describe</i>	Describe a component and its parameters
<i>feature_importance</i>	Not implemented for SklearnStackedEnsembleClassifier and SklearnStackedEnsembleRegressor
<i>fit</i>	Fits component to data
<i>load</i>	Loads component at file path
<i>needs_fitting</i>	Returns boolean determining if component needs fitting before
<i>parameters</i>	Returns the parameters which were used to initialize the component
<i>predict</i>	Make predictions using selected features.
<i>predict_proba</i>	Make probability estimates for labels.
<i>save</i>	Saves component at file path

#### **clone** (*self*)

Constructs a new component with the same parameters and random state.

**Returns** A new instance of this component with identical parameters and random state.

#### **default\_parameters** (*cls*)

Returns the default parameters for stacked ensemble classes.

**Returns** default parameters for this component.

**Return type** dict

**describe** (*self*, *print\_name=False*, *return\_dict=False*)

Describe a component and its parameters

**Parameters**

- **print\_name** (*bool*, *optional*) – whether to print name of component
- **return\_dict** (*bool*, *optional*) – whether to return description as dictionary in the format {"name": name, "parameters": parameters}

**Returns** prints and returns dictionary

**Return type** None or dict

**property feature\_importance** (*self*)

Not implemented for SklearnStackedEnsembleClassifier and SklearnStackedEnsembleRegressor

**fit** (*self*, *X*, *y=None*)

Fits component to data

**Parameters**

- **X** (*list*, *pd.DataFrame* or *np.ndarray*) – The input training data of shape [n\_samples, n\_features]
- **y** (*list*, *pd.Series*, *np.ndarray*, *optional*) – The target training data of length [n\_samples]

**Returns** self

**static load** (*file\_path*)

Loads component at file path

**Parameters** **file\_path** (*str*) – Location to load file

**Returns** ComponentBase object

**needs\_fitting** (*self*)

Returns boolean determining if component needs fitting before calling predict, predict\_proba, transform, or feature\_importances. This can be overridden to False for components that do not need to be fit or whose fit methods do nothing.

**property parameters** (*self*)

Returns the parameters which were used to initialize the component

**predict** (*self*, *X*)

Make predictions using selected features.

**Parameters** **X** (*pd.DataFrame*, *np.ndarray*) – Data of shape [n\_samples, n\_features]

**Returns** Predicted values

**Return type** pd.Series

**predict\_proba** (*self*, *X*)

Make probability estimates for labels.

**Parameters** **X** (*pd.DataFrame*, or *np.ndarray*) – Features

**Returns** Probability estimates

**Return type** `pd.Series`

**save** (*self*, *file\_path*, *pickle\_protocol*=*cloudpickle.DEFAULT\_PROTOCOL*)  
Saves component at file path

**Parameters**

- **file\_path** (*str*) – Location to save file
- **pickle\_protocol** (*int*) – The pickle data stream format.

**Returns** `None`

**estimators**

**Subpackages**

**classifiers**

**Submodules**

**baseline\_classifier**

**Module Contents**

**Classes Summary**

---

*BaselineClassifier*

Classifier that predicts using the specified strategy.

---

**Contents**

**class** `evalml.pipelines.components.estimators.classifiers.baseline_classifier.BaselineClass`

Classifier that predicts using the specified strategy.

This is useful as a simple baseline classifier to compare with other classifiers.

**Parameters**

- **strategy** (*str*) – Method used to predict. Valid options are “mode”, “random” and “random\_weighted”. Defaults to “mode”.
- **random\_seed** (*int*) – Seed for the random number generator. Defaults to 0.

**Attributes**

<b>hyper-parameter_ranges</b>	{}
<b>model_family</b>	ModelFamily.BASELINE
<b>modifies_features</b>	True
<b>modifies_target</b>	False
<b>name</b>	Baseline Classifier
<b>pre-dict_uses_y</b>	False
<b>supported_problem_types</b>	[ProblemTypes.BINARY, ProblemTypes.MULTICLASS]

## Methods

<i>classes_</i>	Returns class labels. Will return None before fitting.
<i>clone</i>	Constructs a new component with the same parameters and random state.
<i>default_parameters</i>	Returns the default parameters for this component.
<i>describe</i>	Describe a component and its parameters
<i>feature_importance</i>	Returns importance associated with each feature. Since baseline classifiers do not use input features to calculate predictions, returns an array of zeroes.
<i>fit</i>	Fits component to data
<i>load</i>	Loads component at file path
<i>needs_fitting</i>	Returns boolean determining if component needs fitting before
<i>parameters</i>	Returns the parameters which were used to initialize the component
<i>predict</i>	Make predictions using selected features.
<i>predict_proba</i>	Make probability estimates for labels.
<i>save</i>	Saves component at file path

### **property** *classes\_*(*self*)

Returns class labels. Will return None before fitting.

**Returns** Class names

**Return type** list[str] or list(float)

### **clone**(*self*)

Constructs a new component with the same parameters and random state.

**Returns** A new instance of this component with identical parameters and random state.

### **default\_parameters**(*cls*)

Returns the default parameters for this component.

Our convention is that `Component.default_parameters == Component().parameters`.

**Returns** default parameters for this component.

**Return type** dict

**describe** (*self*, *print\_name=False*, *return\_dict=False*)

Describe a component and its parameters

**Parameters**

- **print\_name** (*bool*, *optional*) – whether to print name of component
- **return\_dict** (*bool*, *optional*) – whether to return description as dictionary in the format {"name": name, "parameters": parameters}

**Returns** prints and returns dictionary

**Return type** None or dict

**property feature\_importance** (*self*)

Returns importance associated with each feature. Since baseline classifiers do not use input features to calculate predictions, returns an array of zeroes.

**Returns** An array of zeroes

**Return type** np.ndarray (float)

**fit** (*self*, *X*, *y=None*)

Fits component to data

**Parameters**

- **X** (*list*, *pd.DataFrame* or *np.ndarray*) – The input training data of shape [n\_samples, n\_features]
- **y** (*list*, *pd.Series*, *np.ndarray*, *optional*) – The target training data of length [n\_samples]

**Returns** self

**static load** (*file\_path*)

Loads component at file path

**Parameters** **file\_path** (*str*) – Location to load file

**Returns** ComponentBase object

**needs\_fitting** (*self*)

Returns boolean determining if component needs fitting before calling predict, predict\_proba, transform, or feature\_importances. This can be overridden to False for components that do not need to be fit or whose fit methods do nothing.

**property parameters** (*self*)

Returns the parameters which were used to initialize the component

**predict** (*self*, *X*)

Make predictions using selected features.

**Parameters** **X** (*pd.DataFrame*, *np.ndarray*) – Data of shape [n\_samples, n\_features]

**Returns** Predicted values

**Return type** pd.Series

**predict\_proba** (*self*, *X*)

Make probability estimates for labels.

**Parameters** **X** (*pd.DataFrame*, *or np.ndarray*) – Features

**Returns** Probability estimates

**Return type** pd.Series

**save** (*self*, *file\_path*, *pickle\_protocol*=*cloudpickle.DEFAULT\_PROTOCOL*)  
Saves component at file path

**Parameters**

- **file\_path** (*str*) – Location to save file
- **pickle\_protocol** (*int*) – The pickle data stream format.

**Returns** None

## catboost\_classifier

### Module Contents

### Classes Summary

---

*CatBoostClassifier*

CatBoost Classifier, a classifier that uses gradient-boosting on decision trees.

---

### Contents

**class** evalml.pipelines.components.estimators.classifiers.catboost\_classifier.CatBoostClasss

CatBoost Classifier, a classifier that uses gradient-boosting on decision trees. CatBoost is an open-source library and natively supports categorical features.

For more information, check out <https://catboost.ai/>

**Parameters**

- **n\_estimators** (*float*) – The maximum number of trees to build. Defaults to 10.
- **eta** (*float*) – The learning rate. Defaults to 0.03.
- **max\_depth** (*int*) – The maximum tree depth for base learners. Defaults to 6.
- **bootstrap\_type** (*string*) – Defines the method for sampling the weights of objects. Available methods are ‘Bayesian’, ‘Bernoulli’, ‘MVS’. Defaults to None.
- **silent** (*boolean*) – Whether to use the “silent” logging mode. Defaults to True.
- **allow\_writing\_files** (*boolean*) – Whether to allow writing snapshot files while training. Defaults to False.

- **n\_jobs** (*int* or *None*) – Number of jobs to run in parallel. -1 uses all processes. Defaults to -1.
- **random\_seed** (*int*) – Seed for the random number generator. Defaults to 0.

#### Attributes

<b>hyper-parameter_ranges</b>	{ “n_estimators”: Integer(4, 100), “eta”: Real(0.000001, 1), “max_depth”: Integer(4, 10), }
<b>model_family</b>	ModelFamily.CATBOOST
<b>modifies_features</b>	True
<b>modifies_target</b>	False
<b>name</b>	CatBoost Classifier
<b>pre_dict_uses_y</b>	False
<b>supported_problem_types</b>	[ ProblemTypes.BINARY, ProblemTypes.MULTICLASS, ProblemTypes.TIME_SERIES_BINARY, ProblemTypes.TIME_SERIES_MULTICLASS, ]

#### Methods

<i>clone</i>	Constructs a new component with the same parameters and random state.
<i>default_parameters</i>	Returns the default parameters for this component.
<i>describe</i>	Describe a component and its parameters
<i>feature_importance</i>	Returns importance associated with each feature.
<i>fit</i>	Fits component to data
<i>load</i>	Loads component at file path
<i>needs_fitting</i>	Returns boolean determining if component needs fitting before
<i>parameters</i>	Returns the parameters which were used to initialize the component
<i>predict</i>	Make predictions using selected features.
<i>predict_proba</i>	Make probability estimates for labels.
<i>save</i>	Saves component at file path

#### **clone** (*self*)

Constructs a new component with the same parameters and random state.

**Returns** A new instance of this component with identical parameters and random state.

#### **default\_parameters** (*cls*)

Returns the default parameters for this component.

Our convention is that `Component.default_parameters == Component().parameters`.

**Returns** default parameters for this component.

**Return type** dict

#### **describe** (*self*, *print\_name=False*, *return\_dict=False*)

Describe a component and its parameters

**Parameters**

- **print\_name** (*bool, optional*) – whether to print name of component
- **return\_dict** (*bool, optional*) – whether to return description as dictionary in the format {"name": name, "parameters": parameters}

**Returns** prints and returns dictionary

**Return type** None or dict

**property feature\_importance** (*self*)

Returns importance associated with each feature.

**Returns** Importance associated with each feature

**Return type** np.ndarray

**fit** (*self, X, y=None*)

Fits component to data

**Parameters**

- **X** (*list, pd.DataFrame or np.ndarray*) – The input training data of shape [n\_samples, n\_features]
- **y** (*list, pd.Series, np.ndarray, optional*) – The target training data of length [n\_samples]

**Returns** self

**static load** (*file\_path*)

Loads component at file path

**Parameters** **file\_path** (*str*) – Location to load file

**Returns** ComponentBase object

**needs\_fitting** (*self*)

Returns boolean determining if component needs fitting before calling predict, predict\_proba, transform, or feature\_importances. This can be overridden to False for components that do not need to be fit or whose fit methods do nothing.

**property parameters** (*self*)

Returns the parameters which were used to initialize the component

**predict** (*self, X*)

Make predictions using selected features.

**Parameters** **X** (*pd.DataFrame, np.ndarray*) – Data of shape [n\_samples, n\_features]

**Returns** Predicted values

**Return type** pd.Series

**predict\_proba** (*self, X*)

Make probability estimates for labels.

**Parameters** **X** (*pd.DataFrame, or np.ndarray*) – Features

**Returns** Probability estimates

**Return type** pd.Series

**save** (*self, file\_path, pickle\_protocol=cloudpickle.DEFAULT\_PROTOCOL*)

Saves component at file path

**Parameters**



- **file\_path** (*str*) – Location to save file
- **pickle\_protocol** (*int*) – The pickle data stream format.

**Returns** None

## decision\_tree\_classifier

### Module Contents

### Classes Summary

---

<i>DecisionTreeClassifier</i>	Decision Tree Classifier.
-------------------------------	---------------------------

---

### Contents

**class** evalml.pipelines.components.estimators.classifiers.decision\_tree\_classifier.**DecisionTreeClassifier**

Decision Tree Classifier.

#### Parameters

- **criterion** (*{ "gini", "entropy" }*) – The function to measure the quality of a split. Supported criteria are “gini” for the Gini impurity and “entropy” for the information gain. Defaults to “gini”.
- **max\_features** (*int, float or { "auto", "sqrt", "log2" }*) – The number of features to consider when looking for the best split:
  - If int, then consider max\_features features at each split.
  - If float, then max\_features is a fraction and int(max\_features \* n\_features) features are considered at each split.
  - If “auto”, then max\_features=sqrt(n\_features).
  - If “sqrt”, then max\_features=sqrt(n\_features).
  - If “log2”, then max\_features=log2(n\_features).
  - If None, then max\_features = n\_features.

The search for a split does not stop until at least one valid partition of the node samples is found, even if it requires to effectively inspect more than max\_features features. Defaults to “auto”.

- **max\_depth** (*int*) – The maximum depth of the tree. Defaults to 6.
- **min\_samples\_split** (*int or float*) – The minimum number of samples required to split an internal node:
  - If int, then consider min\_samples\_split as the minimum number.

- If float, then `min_samples_split` is a fraction and `ceil(min_samples_split * n_samples)` are the minimum number of samples for each split.

Defaults to 2.

- **`min_weight_fraction_leaf`** (*float*) – The minimum weighted fraction of the sum total of weights (of all the input samples) required to be at a leaf node. Defaults to 0.0.
- **`random_seed`** (*int*) – Seed for the random number generator. Defaults to 0.

#### Attributes

<b>hyper-parameter_ranges</b>	{ “criterion”: [“gini”, “entropy”], “max_features”: [“auto”, “sqrt”, “log2”], “max_depth”: Integer(4, 10), }
<b>model_family</b>	ModelFamily.DECISION_TREE
<b>modifies_features</b>	True
<b>modifies_target</b>	False
<b>name</b>	Decision Tree Classifier
<b>predict_uses_y</b>	False
<b>supported_problem_types</b>	[ ProblemTypes.BINARY, ProblemTypes.MULTICLASS, ProblemTypes.TIME_SERIES_BINARY, ProblemTypes.TIME_SERIES_MULTICLASS, ]

#### Methods

<i>clone</i>	Constructs a new component with the same parameters and random state.
<i>default_parameters</i>	Returns the default parameters for this component.
<i>describe</i>	Describe a component and its parameters
<i>feature_importance</i>	Returns importance associated with each feature.
<i>fit</i>	Fits component to data
<i>load</i>	Loads component at file path
<i>needs_fitting</i>	Returns boolean determining if component needs fitting before
<i>parameters</i>	Returns the parameters which were used to initialize the component
<i>predict</i>	Make predictions using selected features.
<i>predict_proba</i>	Make probability estimates for labels.
<i>save</i>	Saves component at file path

#### **`clone`** (*self*)

Constructs a new component with the same parameters and random state.

**Returns** A new instance of this component with identical parameters and random state.

#### **`default_parameters`** (*cls*)

Returns the default parameters for this component.

Our convention is that `Component.default_parameters == Component().parameters`.

**Returns** default parameters for this component.

**Return type** dict

**describe** (*self*, *print\_name=False*, *return\_dict=False*)

Describe a component and its parameters

**Parameters**

- **print\_name** (*bool*, *optional*) – whether to print name of component
- **return\_dict** (*bool*, *optional*) – whether to return description as dictionary in the format {"name": name, "parameters": parameters}

**Returns** prints and returns dictionary

**Return type** None or dict

**property feature\_importance** (*self*)

Returns importance associated with each feature.

**Returns** Importance associated with each feature

**Return type** np.ndarray

**fit** (*self*, *X*, *y=None*)

Fits component to data

**Parameters**

- **X** (*list*, *pd.DataFrame* or *np.ndarray*) – The input training data of shape [n\_samples, n\_features]
- **y** (*list*, *pd.Series*, *np.ndarray*, *optional*) – The target training data of length [n\_samples]

**Returns** self

**static load** (*file\_path*)

Loads component at file path

**Parameters** **file\_path** (*str*) – Location to load file

**Returns** ComponentBase object

**needs\_fitting** (*self*)

Returns boolean determining if component needs fitting before calling predict, predict\_proba, transform, or feature\_importances. This can be overridden to False for components that do not need to be fit or whose fit methods do nothing.

**property parameters** (*self*)

Returns the parameters which were used to initialize the component

**predict** (*self*, *X*)

Make predictions using selected features.

**Parameters** **X** (*pd.DataFrame*, or *np.ndarray*) – Data of shape [n\_samples, n\_features]

**Returns** Predicted values

**Return type** pd.Series

**predict\_proba** (*self*, *X*)

Make probability estimates for labels.

**Parameters** **X** (*pd.DataFrame*, or *np.ndarray*) – Features

**Returns** Probability estimates

**Return type** pd.Series

**save** (*self*, *file\_path*, *pickle\_protocol*=*cloudpickle.DEFAULT\_PROTOCOL*)

Saves component at file path

#### Parameters

- **file\_path** (*str*) – Location to save file
- **pickle\_protocol** (*int*) – The pickle data stream format.

**Returns** None

## elasticnet\_classifier

### Module Contents

### Classes Summary

---

*ElasticNetClassifier*

Elastic Net Classifier. Uses Logistic Regression with elasticnet penalty as the base estimator.

---

### Contents

**class** evalml.pipelines.components.estimators.classifiers.elasticnet\_classifier.**ElasticNetC**

Elastic Net Classifier. Uses Logistic Regression with elasticnet penalty as the base estimator.

#### Parameters

- **penalty** (*{ "l1", "l2", "elasticnet", "none" }*) – The norm used in penalization. Defaults to “elasticnet”.
- **C** (*float*) – Inverse of regularization strength. Must be a positive float. Defaults to 1.0.
- **l1\_ratio** (*float*) – The mixing parameter, with  $0 \leq \text{l1\_ratio} \leq 1$ . Only used if `penalty='elasticnet'`. Setting `l1_ratio=0` is equivalent to using `penalty='l2'`, while setting `l1_ratio=1` is equivalent to using `penalty='l1'`. For  $0 < \text{l1\_ratio} < 1$ , the penalty is a combination of L1 and L2. Defaults to 0.15.
- **multi\_class** (*{ "auto", "ovr", "multinomial" }*) – If the option chosen is “ovr”, then a binary problem is fit for each label. For “multinomial” the loss minimised is the multinomial loss fit across the entire probability distribution, even when the data is binary. “multinomial” is unavailable when `solver='liblinear'`. “auto” selects “ovr” if the data is binary, or if `solver='liblinear'`, and otherwise selects “multinomial”. Defaults to “auto”.
- **solver** (*{ "newton-cg", "lbfgs", "liblinear", "sag", "saga" }*) – Algorithm to use in the optimization problem. For small datasets, “liblinear” is a good

choice, whereas “sag” and “saga” are faster for large ones. For multiclass problems, only “newton-cg”, “sag”, “saga” and “lbfgs” handle multinomial loss; “liblinear” is limited to one-versus-rest schemes.

- “newton-cg”, “lbfgs”, “sag” and “saga” handle L2 or no penalty
- “liblinear” and “saga” also handle L1 penalty
- “saga” also supports “elasticnet” penalty
- “liblinear” does not support setting `penalty='none'`

Defaults to “saga”.

- **n\_jobs** (*int*) – Number of parallel threads used to run xgboost. Note that creating thread contention will significantly slow down the algorithm. Defaults to -1.
- **random\_seed** (*int*) – Seed for the random number generator. Defaults to 0.

#### Attributes

<b>hyper-parameter_ranges</b>	{ “C”: Real(0.01, 10), “l1_ratio”: Real(0, 1)}
<b>model_family</b>	ModelFamily.LINEAR_MODEL
<b>modifies_features</b>	True
<b>modifies_target</b>	False
<b>name</b>	Elastic Net Classifier
<b>predict_uses_y</b>	False
<b>supported_problem_types</b>	[ ProblemTypes.BINARY, ProblemTypes.MULTICLASS, ProblemTypes.TIME_SERIES_BINARY, ProblemTypes.TIME_SERIES_MULTICLASS,]

#### Methods

<i>clone</i>	Constructs a new component with the same parameters and random state.
<i>default_parameters</i>	Returns the default parameters for this component.
<i>describe</i>	Describe a component and its parameters
<i>feature_importance</i>	Returns importance associated with each feature.
<i>fit</i>	Fits component to data
<i>load</i>	Loads component at file path
<i>needs_fitting</i>	Returns boolean determining if component needs fitting before
<i>parameters</i>	Returns the parameters which were used to initialize the component
<i>predict</i>	Make predictions using selected features.
<i>predict_proba</i>	Make probability estimates for labels.
<i>save</i>	Saves component at file path

#### **clone** (*self*)

Constructs a new component with the same parameters and random state.

**Returns** A new instance of this component with identical parameters and random state.

**default\_parameters** (*cls*)

Returns the default parameters for this component.

Our convention is that `Component.default_parameters == Component().parameters`.

**Returns** default parameters for this component.

**Return type** dict

**describe** (*self*, *print\_name=False*, *return\_dict=False*)

Describe a component and its parameters

**Parameters**

- **print\_name** (*bool*, *optional*) – whether to print name of component
- **return\_dict** (*bool*, *optional*) – whether to return description as dictionary in the format {"name": name, "parameters": parameters}

**Returns** prints and returns dictionary

**Return type** None or dict

**property feature\_importance** (*self*)

Returns importance associated with each feature.

**Returns** Importance associated with each feature

**Return type** np.ndarray

**fit** (*self*, *X*, *y*)

Fits component to data

**Parameters**

- **X** (*list*, *pd.DataFrame* or *np.ndarray*) – The input training data of shape [n\_samples, n\_features]
- **y** (*list*, *pd.Series*, *np.ndarray*, *optional*) – The target training data of length [n\_samples]

**Returns** self

**static load** (*file\_path*)

Loads component at file path

**Parameters** **file\_path** (*str*) – Location to load file

**Returns** ComponentBase object

**needs\_fitting** (*self*)

Returns boolean determining if component needs fitting before calling predict, predict\_proba, transform, or feature\_importances. This can be overridden to False for components that do not need to be fit or whose fit methods do nothing.

**property parameters** (*self*)

Returns the parameters which were used to initialize the component

**predict** (*self*, *X*)

Make predictions using selected features.

**Parameters** **X** (*pd.DataFrame*, *np.ndarray*) – Data of shape [n\_samples, n\_features]

**Returns** Predicted values

**Return type** pd.Series

**predict\_proba** (*self*, *X*)

Make probability estimates for labels.

**Parameters** **X** (*pd.DataFrame*, or *np.ndarray*) – Features

**Returns** Probability estimates

**Return type** *pd.Series*

**save** (*self*, *file\_path*, *pickle\_protocol=cloudpickle.DEFAULT\_PROTOCOL*)

Saves component at file path

**Parameters**

- **file\_path** (*str*) – Location to save file
- **pickle\_protocol** (*int*) – The pickle data stream format.

**Returns** None

**et\_classifier**

## Module Contents

## Classes Summary

---

*ExtraTreesClassifier*

Extra Trees Classifier.

---

## Contents

**class** evalml.pipelines.components.estimators.classifiers.et\_classifier.**ExtraTreesClassifier**

Extra Trees Classifier.

**Parameters**

- **n\_estimators** (*float*) – The number of trees in the forest. Defaults to 100.
- **max\_features** (*int*, *float* or {"auto", "sqrt", "log2"}) – The number of features to consider when looking for the best split:
  - If int, then consider max\_features features at each split.
  - If float, then max\_features is a fraction and int(max\_features \* n\_features) features are considered at each split.
  - If “auto”, then max\_features=sqrt(n\_features).
  - If “sqrt”, then max\_features=sqrt(n\_features).

- If “log2”, then `max_features=log2(n_features)`.
- If `None`, then `max_features = n_features`.

The search for a split does not stop until at least one valid partition of the node samples is found, even if it requires to effectively inspect more than `max_features` features. Defaults to “auto”.

- **`max_depth`** (*int*) – The maximum depth of the tree. Defaults to 6.
- **`min_samples_split`** (*int or float*) – The minimum number of samples required to split an internal node:
  - If *int*, then consider `min_samples_split` as the minimum number.
  - If *float*, then `min_samples_split` is a fraction and `ceil(min_samples_split * n_samples)` are the minimum number of samples for each split.
- **`to 2.`** (*Defaults*) –
- **`min_weight_fraction_leaf`** (*float*) – The minimum weighted fraction of the sum total of weights (of all the input samples) required to be at a leaf node. Defaults to 0.0.
- **`n_jobs`** (*int or None*) – Number of jobs to run in parallel. -1 uses all processes. Defaults to -1.
- **`random_seed`** (*int*) – Seed for the random number generator. Defaults to 0.

#### Attributes

<b>hyper-parameter_ranges</b>	{ “n_estimators”: Integer(10, 1000), “max_features”: [“auto”, “sqrt”, “log2”], “max_depth”: Integer(4, 10), }
<b>model_family</b>	ModelFamily.EXTRA_TREES
<b>modifies_features</b>	True
<b>modifies_target</b>	False
<b>name</b>	Extra Trees Classifier
<b>predict_uses_y</b>	False
<b>supported_problem_types</b>	[ ProblemTypes.BINARY, ProblemTypes.MULTICLASS, ProblemTypes.TIME_SERIES_BINARY, ProblemTypes.TIME_SERIES_MULTICLASS, ]

#### Methods

<code>clone</code>	Constructs a new component with the same parameters and random state.
<code>default_parameters</code>	Returns the default parameters for this component.
<code>describe</code>	Describe a component and its parameters
<code>feature_importance</code>	Returns importance associated with each feature.
<code>fit</code>	Fits component to data
<code>load</code>	Loads component at file path
<code>needs_fitting</code>	Returns boolean determining if component needs fitting before
<code>parameters</code>	Returns the parameters which were used to initialize the component

continues on next page



Table 309 – continued from previous page

<code>predict</code>	Make predictions using selected features.
<code>predict_proba</code>	Make probability estimates for labels.
<code>save</code>	Saves component at file path

**clone** (*self*)

Constructs a new component with the same parameters and random state.

**Returns** A new instance of this component with identical parameters and random state.

**default\_parameters** (*cls*)

Returns the default parameters for this component.

Our convention is that `Component.default_parameters == Component().parameters`.

**Returns** default parameters for this component.

**Return type** dict

**describe** (*self*, *print\_name=False*, *return\_dict=False*)

Describe a component and its parameters

**Parameters**

- **print\_name** (*bool*, *optional*) – whether to print name of component
- **return\_dict** (*bool*, *optional*) – whether to return description as dictionary in the format {"name": name, "parameters": parameters}

**Returns** prints and returns dictionary

**Return type** None or dict

**property feature\_importance** (*self*)

Returns importance associated with each feature.

**Returns** Importance associated with each feature

**Return type** np.ndarray

**fit** (*self*, *X*, *y=None*)

Fits component to data

**Parameters**

- **X** (*list*, *pd.DataFrame* or *np.ndarray*) – The input training data of shape [n\_samples, n\_features]
- **y** (*list*, *pd.Series*, *np.ndarray*, *optional*) – The target training data of length [n\_samples]

**Returns** self

**static load** (*file\_path*)

Loads component at file path

**Parameters** **file\_path** (*str*) – Location to load file

**Returns** ComponentBase object

**needs\_fitting** (*self*)

Returns boolean determining if component needs fitting before calling predict, predict\_proba, transform, or feature\_importances. This can be overridden to False for components that do not need to be fit or whose fit methods do nothing.

**property parameters** (*self*)

Returns the parameters which were used to initialize the component

**predict** (*self*, *X*)

Make predictions using selected features.

**Parameters** *X* (*pd.DataFrame*, *np.ndarray*) – Data of shape [n\_samples, n\_features]

**Returns** Predicted values

**Return type** *pd.Series*

**predict\_proba** (*self*, *X*)

Make probability estimates for labels.

**Parameters** *X* (*pd.DataFrame*, *or np.ndarray*) – Features

**Returns** Probability estimates

**Return type** *pd.Series*

**save** (*self*, *file\_path*, *pickle\_protocol=cloudpickle.DEFAULT\_PROTOCOL*)

Saves component at file path

**Parameters**

- **file\_path** (*str*) – Location to save file
- **pickle\_protocol** (*int*) – The pickle data stream format.

**Returns** None

## kneighbors\_classifier

### Module Contents

### Classes Summary

---

*KNeighborsClassifier*

K-Nearest Neighbors Classifier.

---

### Contents

**class** evalml.pipelines.components.estimators.classifiers.kneighbors\_classifier.**KNeighborsC**

K-Nearest Neighbors Classifier.

**Parameters**

- **n\_neighbors** (*int*) – Number of neighbors to use by default. Defaults to 5.

- **weights** ({'uniform', 'distance'} or callable) – Weight function used in prediction. Can be:
  - 'uniform' : uniform weights. All points in each neighborhood are weighted equally.
  - 'distance' : weight points by the inverse of their distance. in this case, closer neighbors of a query point will have a greater influence than neighbors which are further away.
  - [callable] : a user-defined function which accepts an array of distances, and returns an array of the same shape containing the weights.
 Defaults to "uniform".
- **algorithm** ({'auto', 'ball\_tree', 'kd\_tree', 'brute'}) – Algorithm used to compute the nearest neighbors:
  - 'ball\_tree' will use BallTree
  - 'kd\_tree' will use KDTree
  - 'brute' will use a brute-force search.
 'auto' will attempt to decide the most appropriate algorithm based on the values passed to fit method. Defaults to "auto". Note: fitting on sparse input will override the setting of this parameter, using brute force.
- **leaf\_size** (int) – Leaf size passed to BallTree or KDTree. This can affect the speed of the construction and query, as well as the memory required to store the tree. The optimal value depends on the nature of the problem. Defaults to 30.
- **p** (int) – Power parameter for the Minkowski metric. When  $p = 1$ , this is equivalent to using `manhattan_distance` (l1), and `euclidean_distance` (l2) for  $p = 2$ . For arbitrary  $p$ , `minkowski_distance (l_p)` is used. Defaults to 2.
- **random\_seed** (int) – Seed for the random number generator. Defaults to 0.

#### Attributes

<b>hyper-parameter_ranges</b>	{ "n_neighbors": Integer(2, 12), "weights": ["uniform", "distance"], "algorithm": ["auto", "ball_tree", "kd_tree", "brute"], "leaf_size": Integer(10, 30), "p": Integer(1, 5), }
<b>model_family</b>	ModelFamily.K_NEIGHBORS
<b>modifies_features</b>	True
<b>modifies_target</b>	False
<b>name</b>	KNN Classifier
<b>predict_uses_y</b>	False
<b>supported_problem_types</b>	[ ProblemTypes.BINARY, ProblemTypes.MULTICLASS, ProblemTypes.TIME_SERIES_BINARY, ProblemTypes.TIME_SERIES_MULTICLASS, ]

#### Methods

<code>clone</code>	Constructs a new component with the same parameters and random state.
<code>default_parameters</code>	Returns the default parameters for this component.
<code>describe</code>	Describe a component and its parameters

continues on next page

Table 311 – continued from previous page

<code>feature_importance</code>	Returns array of 0's matching the input number of features as <code>feature_importance</code> is
<code>fit</code>	Fits component to data
<code>load</code>	Loads component at file path
<code>needs_fitting</code>	Returns boolean determining if component needs fitting before
<code>parameters</code>	Returns the parameters which were used to initialize the component
<code>predict</code>	Make predictions using selected features.
<code>predict_proba</code>	Make probability estimates for labels.
<code>save</code>	Saves component at file path

**clone** (*self*)

Constructs a new component with the same parameters and random state.

**Returns** A new instance of this component with identical parameters and random state.

**default\_parameters** (*cls*)

Returns the default parameters for this component.

Our convention is that `Component.default_parameters == Component().parameters`.

**Returns** default parameters for this component.

**Return type** dict

**describe** (*self*, *print\_name=False*, *return\_dict=False*)

Describe a component and its parameters

**Parameters**

- **print\_name** (*bool*, *optional*) – whether to print name of component
- **return\_dict** (*bool*, *optional*) – whether to return description as dictionary in the format {"name": name, "parameters": parameters}

**Returns** prints and returns dictionary

**Return type** None or dict

**property feature\_importance** (*self*)

Returns array of 0's matching the input number of features as `feature_importance` is not defined for KNN classifiers.

**fit** (*self*, *X*, *y=None*)

Fits component to data

**Parameters**

- **X** (*list*, *pd.DataFrame* or *np.ndarray*) – The input training data of shape [n\_samples, n\_features]
- **y** (*list*, *pd.Series*, *np.ndarray*, *optional*) – The target training data of length [n\_samples]

**Returns** self

**static load** (*file\_path*)

Loads component at file path

**Parameters** **file\_path** (*str*) – Location to load file

**Returns** ComponentBase object

**needs\_fitting** (*self*)

Returns boolean determining if component needs fitting before calling predict, predict\_proba, transform, or feature\_importances. This can be overridden to False for components that do not need to be fit or whose fit methods do nothing.

**property parameters** (*self*)

Returns the parameters which were used to initialize the component

**predict** (*self*, *X*)

Make predictions using selected features.

**Parameters** *X* (*pd.DataFrame*, *np.ndarray*) – Data of shape [n\_samples, n\_features]

**Returns** Predicted values

**Return type** *pd.Series*

**predict\_proba** (*self*, *X*)

Make probability estimates for labels.

**Parameters** *X* (*pd.DataFrame*, or *np.ndarray*) – Features

**Returns** Probability estimates

**Return type** *pd.Series*

**save** (*self*, *file\_path*, *pickle\_protocol=cloudpickle.DEFAULT\_PROTOCOL*)

Saves component at file path

**Parameters**

- **file\_path** (*str*) – Location to save file
- **pickle\_protocol** (*int*) – The pickle data stream format.

**Returns** None

## lightgbm\_classifier

### Module Contents

### Classes Summary

---

*LightGBMClassifier*

LightGBM Classifier.

---

## Contents

`class evalml.pipelines.components.estimators.classifiers.lightgbm_classifier.LightGBMClassifier`

LightGBM Classifier.

### Parameters

- **boosting\_type** (*string*) – Type of boosting to use. Defaults to “gbdt”. - ‘gbdt’ uses traditional Gradient Boosting Decision Tree - “dart”, uses Dropouts meet Multiple Additive Regression Trees - “goss”, uses Gradient-based One-Side Sampling - “rf”, uses Random Forest
- **learning\_rate** (*float*) – Boosting learning rate. Defaults to 0.1.
- **n\_estimators** (*int*) – Number of boosted trees to fit. Defaults to 100.
- **max\_depth** (*int*) – Maximum tree depth for base learners,  $\leq 0$  means no limit. Defaults to 0.
- **num\_leaves** (*int*) – Maximum tree leaves for base learners. Defaults to 31.
- **min\_child\_samples** (*int*) – Minimum number of data needed in a child (leaf). Defaults to 20.
- **bagging\_fraction** (*float*) – LightGBM will randomly select a subset of features on each iteration (tree) without resampling if this is smaller than 1.0. For example, if set to 0.8, LightGBM will select 80% of features before training each tree. This can be used to speed up training and deal with overfitting. Defaults to 0.9.
- **bagging\_freq** (*int*) – Frequency for bagging. 0 means bagging is disabled. k means perform bagging at every k iteration. Every k-th iteration, LightGBM will randomly select  $\text{bagging\_fraction} * 100\%$  of the data to use for the next k iterations. Defaults to 0.
- **n\_jobs** (*int or None*) – Number of threads to run in parallel. -1 uses all threads. Defaults to -1.
- **random\_seed** (*int*) – Seed for the random number generator. Defaults to 0.

### Attributes

<b>hyper-parameter_ranges</b>	{ "learning_rate": Real(0.000001, 1), "boosting_type": ["gbdt", "dart", "goss", "rf"], "n_estimators": Integer(10, 100), "max_depth": Integer(0, 10), "num_leaves": Integer(2, 100), "min_child_samples": Integer(1, 100), "bagging_fraction": Real(0.000001, 1), "bagging_freq": Integer(0, 1),}
<b>model_family</b>	ModelFamily.LIGHTGBM
<b>modifies_features</b>	True
<b>modifies_target</b>	False
<b>name</b>	LightGBM Classifier
<b>pre-dict_uses_y</b>	False
<b>SEED_MAX</b>	SEED_BOUNDS.max_bound
<b>SEED_MIN</b>	0
<b>supported_problem_types</b>	[ ProblemTypes.BINARY, ProblemTypes.MULTICLASS, ProblemTypes.TIME_SERIES_BINARY, ProblemTypes.TIME_SERIES_MULTICLASS,]

## Methods

<i>clone</i>	Constructs a new component with the same parameters and random state.
<i>default_parameters</i>	Returns the default parameters for this component.
<i>describe</i>	Describe a component and its parameters
<i>feature_importance</i>	Returns importance associated with each feature.
<i>fit</i>	Fits component to data
<i>load</i>	Loads component at file path
<i>needs_fitting</i>	Returns boolean determining if component needs fitting before
<i>parameters</i>	Returns the parameters which were used to initialize the component
<i>predict</i>	Make predictions using selected features.
<i>predict_proba</i>	Make probability estimates for labels.
<i>save</i>	Saves component at file path

### **clone** (*self*)

Constructs a new component with the same parameters and random state.

**Returns** A new instance of this component with identical parameters and random state.

### **default\_parameters** (*cls*)

Returns the default parameters for this component.

Our convention is that `Component.default_parameters == Component().parameters`.

**Returns** default parameters for this component.

**Return type** dict

### **describe** (*self*, *print\_name=False*, *return\_dict=False*)

Describe a component and its parameters

#### Parameters

- **print\_name** (*bool*, *optional*) – whether to print name of component

- **return\_dict** (*bool, optional*) – whether to return description as dictionary in the format {"name": name, "parameters": parameters}

**Returns** prints and returns dictionary

**Return type** None or dict

**property feature\_importance** (*self*)

Returns importance associated with each feature.

**Returns** Importance associated with each feature

**Return type** np.ndarray

**fit** (*self, X, y=None*)

Fits component to data

**Parameters**

- **X** (*list, pd.DataFrame or np.ndarray*) – The input training data of shape [n\_samples, n\_features]
- **y** (*list, pd.Series, np.ndarray, optional*) – The target training data of length [n\_samples]

**Returns** self

**static load** (*file\_path*)

Loads component at file path

**Parameters** **file\_path** (*str*) – Location to load file

**Returns** ComponentBase object

**needs\_fitting** (*self*)

Returns boolean determining if component needs fitting before calling predict, predict\_proba, transform, or feature\_importances. This can be overridden to False for components that do not need to be fit or whose fit methods do nothing.

**property parameters** (*self*)

Returns the parameters which were used to initialize the component

**predict** (*self, X*)

Make predictions using selected features.

**Parameters** **X** (*pd.DataFrame, np.ndarray*) – Data of shape [n\_samples, n\_features]

**Returns** Predicted values

**Return type** pd.Series

**predict\_proba** (*self, X*)

Make probability estimates for labels.

**Parameters** **X** (*pd.DataFrame, or np.ndarray*) – Features

**Returns** Probability estimates

**Return type** pd.Series

**save** (*self, file\_path, pickle\_protocol=cloudpickle.DEFAULT\_PROTOCOL*)

Saves component at file path

**Parameters**

- **file\_path** (*str*) – Location to save file



- **pickle\_protocol** (*int*) – The pickle data stream format.

**Returns** None

## logistic\_regression\_classifier

### Module Contents

### Classes Summary

---

*LogisticRegressionClassifier*

Logistic Regression Classifier.

---

### Contents

**class** evalml.pipelines.components.estimators.classifiers.logistic\_regression\_classifier.**Log**

Logistic Regression Classifier.

#### Parameters

- **penalty** ({*"l1"*, *"l2"*, *"elasticnet"*, *"none"*}) – The norm used in penalization. Defaults to *"l2"*.
- **C** (*float*) – Inverse of regularization strength. Must be a positive float. Defaults to 1.0.
- **multi\_class** ({*"auto"*, *"ovr"*, *"multinomial"*}) – If the option chosen is *"ovr"*, then a binary problem is fit for each label. For *"multinomial"* the loss minimised is the multinomial loss fit across the entire probability distribution, even when the data is binary. *"multinomial"* is unavailable when *solver="liblinear"*. *"auto"* selects *"ovr"* if the data is binary, or if *solver="liblinear"*, and otherwise selects *"multinomial"*. Defaults to *"auto"*.
- **solver** ({*"newton-cg"*, *"lbfgs"*, *"liblinear"*, *"sag"*, *"saga"*}) – Algorithm to use in the optimization problem. For small datasets, *"liblinear"* is a good choice, whereas *"sag"* and *"saga"* are faster for large ones. For multiclass problems, only *"newton-cg"*, *"sag"*, *"saga"* and *"lbfgs"* handle multinomial loss; *"liblinear"* is limited to one-versus-rest schemes.
  - *"newton-cg"*, *"lbfgs"*, *"sag"* and *"saga"* handle L2 or no penalty
  - *"liblinear"* and *"saga"* also handle L1 penalty
  - *"saga"* also supports *"elasticnet"* penalty
  - *"liblinear"* does not support setting *penalty='none'*
 Defaults to *"lbfgs"*.

- **n\_jobs** (*int*) – Number of parallel threads used to run xgboost. Note that creating thread contention will significantly slow down the algorithm. Defaults to -1.
- **random\_seed** (*int*) – Seed for the random number generator. Defaults to 0.

#### Attributes

<b>hyper-parameter_ranges</b>	{ "penalty": ["l2"], "C": Real(0.01, 10), }
<b>model_family</b>	ModelFamily.LINEAR_MODEL
<b>modifies_features</b>	True
<b>modifies_target</b>	False
<b>name</b>	Logistic Regression Classifier
<b>predict_uses_y</b>	False
<b>supported_problem_types</b>	[ ProblemTypes.BINARY, ProblemTypes.MULTICLASS, ProblemTypes.TIME_SERIES_BINARY, ProblemTypes.TIME_SERIES_MULTICLASS, ]

#### Methods

<i>clone</i>	Constructs a new component with the same parameters and random state.
<i>default_parameters</i>	Returns the default parameters for this component.
<i>describe</i>	Describe a component and its parameters
<i>feature_importance</i>	Returns importance associated with each feature.
<i>fit</i>	Fits component to data
<i>load</i>	Loads component at file path
<i>needs_fitting</i>	Returns boolean determining if component needs fitting before
<i>parameters</i>	Returns the parameters which were used to initialize the component
<i>predict</i>	Make predictions using selected features.
<i>predict_proba</i>	Make probability estimates for labels.
<i>save</i>	Saves component at file path

#### **clone** (*self*)

Constructs a new component with the same parameters and random state.

**Returns** A new instance of this component with identical parameters and random state.

#### **default\_parameters** (*cls*)

Returns the default parameters for this component.

Our convention is that `Component.default_parameters == Component().parameters`.

**Returns** default parameters for this component.

**Return type** dict

#### **describe** (*self*, *print\_name=False*, *return\_dict=False*)

Describe a component and its parameters

**Parameters**

- **print\_name** (*bool, optional*) – whether to print name of component
- **return\_dict** (*bool, optional*) – whether to return description as dictionary in the format {"name": name, "parameters": parameters}

**Returns** prints and returns dictionary

**Return type** None or dict

**property feature\_importance** (*self*)

Returns importance associated with each feature.

**Returns** Importance associated with each feature

**Return type** np.ndarray

**fit** (*self, X, y=None*)

Fits component to data

**Parameters**

- **X** (*list, pd.DataFrame or np.ndarray*) – The input training data of shape [n\_samples, n\_features]
- **y** (*list, pd.Series, np.ndarray, optional*) – The target training data of length [n\_samples]

**Returns** self

**static load** (*file\_path*)

Loads component at file path

**Parameters** **file\_path** (*str*) – Location to load file

**Returns** ComponentBase object

**needs\_fitting** (*self*)

Returns boolean determining if component needs fitting before calling predict, predict\_proba, transform, or feature\_importances. This can be overridden to False for components that do not need to be fit or whose fit methods do nothing.

**property parameters** (*self*)

Returns the parameters which were used to initialize the component

**predict** (*self, X*)

Make predictions using selected features.

**Parameters** **X** (*pd.DataFrame, np.ndarray*) – Data of shape [n\_samples, n\_features]

**Returns** Predicted values

**Return type** pd.Series

**predict\_proba** (*self, X*)

Make probability estimates for labels.

**Parameters** **X** (*pd.DataFrame, or np.ndarray*) – Features

**Returns** Probability estimates

**Return type** pd.Series

**save** (*self, file\_path, pickle\_protocol=cloudpickle.DEFAULT\_PROTOCOL*)

Saves component at file path

**Parameters**

- **file\_path** (*str*) – Location to save file
- **pickle\_protocol** (*int*) – The pickle data stream format.

**Returns** None

**rf\_classifier**

## Module Contents

## Classes Summary

---

<i>RandomForestClassifier</i>
-------------------------------

---

Random Forest Classifier.
---------------------------

## Contents

**class** evalml.pipelines.components.estimators.classifiers.rf\_classifier.**RandomForestClassifier**

Random Forest Classifier.

### Parameters

- **n\_estimators** (*float*) – The number of trees in the forest. Defaults to 100.
- **max\_depth** (*int*) – Maximum tree depth for base learners. Defaults to 6.
- **n\_jobs** (*int or None*) – Number of jobs to run in parallel. -1 uses all processes. Defaults to -1.
- **random\_seed** (*int*) – Seed for the random number generator. Defaults to 0.

### Attributes

<b>hyper-parameter_ranges</b>	{ "n_estimators": Integer(10, 1000), "max_depth": Integer(1, 10), }
<b>model_family</b>	ModelFamily.RANDOM_FOREST
<b>modifies_features</b>	True
<b>modifies_target</b>	False
<b>name</b>	Random Forest Classifier
<b>predict Uses y</b>	False
<b>supported problem types</b>	[ ProblemTypes.BINARY, ProblemTypes.MULTICLASS, ProblemTypes.TIME_SERIES_BINARY, ProblemTypes.TIME_SERIES_MULTICLASS, ]

### Methods

<code>clone</code>	Constructs a new component with the same parameters and random state.
<code>default_parameters</code>	Returns the default parameters for this component.
<code>describe</code>	Describe a component and its parameters
<code>feature_importance</code>	Returns importance associated with each feature.
<code>fit</code>	Fits component to data
<code>load</code>	Loads component at file path
<code>needs_fitting</code>	Returns boolean determining if component needs fitting before
<code>parameters</code>	Returns the parameters which were used to initialize the component
<code>predict</code>	Make predictions using selected features.
<code>predict_proba</code>	Make probability estimates for labels.
<code>save</code>	Saves component at file path

**clone** (*self*)

Constructs a new component with the same parameters and random state.

**Returns** A new instance of this component with identical parameters and random state.

**default\_parameters** (*cls*)

Returns the default parameters for this component.

Our convention is that `Component.default_parameters == Component().parameters`.

**Returns** default parameters for this component.

**Return type** dict

**describe** (*self*, *print\_name=False*, *return\_dict=False*)

Describe a component and its parameters

**Parameters**

- **print\_name** (*bool*, *optional*) – whether to print name of component
- **return\_dict** (*bool*, *optional*) – whether to return description as dictionary in the format {"name": name, "parameters": parameters}

**Returns** prints and returns dictionary

**Return type** None or dict

**property feature\_importance** (*self*)

Returns importance associated with each feature.

**Returns** Importance associated with each feature

**Return type** np.ndarray

**fit** (*self*, *X*, *y=None*)

Fits component to data

**Parameters**

- **X** (*list*, *pd.DataFrame* or *np.ndarray*) – The input training data of shape [n\_samples, n\_features]
- **y** (*list*, *pd.Series*, *np.ndarray*, *optional*) – The target training data of length [n\_samples]

**Returns** self

**static load** (*file\_path*)

Loads component at file path

**Parameters** *file\_path* (*str*) – Location to load file

**Returns** ComponentBase object

**needs\_fitting** (*self*)

Returns boolean determining if component needs fitting before calling predict, predict\_proba, transform, or feature\_importances. This can be overridden to False for components that do not need to be fit or whose fit methods do nothing.

**property parameters** (*self*)

Returns the parameters which were used to initialize the component

**predict** (*self*, *X*)

Make predictions using selected features.

**Parameters** *X* (*pd.DataFrame*, *np.ndarray*) – Data of shape [n\_samples, n\_features]

**Returns** Predicted values

**Return type** *pd.Series*

**predict\_proba** (*self*, *X*)

Make probability estimates for labels.

**Parameters** *X* (*pd.DataFrame*, *or np.ndarray*) – Features

**Returns** Probability estimates

**Return type** *pd.Series*

**save** (*self*, *file\_path*, *pickle\_protocol=cloudpickle.DEFAULT\_PROTOCOL*)

Saves component at file path

**Parameters**

- **file\_path** (*str*) – Location to save file
- **pickle\_protocol** (*int*) – The pickle data stream format.

**Returns** None

**svm\_classifier**

**Module Contents**

**Classes Summary**

---

*SVMClassifier*

Support Vector Machine Classifier.

---

## Contents

**class** evalml.pipelines.components.estimators.classifiers.svm\_classifier.**SVMClassifier** (*C=1.0*

*ker-*  
*nel='r*  
*gamma*  
*prob-*  
*a-*  
*bil-*  
*ity=Tr*  
*ran-*  
*dom\_s*  
*\*\*kwa*

Support Vector Machine Classifier.

### Parameters

- **C** (*float*) – The regularization parameter. The strength of the regularization is inversely proportional to C. Must be strictly positive. The penalty is a squared l2 penalty. Defaults to 1.0.
- **kernel** (*{ "linear", "poly", "rbf", "sigmoid", "precomputed" }*) – Specifies the kernel type to be used in the algorithm. Defaults to “rbf”.
- **gamma** (*{ "scale", "auto" } or float*) – Kernel coefficient for “rbf”, “poly” and “sigmoid”. Defaults to “scale”. - If gamma=’scale’ (default) is passed then it uses  $1 / (n\_features * X.var())$  as value of gamma - if “auto”, uses  $1 / n\_features$
- **probability** (*boolean*) – Whether to enable probability estimates. Defaults to True.
- **random\_seed** (*int*) – Seed for the random number generator. Defaults to 0.

### Attributes

<b>hyper-parameter_ranges</b>	{ “C”: Real(0, 10), “kernel”: [“linear”, “poly”, “rbf”, “sigmoid”, “precomputed”], “gamma”: [“scale”, “auto”], }
<b>model_family</b>	ModelFamily.SVM
<b>modifies_features</b>	True
<b>modifies_target</b>	False
<b>name</b>	SVM Classifier
<b>pre-dict_uses_y</b>	False
<b>supported_problem_types</b>	[ ProblemTypes.BINARY, ProblemTypes.MULTICLASS, ProblemTypes.TIME_SERIES_BINARY, ProblemTypes.TIME_SERIES_MULTICLASS, ]

### Methods

<i>clone</i>	Constructs a new component with the same parameters and random state.
<i>default_parameters</i>	Returns the default parameters for this component.
<i>describe</i>	Describe a component and its parameters
<i>feature_importance</i>	Feature importance only works with linear kernels.

continues on next page

Table 319 – continued from previous page

<i>fit</i>	Fits component to data
<i>load</i>	Loads component at file path
<i>needs_fitting</i>	Returns boolean determining if component needs fitting before
<i>parameters</i>	Returns the parameters which were used to initialize the component
<i>predict</i>	Make predictions using selected features.
<i>predict_proba</i>	Make probability estimates for labels.
<i>save</i>	Saves component at file path

**clone** (*self*)

Constructs a new component with the same parameters and random state.

**Returns** A new instance of this component with identical parameters and random state.

**default\_parameters** (*cls*)

Returns the default parameters for this component.

Our convention is that `Component.default_parameters == Component().parameters`.

**Returns** default parameters for this component.

**Return type** dict

**describe** (*self*, *print\_name=False*, *return\_dict=False*)

Describe a component and its parameters

**Parameters**

- **print\_name** (*bool*, *optional*) – whether to print name of component
- **return\_dict** (*bool*, *optional*) – whether to return description as dictionary in the format {"name": name, "parameters": parameters}

**Returns** prints and returns dictionary

**Return type** None or dict

**property feature\_importance** (*self*)

Feature importance only works with linear kernels. If the kernel isn't linear, we return a numpy array of zeros

**fit** (*self*, *X*, *y=None*)

Fits component to data

**Parameters**

- **X** (*list*, *pd.DataFrame* or *np.ndarray*) – The input training data of shape [n\_samples, n\_features]
- **y** (*list*, *pd.Series*, *np.ndarray*, *optional*) – The target training data of length [n\_samples]

**Returns** self

**static load** (*file\_path*)

Loads component at file path

**Parameters** **file\_path** (*str*) – Location to load file

**Returns** ComponentBase object



**needs\_fitting** (*self*)

Returns boolean determining if component needs fitting before calling predict, predict\_proba, transform, or feature\_importances. This can be overridden to False for components that do not need to be fit or whose fit methods do nothing.

**property parameters** (*self*)

Returns the parameters which were used to initialize the component

**predict** (*self*, *X*)

Make predictions using selected features.

**Parameters** *X* (*pd.DataFrame*, *np.ndarray*) – Data of shape [n\_samples, n\_features]

**Returns** Predicted values

**Return type** *pd.Series*

**predict\_proba** (*self*, *X*)

Make probability estimates for labels.

**Parameters** *X* (*pd.DataFrame*, or *np.ndarray*) – Features

**Returns** Probability estimates

**Return type** *pd.Series*

**save** (*self*, *file\_path*, *pickle\_protocol=cloudpickle.DEFAULT\_PROTOCOL*)

Saves component at file path

**Parameters**

- **file\_path** (*str*) – Location to save file
- **pickle\_protocol** (*int*) – The pickle data stream format.

**Returns** None

**xgboost\_classifier****Module Contents****Classes Summary**


---

*XGBoostClassifier*


---

XGBoost Classifier.

---

Contents

`class evalml.pipelines.components.estimators.classifiers.xgboost_classifier.XGBoostClassifier`

XGBoost Classifier.

Parameters

- `eta` (*float*) – Boosting learning rate. Defaults to 0.1.
- `max_depth` (*int*) – Maximum tree depth for base learners. Defaults to 6.
- `min_child_weight` (*float*) – Minimum sum of instance weight (hessian) needed in a child. Defaults to 1.0
- `n_estimators` (*int*) – Number of gradient boosted trees. Equivalent to number of boosting rounds. Defaults to 100.
- `random_seed` (*int*) – Seed for the random number generator. Defaults to 0.
- `n_jobs` (*int*) – Number of parallel threads used to run xgboost. Note that creating thread contention will significantly slow down the algorithm. Defaults to -1.

Attributes

hyper-parameter_ranges	{ "eta": Real(0.000001, 1), "max_depth": Integer(1, 10), "min_child_weight": Real(1, 10), "n_estimators": Integer(1, 1000), }
model_family	ModelFamily.XGBOOST
modifies_features	True
modifies_target	False
name	XGBoost Classifier
pre-dict_uses_y	False
SEED_MAX	None
SEED_MIN	None
supported_problem_types	[ ProblemTypes.BINARY, ProblemTypes.MULTICLASS, ProblemTypes.TIME_SERIES_BINARY, ProblemTypes.TIME_SERIES_MULTICLASS, ]

Methods

<code>clone</code>	Constructs a new component with the same parameters and random state.
<code>default_parameters</code>	Returns the default parameters for this component.
<code>describe</code>	Describe a component and its parameters
<code>feature_importance</code>	Returns importance associated with each feature.

continues on next page

Table 321 – continued from previous page

<i>fit</i>	Fits component to data
<i>load</i>	Loads component at file path
<i>needs_fitting</i>	Returns boolean determining if component needs fitting before
<i>parameters</i>	Returns the parameters which were used to initialize the component
<i>predict</i>	Make predictions using selected features.
<i>predict_proba</i>	Make probability estimates for labels.
<i>save</i>	Saves component at file path

**clone** (*self*)

Constructs a new component with the same parameters and random state.

**Returns** A new instance of this component with identical parameters and random state.

**default\_parameters** (*cls*)

Returns the default parameters for this component.

Our convention is that `Component.default_parameters == Component().parameters`.

**Returns** default parameters for this component.

**Return type** dict

**describe** (*self*, *print\_name=False*, *return\_dict=False*)

Describe a component and its parameters

**Parameters**

- **print\_name** (*bool*, *optional*) – whether to print name of component
- **return\_dict** (*bool*, *optional*) – whether to return description as dictionary in the format {"name": name, "parameters": parameters}

**Returns** prints and returns dictionary

**Return type** None or dict

**property feature\_importance** (*self*)

Returns importance associated with each feature.

**Returns** Importance associated with each feature

**Return type** np.ndarray

**fit** (*self*, *X*, *y=None*)

Fits component to data

**Parameters**

- **X** (*list*, *pd.DataFrame* or *np.ndarray*) – The input training data of shape [n\_samples, n\_features]
- **y** (*list*, *pd.Series*, *np.ndarray*, *optional*) – The target training data of length [n\_samples]

**Returns** self

**static load** (*file\_path*)

Loads component at file path

**Parameters** **file\_path** (*str*) – Location to load file

**Returns** ComponentBase object

**needs\_fitting** (*self*)

Returns boolean determining if component needs fitting before calling predict, predict\_proba, transform, or feature\_importances. This can be overridden to False for components that do not need to be fit or whose fit methods do nothing.

**property parameters** (*self*)

Returns the parameters which were used to initialize the component

**predict** (*self*, *X*)

Make predictions using selected features.

**Parameters** *X* (*pd.DataFrame*, *np.ndarray*) – Data of shape [n\_samples, n\_features]

**Returns** Predicted values

**Return type** *pd.Series*

**predict\_proba** (*self*, *X*)

Make probability estimates for labels.

**Parameters** *X* (*pd.DataFrame*, or *np.ndarray*) – Features

**Returns** Probability estimates

**Return type** *pd.Series*

**save** (*self*, *file\_path*, *pickle\_protocol=cloudpickle.DEFAULT\_PROTOCOL*)

Saves component at file path

**Parameters**

- **file\_path** (*str*) – Location to save file
- **pickle\_protocol** (*int*) – The pickle data stream format.

**Returns** None

## Package Contents

### Classes Summary

<i>BaselineClassifier</i>	Classifier that predicts using the specified strategy.
<i>CatBoostClassifier</i>	CatBoost Classifier, a classifier that uses gradient-boosting on decision trees.
<i>DecisionTreeClassifier</i>	Decision Tree Classifier.
<i>ElasticNetClassifier</i>	Elastic Net Classifier. Uses Logistic Regression with elasticnet penalty as the base estimator.
<i>ExtraTreesClassifier</i>	Extra Trees Classifier.
<i>KNeighborsClassifier</i>	K-Nearest Neighbors Classifier.
<i>LightGBMClassifier</i>	LightGBM Classifier.
<i>LogisticRegressionClassifier</i>	Logistic Regression Classifier.
<i>RandomForestClassifier</i>	Random Forest Classifier.
<i>SVMClassifier</i>	Support Vector Machine Classifier.
<i>XGBoostClassifier</i>	XGBoost Classifier.

## Contents

**class** evalml.pipelines.components.estimators.classifiers.**BaselineClassifier** (*strategy='mode', random\_seed=0, \*\*kwargs*)

Classifier that predicts using the specified strategy.

This is useful as a simple baseline classifier to compare with other classifiers.

### Parameters

- **strategy** (*str*) – Method used to predict. Valid options are “mode”, “random” and “random\_weighted”. Defaults to “mode”.
- **random\_seed** (*int*) – Seed for the random number generator. Defaults to 0.

### Attributes

<b>hyper-parameter_ranges</b>	{}
<b>model_family</b>	ModelFamily.BASELINE
<b>modifies_features</b>	True
<b>modifies_target</b>	False
<b>name</b>	Baseline Classifier
<b>pre-dict_uses_y</b>	False
<b>supported_problem_types</b>	[ProblemTypes.BINARY, ProblemTypes.MULTICLASS]

### Methods

<i>classes_</i>	Returns class labels. Will return None before fitting.
<i>clone</i>	Constructs a new component with the same parameters and random state.
<i>default_parameters</i>	Returns the default parameters for this component.
<i>describe</i>	Describe a component and its parameters
<i>feature_importance</i>	Returns importance associated with each feature. Since baseline classifiers do not use input features to calculate predictions, returns an array of zeroes.
<i>fit</i>	Fits component to data
<i>load</i>	Loads component at file path
<i>needs_fitting</i>	Returns boolean determining if component needs fitting before
<i>parameters</i>	Returns the parameters which were used to initialize the component
<i>predict</i>	Make predictions using selected features.
<i>predict_proba</i>	Make probability estimates for labels.
<i>save</i>	Saves component at file path

**property** *classes\_* (*self*)

Returns class labels. Will return None before fitting.

**Returns** Class names

**Return type** list[str] or list(float)

**clone** (*self*)

Constructs a new component with the same parameters and random state.

**Returns** A new instance of this component with identical parameters and random state.

**default\_parameters** (*cls*)

Returns the default parameters for this component.

Our convention is that `Component.default_parameters == Component().parameters`.

**Returns** default parameters for this component.

**Return type** dict

**describe** (*self*, *print\_name=False*, *return\_dict=False*)

Describe a component and its parameters

**Parameters**

- **print\_name** (*bool*, *optional*) – whether to print name of component
- **return\_dict** (*bool*, *optional*) – whether to return description as dictionary in the format {"name": name, "parameters": parameters}

**Returns** prints and returns dictionary

**Return type** None or dict

**property feature\_importance** (*self*)

Returns importance associated with each feature. Since baseline classifiers do not use input features to calculate predictions, returns an array of zeroes.

**Returns** An array of zeroes

**Return type** np.ndarray (float)

**fit** (*self*, *X*, *y=None*)

Fits component to data

**Parameters**

- **X** (*list*, *pd.DataFrame* or *np.ndarray*) – The input training data of shape [n\_samples, n\_features]
- **y** (*list*, *pd.Series*, *np.ndarray*, *optional*) – The target training data of length [n\_samples]

**Returns** self

**static load** (*file\_path*)

Loads component at file path

**Parameters** **file\_path** (*str*) – Location to load file

**Returns** ComponentBase object

**needs\_fitting** (*self*)

Returns boolean determining if component needs fitting before calling predict, predict\_proba, transform, or feature\_importances. This can be overridden to False for components that do not need to be fit or whose fit methods do nothing.

**property parameters** (*self*)

Returns the parameters which were used to initialize the component

**predict** (*self*, *X*)

Make predictions using selected features.

**Parameters** *X* (*pd.DataFrame*, *np.ndarray*) – Data of shape [n\_samples, n\_features]

**Returns** Predicted values

**Return type** *pd.Series*

**predict\_proba** (*self*, *X*)

Make probability estimates for labels.

**Parameters** *X* (*pd.DataFrame*, *or np.ndarray*) – Features

**Returns** Probability estimates

**Return type** *pd.Series*

**save** (*self*, *file\_path*, *pickle\_protocol=cloudpickle.DEFAULT\_PROTOCOL*)

Saves component at file path

**Parameters**

- **file\_path** (*str*) – Location to save file
- **pickle\_protocol** (*int*) – The pickle data stream format.

**Returns** None

```
class evalml.pipelines.components.estimators.classifiers.CatBoostClassifier (n_estimators=10,
                                                                    eta=0.03,
                                                                    max_depth=6,
                                                                    boot-
                                                                    strap_type=None,
                                                                    silent=True,
                                                                    al-
                                                                    low_writing_files=False,
                                                                    ran-
                                                                    dom_seed=0,
                                                                    n_jobs=-
                                                                    1,
                                                                    **kwargs)
```

CatBoost Classifier, a classifier that uses gradient-boosting on decision trees. CatBoost is an open-source library and natively supports categorical features.

For more information, check out <https://catboost.ai/>

**Parameters**

- **n\_estimators** (*float*) – The maximum number of trees to build. Defaults to 10.
- **eta** (*float*) – The learning rate. Defaults to 0.03.
- **max\_depth** (*int*) – The maximum tree depth for base learners. Defaults to 6.
- **bootstrap\_type** (*string*) – Defines the method for sampling the weights of objects. Available methods are ‘Bayesian’, ‘Bernoulli’, ‘MVS’. Defaults to None.
- **silent** (*boolean*) – Whether to use the “silent” logging mode. Defaults to True.
- **allow\_writing\_files** (*boolean*) – Whether to allow writing snapshot files while training. Defaults to False.

- **n\_jobs** (*int* or *None*) – Number of jobs to run in parallel. -1 uses all processes. Defaults to -1.
- **random\_seed** (*int*) – Seed for the random number generator. Defaults to 0.

#### Attributes

<b>hyper-parameter_ranges</b>	{ "n_estimators": Integer(4, 100), "eta": Real(0.000001, 1), "max_depth": Integer(4, 10), }
<b>model_family</b>	ModelFamily.CATBOOST
<b>modifies_features</b>	True
<b>modifies_target</b>	False
<b>name</b>	CatBoost Classifier
<b>predict Uses y</b>	False
<b>supported_problem_types</b>	[ ProblemTypes.BINARY, ProblemTypes.MULTICLASS, ProblemTypes.TIME_SERIES_BINARY, ProblemTypes.TIME_SERIES_MULTICLASS, ]

#### Methods

<i>clone</i>	Constructs a new component with the same parameters and random state.
<i>default_parameters</i>	Returns the default parameters for this component.
<i>describe</i>	Describe a component and its parameters
<i>feature_importance</i>	Returns importance associated with each feature.
<i>fit</i>	Fits component to data
<i>load</i>	Loads component at file path
<i>needs_fitting</i>	Returns boolean determining if component needs fitting before
<i>parameters</i>	Returns the parameters which were used to initialize the component
<i>predict</i>	Make predictions using selected features.
<i>predict_proba</i>	Make probability estimates for labels.
<i>save</i>	Saves component at file path

#### **clone** (*self*)

Constructs a new component with the same parameters and random state.

**Returns** A new instance of this component with identical parameters and random state.

#### **default\_parameters** (*cls*)

Returns the default parameters for this component.

Our convention is that `Component.default_parameters == Component().parameters`.

**Returns** default parameters for this component.

**Return type** dict

#### **describe** (*self*, *print\_name=False*, *return\_dict=False*)

Describe a component and its parameters

**Parameters**



- **print\_name** (*bool, optional*) – whether to print name of component
- **return\_dict** (*bool, optional*) – whether to return description as dictionary in the format {"name": name, "parameters": parameters}

**Returns** prints and returns dictionary

**Return type** None or dict

**property feature\_importance** (*self*)

Returns importance associated with each feature.

**Returns** Importance associated with each feature

**Return type** np.ndarray

**fit** (*self, X, y=None*)

Fits component to data

**Parameters**

- **X** (*list, pd.DataFrame or np.ndarray*) – The input training data of shape [n\_samples, n\_features]
- **y** (*list, pd.Series, np.ndarray, optional*) – The target training data of length [n\_samples]

**Returns** self

**static load** (*file\_path*)

Loads component at file path

**Parameters** **file\_path** (*str*) – Location to load file

**Returns** ComponentBase object

**needs\_fitting** (*self*)

Returns boolean determining if component needs fitting before calling predict, predict\_proba, transform, or feature\_importances. This can be overridden to False for components that do not need to be fit or whose fit methods do nothing.

**property parameters** (*self*)

Returns the parameters which were used to initialize the component

**predict** (*self, X*)

Make predictions using selected features.

**Parameters** **X** (*pd.DataFrame, np.ndarray*) – Data of shape [n\_samples, n\_features]

**Returns** Predicted values

**Return type** pd.Series

**predict\_proba** (*self, X*)

Make probability estimates for labels.

**Parameters** **X** (*pd.DataFrame, or np.ndarray*) – Features

**Returns** Probability estimates

**Return type** pd.Series

**save** (*self, file\_path, pickle\_protocol=cloudpickle.DEFAULT\_PROTOCOL*)

Saves component at file path

**Parameters**

- **file\_path** (*str*) – Location to save file
- **pickle\_protocol** (*int*) – The pickle data stream format.

**Returns** None

```
class evalml.pipelines.components.estimators.classifiers.DecisionTreeClassifier (criterion='gini',  
                                                                    max_features=  
                                                                    max_depth=6,  
                                                                    min_samples_s  
                                                                    min_weight_fra  
                                                                    ran-  
                                                                    dom_seed=0,  
                                                                    **kwargs)
```

Decision Tree Classifier.

#### Parameters

- **criterion** (*{ "gini", "entropy" }*) – The function to measure the quality of a split. Supported criteria are “gini” for the Gini impurity and “entropy” for the information gain. Defaults to “gini”.
- **max\_features** (*int, float or { "auto", "sqrt", "log2" }*) – The number of features to consider when looking for the best split:
  - If int, then consider max\_features features at each split.
  - If float, then max\_features is a fraction and int(max\_features \* n\_features) features are considered at each split.
  - If “auto”, then max\_features=sqrt(n\_features).
  - If “sqrt”, then max\_features=sqrt(n\_features).
  - If “log2”, then max\_features=log2(n\_features).
  - If None, then max\_features = n\_features.

The search for a split does not stop until at least one valid partition of the node samples is found, even if it requires to effectively inspect more than max\_features features. Defaults to “auto”.

- **max\_depth** (*int*) – The maximum depth of the tree. Defaults to 6.
- **min\_samples\_split** (*int or float*) – The minimum number of samples required to split an internal node:
  - If int, then consider min\_samples\_split as the minimum number.
  - If float, then min\_samples\_split is a fraction and ceil(min\_samples\_split \* n\_samples) are the minimum number of samples for each split.

Defaults to 2.

- **min\_weight\_fraction\_leaf** (*float*) – The minimum weighted fraction of the sum total of weights (of all the input samples) required to be at a leaf node. Defaults to 0.0.
- **random\_seed** (*int*) – Seed for the random number generator. Defaults to 0.

#### Attributes

<b>hyper-parameter_ranges</b>	{ "criterion": ["gini", "entropy"], "max_features": ["auto", "sqrt", "log2"], "max_depth": Integer(4, 10), }
<b>model_family</b>	ModelFamily.DECISION_TREE
<b>modifies_features</b>	True
<b>modifies_target</b>	False
<b>name</b>	Decision Tree Classifier
<b>pre-dict_uses_y</b>	False
<b>supported_problem_types</b>	[ ProblemTypes.BINARY, ProblemTypes.MULTICLASS, ProblemTypes.TIME_SERIES_BINARY, ProblemTypes.TIME_SERIES_MULTICLASS, ]

## Methods

<i>clone</i>	Constructs a new component with the same parameters and random state.
<i>default_parameters</i>	Returns the default parameters for this component.
<i>describe</i>	Describe a component and its parameters
<i>feature_importance</i>	Returns importance associated with each feature.
<i>fit</i>	Fits component to data
<i>load</i>	Loads component at file path
<i>needs_fitting</i>	Returns boolean determining if component needs fitting before
<i>parameters</i>	Returns the parameters which were used to initialize the component
<i>predict</i>	Make predictions using selected features.
<i>predict_proba</i>	Make probability estimates for labels.
<i>save</i>	Saves component at file path

### **clone** (*self*)

Constructs a new component with the same parameters and random state.

**Returns** A new instance of this component with identical parameters and random state.

### **default\_parameters** (*cls*)

Returns the default parameters for this component.

Our convention is that `Component.default_parameters == Component().parameters`.

**Returns** default parameters for this component.

**Return type** dict

### **describe** (*self*, *print\_name=False*, *return\_dict=False*)

Describe a component and its parameters

#### Parameters

- **print\_name** (*bool*, *optional*) – whether to print name of component
- **return\_dict** (*bool*, *optional*) – whether to return description as dictionary in the format {"name": name, "parameters": parameters}

**Returns** prints and returns dictionary

**Return type** None or dict

**property feature\_importance** (*self*)

Returns importance associated with each feature.

**Returns** Importance associated with each feature

**Return type** np.ndarray

**fit** (*self*, *X*, *y=None*)

Fits component to data

**Parameters**

- **X** (*list*, *pd.DataFrame* or *np.ndarray*) – The input training data of shape [n\_samples, n\_features]
- **y** (*list*, *pd.Series*, *np.ndarray*, *optional*) – The target training data of length [n\_samples]

**Returns** self

**static load** (*file\_path*)

Loads component at file path

**Parameters** **file\_path** (*str*) – Location to load file

**Returns** ComponentBase object

**needs\_fitting** (*self*)

Returns boolean determining if component needs fitting before calling predict, predict\_proba, transform, or feature\_importances. This can be overridden to False for components that do not need to be fit or whose fit methods do nothing.

**property parameters** (*self*)

Returns the parameters which were used to initialize the component

**predict** (*self*, *X*)

Make predictions using selected features.

**Parameters** **X** (*pd.DataFrame*, *np.ndarray*) – Data of shape [n\_samples, n\_features]

**Returns** Predicted values

**Return type** pd.Series

**predict\_proba** (*self*, *X*)

Make probability estimates for labels.

**Parameters** **X** (*pd.DataFrame*, or *np.ndarray*) – Features

**Returns** Probability estimates

**Return type** pd.Series

**save** (*self*, *file\_path*, *pickle\_protocol=cloudpickle.DEFAULT\_PROTOCOL*)

Saves component at file path

**Parameters**

- **file\_path** (*str*) – Location to save file
- **pickle\_protocol** (*int*) – The pickle data stream format.

**Returns** None

```

class evalml.pipelines.components.estimators.classifiers.ElasticNetClassifier (penalty='elasticnet',
                                                                              C=1.0,
                                                                              l1_ratio=0.15,
                                                                              multi_class='auto',
                                                                              solver='saga',
                                                                              n_jobs=-1,
                                                                              random_seed=0,
                                                                              **kwargs)

```

Elastic Net Classifier. Uses Logistic Regression with elasticnet penalty as the base estimator.

### Parameters

- **penalty** (`{ "l1", "l2", "elasticnet", "none" }`) – The norm used in penalization. Defaults to “elasticnet”.
- **C** (`float`) – Inverse of regularization strength. Must be a positive float. Defaults to 1.0.
- **l1\_ratio** (`float`) – The mixing parameter, with  $0 \leq \text{l1\_ratio} \leq 1$ . Only used if `penalty='elasticnet'`. Setting `l1_ratio=0` is equivalent to using `penalty='l2'`, while setting `l1_ratio=1` is equivalent to using `penalty='l1'`. For  $0 < \text{l1\_ratio} < 1$ , the penalty is a combination of L1 and L2. Defaults to 0.15.
- **multi\_class** (`{ "auto", "ovr", "multinomial" }`) – If the option chosen is “ovr”, then a binary problem is fit for each label. For “multinomial” the loss minimised is the multinomial loss fit across the entire probability distribution, even when the data is binary. “multinomial” is unavailable when `solver="liblinear"`. “auto” selects “ovr” if the data is binary, or if `solver="liblinear"`, and otherwise selects “multinomial”. Defaults to “auto”.
- **solver** (`{ "newton-cg", "lbfgs", "liblinear", "sag", "saga" }`) – Algorithm to use in the optimization problem. For small datasets, “liblinear” is a good choice, whereas “sag” and “saga” are faster for large ones. For multiclass problems, only “newton-cg”, “sag”, “saga” and “lbfgs” handle multinomial loss; “liblinear” is limited to one-versus-rest schemes.
  - “newton-cg”, “lbfgs”, “sag” and “saga” handle L2 or no penalty
  - “liblinear” and “saga” also handle L1 penalty
  - “saga” also supports “elasticnet” penalty
  - “liblinear” does not support setting `penalty='none'`
 Defaults to “saga”.
- **n\_jobs** (`int`) – Number of parallel threads used to run xgboost. Note that creating thread contention will significantly slow down the algorithm. Defaults to -1.
- **random\_seed** (`int`) – Seed for the random number generator. Defaults to 0.

### Attributes

<b>hyper-parameter_ranges</b>	{ "C": Real(0.01, 10), "l1_ratio": Real(0, 1)}
<b>model_family</b>	ModelFamily.LINEAR_MODEL
<b>modifies_features</b>	True
<b>modifies_target</b>	False
<b>name</b>	Elastic Net Classifier
<b>pre-dict_uses_y</b>	False
<b>supported_problem_types</b>	[ ProblemTypes.BINARY, ProblemTypes.MULTICLASS, ProblemTypes.TIME_SERIES_BINARY, ProblemTypes.TIME_SERIES_MULTICLASS,]

## Methods

<i>clone</i>	Constructs a new component with the same parameters and random state.
<i>default_parameters</i>	Returns the default parameters for this component.
<i>describe</i>	Describe a component and its parameters
<i>feature_importance</i>	Returns importance associated with each feature.
<i>fit</i>	Fits component to data
<i>load</i>	Loads component at file path
<i>needs_fitting</i>	Returns boolean determining if component needs fitting before
<i>parameters</i>	Returns the parameters which were used to initialize the component
<i>predict</i>	Make predictions using selected features.
<i>predict_proba</i>	Make probability estimates for labels.
<i>save</i>	Saves component at file path

### **clone** (*self*)

Constructs a new component with the same parameters and random state.

**Returns** A new instance of this component with identical parameters and random state.

### **default\_parameters** (*cls*)

Returns the default parameters for this component.

Our convention is that `Component.default_parameters == Component().parameters`.

**Returns** default parameters for this component.

**Return type** dict

### **describe** (*self*, *print\_name=False*, *return\_dict=False*)

Describe a component and its parameters

#### Parameters

- **print\_name** (*bool*, *optional*) – whether to print name of component
- **return\_dict** (*bool*, *optional*) – whether to return description as dictionary in the format {"name": name, "parameters": parameters}

**Returns** prints and returns dictionary

**Return type** None or dict

**property feature\_importance** (*self*)

Returns importance associated with each feature.

**Returns** Importance associated with each feature

**Return type** np.ndarray

**fit** (*self*, *X*, *y*)

Fits component to data

**Parameters**

- **X** (*list*, *pd.DataFrame* or *np.ndarray*) – The input training data of shape [n\_samples, n\_features]
- **y** (*list*, *pd.Series*, *np.ndarray*, *optional*) – The target training data of length [n\_samples]

**Returns** self

**static load** (*file\_path*)

Loads component at file path

**Parameters** **file\_path** (*str*) – Location to load file

**Returns** ComponentBase object

**needs\_fitting** (*self*)

Returns boolean determining if component needs fitting before calling predict, predict\_proba, transform, or feature\_importances. This can be overridden to False for components that do not need to be fit or whose fit methods do nothing.

**property parameters** (*self*)

Returns the parameters which were used to initialize the component

**predict** (*self*, *X*)

Make predictions using selected features.

**Parameters** **X** (*pd.DataFrame*, *np.ndarray*) – Data of shape [n\_samples, n\_features]

**Returns** Predicted values

**Return type** pd.Series

**predict\_proba** (*self*, *X*)

Make probability estimates for labels.

**Parameters** **X** (*pd.DataFrame*, or *np.ndarray*) – Features

**Returns** Probability estimates

**Return type** pd.Series

**save** (*self*, *file\_path*, *pickle\_protocol=cloudpickle.DEFAULT\_PROTOCOL*)

Saves component at file path

**Parameters**

- **file\_path** (*str*) – Location to save file
- **pickle\_protocol** (*int*) – The pickle data stream format.

**Returns** None

```
class evalml.pipelines.components.estimators.classifiers.ExtraTreesClassifier (n_estimators=100,
                                                                    max_features='auto',
                                                                    max_depth=6,
                                                                    min_samples_split=2,
                                                                    min_weight_fraction_leaf=0.0,
                                                                    n_jobs=-1,
                                                                    random_state=None,
                                                                    random_seed=0,
                                                                    **kwargs)
```

Extra Trees Classifier.

### Parameters

- **n\_estimators** (*float*) – The number of trees in the forest. Defaults to 100.
- **max\_features** (*int, float or {"auto", "sqrt", "log2"}*) – The number of features to consider when looking for the best split:
  - If *int*, then consider *max\_features* features at each split.
  - If *float*, then *max\_features* is a fraction and  $\text{int}(\text{max\_features} * \text{n\_features})$  features are considered at each split.
  - If “auto”, then  $\text{max\_features} = \sqrt{\text{n\_features}}$ .
  - If “sqrt”, then  $\text{max\_features} = \sqrt{\text{n\_features}}$ .
  - If “log2”, then  $\text{max\_features} = \log_2(\text{n\_features})$ .
  - If *None*, then  $\text{max\_features} = \text{n\_features}$ .

The search for a split does not stop until at least one valid partition of the node samples is found, even if it requires to effectively inspect more than *max\_features* features. Defaults to “auto”.
- **max\_depth** (*int*) – The maximum depth of the tree. Defaults to 6.
- **min\_samples\_split** (*int or float*) – The minimum number of samples required to split an internal node:
  - If *int*, then consider *min\_samples\_split* as the minimum number.
  - If *float*, then *min\_samples\_split* is a fraction and  $\text{ceil}(\text{min\_samples\_split} * \text{n\_samples})$  are the minimum number of samples for each split.
- **to 2.** (*Defaults*) –
- **min\_weight\_fraction\_leaf** (*float*) – The minimum weighted fraction of the sum total of weights (of all the input samples) required to be at a leaf node. Defaults to 0.0.
- **n\_jobs** (*int or None*) – Number of jobs to run in parallel. -1 uses all processes. Defaults to -1.
- **random\_seed** (*int*) – Seed for the random number generator. Defaults to 0.

### Attributes



<b>hyper-parameter_ranges</b>	{ “n_estimators”: Integer(10, 1000), “max_features”: [“auto”, “sqrt”, “log2”], “max_depth”: Integer(4, 10),}
<b>model_family</b>	ModelFamily.EXTRA_TREES
<b>modifies_features</b>	True
<b>modifies_target</b>	False
<b>name</b>	Extra Trees Classifier
<b>pre-dict_uses_y</b>	False
<b>supported_problem_types</b>	[ ProblemTypes.BINARY, ProblemTypes.MULTICLASS, ProblemTypes.TIME_SERIES_BINARY, ProblemTypes.TIME_SERIES_MULTICLASS,]

## Methods

<i>clone</i>	Constructs a new component with the same parameters and random state.
<i>default_parameters</i>	Returns the default parameters for this component.
<i>describe</i>	Describe a component and its parameters
<i>feature_importance</i>	Returns importance associated with each feature.
<i>fit</i>	Fits component to data
<i>load</i>	Loads component at file path
<i>needs_fitting</i>	Returns boolean determining if component needs fitting before
<i>parameters</i>	Returns the parameters which were used to initialize the component
<i>predict</i>	Make predictions using selected features.
<i>predict_proba</i>	Make probability estimates for labels.
<i>save</i>	Saves component at file path

### **clone** (*self*)

Constructs a new component with the same parameters and random state.

**Returns** A new instance of this component with identical parameters and random state.

### **default\_parameters** (*cls*)

Returns the default parameters for this component.

Our convention is that `Component.default_parameters == Component().parameters`.

**Returns** default parameters for this component.

**Return type** dict

### **describe** (*self*, *print\_name=False*, *return\_dict=False*)

Describe a component and its parameters

#### Parameters

- **print\_name** (*bool*, *optional*) – whether to print name of component
- **return\_dict** (*bool*, *optional*) – whether to return description as dictionary in the format {“name”: name, “parameters”: parameters}

**Returns** prints and returns dictionary

**Return type** None or dict

**property feature\_importance** (*self*)

Returns importance associated with each feature.

**Returns** Importance associated with each feature

**Return type** np.ndarray

**fit** (*self*, *X*, *y=None*)

Fits component to data

**Parameters**

- **X** (*list*, *pd.DataFrame* or *np.ndarray*) – The input training data of shape [n\_samples, n\_features]
- **y** (*list*, *pd.Series*, *np.ndarray*, *optional*) – The target training data of length [n\_samples]

**Returns** self

**static load** (*file\_path*)

Loads component at file path

**Parameters** **file\_path** (*str*) – Location to load file

**Returns** ComponentBase object

**needs\_fitting** (*self*)

Returns boolean determining if component needs fitting before calling predict, predict\_proba, transform, or feature\_importances. This can be overridden to False for components that do not need to be fit or whose fit methods do nothing.

**property parameters** (*self*)

Returns the parameters which were used to initialize the component

**predict** (*self*, *X*)

Make predictions using selected features.

**Parameters** **X** (*pd.DataFrame*, *np.ndarray*) – Data of shape [n\_samples, n\_features]

**Returns** Predicted values

**Return type** pd.Series

**predict\_proba** (*self*, *X*)

Make probability estimates for labels.

**Parameters** **X** (*pd.DataFrame*, or *np.ndarray*) – Features

**Returns** Probability estimates

**Return type** pd.Series

**save** (*self*, *file\_path*, *pickle\_protocol=cloudpickle.DEFAULT\_PROTOCOL*)

Saves component at file path

**Parameters**

- **file\_path** (*str*) – Location to save file
- **pickle\_protocol** (*int*) – The pickle data stream format.

**Returns** None

```
class evalml.pipelines.components.estimators.classifiers.KNeighborsClassifier (n_neighbors=5,  
                                                                    weights='uniform'  
                                                                    al-  
                                                                    go-  
                                                                    rithm='auto',  
                                                                    leaf_size=30,  
                                                                    p=2,  
                                                                    ran-  
                                                                    dom_seed=0,  
                                                                    **kwargs)
```

K-Nearest Neighbors Classifier.

#### Parameters

- **n\_neighbors** (*int*) – Number of neighbors to use by default. Defaults to 5.
- **weights** ({ *'uniform'*, *'distance'* } or *callable*) – Weight function used in prediction. Can be:
  - *'uniform'* : uniform weights. All points in each neighborhood are weighted equally.
  - *'distance'* : weight points by the inverse of their distance. in this case, closer neighbors of a query point will have a greater influence than neighbors which are further away.
  - [*callable*] : a user-defined function which accepts an array of distances, and returns an array of the same shape containing the weights.

Defaults to “uniform”.

- **algorithm** ({ *'auto'*, *'ball\_tree'*, *'kd\_tree'*, *'brute'* }) – Algorithm used to compute the nearest neighbors:
  - *'ball\_tree'* will use BallTree
  - *'kd\_tree'* will use KDTree
  - *'brute'* will use a brute-force search.

*'auto'* will attempt to decide the most appropriate algorithm based on the values passed to fit method. Defaults to “auto”. Note: fitting on sparse input will override the setting of this parameter, using brute force.
- **leaf\_size** (*int*) – Leaf size passed to BallTree or KDTree. This can affect the speed of the construction and query, as well as the memory required to store the tree. The optimal value depends on the nature of the problem. Defaults to 30.
- **p** (*int*) – Power parameter for the Minkowski metric. When  $p = 1$ , this is equivalent to using `manhattan_distance` (l1), and `euclidean_distance` (l2) for  $p = 2$ . For arbitrary  $p$ , `minkowski_distance` (l\_p) is used. Defaults to 2.
- **random\_seed** (*int*) – Seed for the random number generator. Defaults to 0.

#### Attributes

<b>hyper-parameter_ranges</b>	{ "n_neighbors": Integer(2, 12), "weights": ["uniform", "distance"], "algorithm": ["auto", "ball_tree", "kd_tree", "brute"], "leaf_size": Integer(10, 30), "p": Integer(1, 5), }
<b>model_family</b>	ModelFamily.K_NEIGHBORS
<b>modifies_features</b>	True
<b>modifies_target</b>	False
<b>name</b>	KNN Classifier
<b>pre-dict_uses_y</b>	False
<b>supported_problem_types</b>	[ ProblemTypes.BINARY, ProblemTypes.MULTICLASS, ProblemTypes.TIME_SERIES_BINARY, ProblemTypes.TIME_SERIES_MULTICLASS, ]

## Methods

<i>clone</i>	Constructs a new component with the same parameters and random state.
<i>default_parameters</i>	Returns the default parameters for this component.
<i>describe</i>	Describe a component and its parameters
<i>feature_importance</i>	Returns array of 0's matching the input number of features as feature_importance is
<i>fit</i>	Fits component to data
<i>load</i>	Loads component at file path
<i>needs_fitting</i>	Returns boolean determining if component needs fitting before
<i>parameters</i>	Returns the parameters which were used to initialize the component
<i>predict</i>	Make predictions using selected features.
<i>predict_proba</i>	Make probability estimates for labels.
<i>save</i>	Saves component at file path

### **clone** (*self*)

Constructs a new component with the same parameters and random state.

**Returns** A new instance of this component with identical parameters and random state.

### **default\_parameters** (*cls*)

Returns the default parameters for this component.

Our convention is that `Component.default_parameters == Component().parameters`.

**Returns** default parameters for this component.

**Return type** dict

### **describe** (*self*, *print\_name=False*, *return\_dict=False*)

Describe a component and its parameters

#### Parameters

- **print\_name** (*bool*, *optional*) – whether to print name of component
- **return\_dict** (*bool*, *optional*) – whether to return description as dictionary in the format {"name": name, "parameters": parameters}

**Returns** prints and returns dictionary

**Return type** None or dict

**property feature\_importance** (*self*)

Returns array of 0's matching the input number of features as feature\_importance is not defined for KNN classifiers.

**fit** (*self*, *X*, *y=None*)

Fits component to data

**Parameters**

- **X** (*list*, *pd.DataFrame* or *np.ndarray*) – The input training data of shape [n\_samples, n\_features]
- **y** (*list*, *pd.Series*, *np.ndarray*, *optional*) – The target training data of length [n\_samples]

**Returns** *self*

**static load** (*file\_path*)

Loads component at file path

**Parameters** **file\_path** (*str*) – Location to load file

**Returns** ComponentBase object

**needs\_fitting** (*self*)

Returns boolean determining if component needs fitting before calling predict, predict\_proba, transform, or feature\_importances. This can be overridden to False for components that do not need to be fit or whose fit methods do nothing.

**property parameters** (*self*)

Returns the parameters which were used to initialize the component

**predict** (*self*, *X*)

Make predictions using selected features.

**Parameters** **X** (*pd.DataFrame*, *np.ndarray*) – Data of shape [n\_samples, n\_features]

**Returns** Predicted values

**Return type** *pd.Series*

**predict\_proba** (*self*, *X*)

Make probability estimates for labels.

**Parameters** **X** (*pd.DataFrame*, or *np.ndarray*) – Features

**Returns** Probability estimates

**Return type** *pd.Series*

**save** (*self*, *file\_path*, *pickle\_protocol=cloudpickle.DEFAULT\_PROTOCOL*)

Saves component at file path

**Parameters**

- **file\_path** (*str*) – Location to save file
- **pickle\_protocol** (*int*) – The pickle data stream format.

**Returns** None

```
class evalml.pipelines.components.estimators.classifiers.LightGBMClassifier (boosting_type='gbdt',  
learning_rate=0.1,  
n_estimators=100,  
max_depth=0,  
num_leaves=31,  
min_child_samples=20,  
bagging_fraction=0.9,  
bagging_freq=0,  
n_jobs=-1,  
random_seed=0,  
**kwargs)
```

LightGBM Classifier.

#### Parameters

- **boosting\_type** (*string*) – Type of boosting to use. Defaults to “gbdt”. - ‘gbdt’ uses traditional Gradient Boosting Decision Tree - “dart”, uses Dropouts meet Multiple Additive Regression Trees - “goss”, uses Gradient-based One-Side Sampling - “rf”, uses Random Forest
- **learning\_rate** (*float*) – Boosting learning rate. Defaults to 0.1.
- **n\_estimators** (*int*) – Number of boosted trees to fit. Defaults to 100.
- **max\_depth** (*int*) – Maximum tree depth for base learners, <=0 means no limit. Defaults to 0.
- **num\_leaves** (*int*) – Maximum tree leaves for base learners. Defaults to 31.
- **min\_child\_samples** (*int*) – Minimum number of data needed in a child (leaf). Defaults to 20.
- **bagging\_fraction** (*float*) – LightGBM will randomly select a subset of features on each iteration (tree) without resampling if this is smaller than 1.0. For example, if set to 0.8, LightGBM will select 80% of features before training each tree. This can be used to speed up training and deal with overfitting. Defaults to 0.9.
- **bagging\_freq** (*int*) – Frequency for bagging. 0 means bagging is disabled. k means perform bagging at every k iteration. Every k-th iteration, LightGBM will randomly select bagging\_fraction \* 100 % of the data to use for the next k iterations. Defaults to 0.
- **n\_jobs** (*int or None*) – Number of threads to run in parallel. -1 uses all threads. Defaults to -1.
- **random\_seed** (*int*) – Seed for the random number generator. Defaults to 0.

#### Attributes

<b>hyper-parameter_ranges</b>	{ "learning_rate": Real(0.000001, 1), "boosting_type": ["gbdt", "dart", "goss", "rf"], "n_estimators": Integer(10, 100), "max_depth": Integer(0, 10), "num_leaves": Integer(2, 100), "min_child_samples": Integer(1, 100), "bagging_fraction": Real(0.000001, 1), "bagging_freq": Integer(0, 1),}
<b>model_family</b>	ModelFamily.LIGHTGBM
<b>modifies_features</b>	True
<b>modifies_target</b>	False
<b>name</b>	LightGBM Classifier
<b>pre-dict_uses_y</b>	False
<b>SEED_MAX</b>	SEED_BOUNDS.max_bound
<b>SEED_MIN</b>	0
<b>supported_problem_types</b>	[ ProblemTypes.BINARY, ProblemTypes.MULTICLASS, ProblemTypes.TIME_SERIES_BINARY, ProblemTypes.TIME_SERIES_MULTICLASS,]

## Methods

<i>clone</i>	Constructs a new component with the same parameters and random state.
<i>default_parameters</i>	Returns the default parameters for this component.
<i>describe</i>	Describe a component and its parameters
<i>feature_importance</i>	Returns importance associated with each feature.
<i>fit</i>	Fits component to data
<i>load</i>	Loads component at file path
<i>needs_fitting</i>	Returns boolean determining if component needs fitting before
<i>parameters</i>	Returns the parameters which were used to initialize the component
<i>predict</i>	Make predictions using selected features.
<i>predict_proba</i>	Make probability estimates for labels.
<i>save</i>	Saves component at file path

### **clone** (*self*)

Constructs a new component with the same parameters and random state.

**Returns** A new instance of this component with identical parameters and random state.

### **default\_parameters** (*cls*)

Returns the default parameters for this component.

Our convention is that `Component.default_parameters == Component().parameters`.

**Returns** default parameters for this component.

**Return type** dict

### **describe** (*self*, *print\_name=False*, *return\_dict=False*)

Describe a component and its parameters

#### Parameters

- **print\_name** (*bool*, *optional*) – whether to print name of component

- **return\_dict** (*bool, optional*) – whether to return description as dictionary in the format {"name": name, "parameters": parameters}

**Returns** prints and returns dictionary

**Return type** None or dict

**property feature\_importance** (*self*)

Returns importance associated with each feature.

**Returns** Importance associated with each feature

**Return type** np.ndarray

**fit** (*self, X, y=None*)

Fits component to data

**Parameters**

- **X** (*list, pd.DataFrame or np.ndarray*) – The input training data of shape [n\_samples, n\_features]
- **y** (*list, pd.Series, np.ndarray, optional*) – The target training data of length [n\_samples]

**Returns** self

**static load** (*file\_path*)

Loads component at file path

**Parameters** **file\_path** (*str*) – Location to load file

**Returns** ComponentBase object

**needs\_fitting** (*self*)

Returns boolean determining if component needs fitting before calling predict, predict\_proba, transform, or feature\_importances. This can be overridden to False for components that do not need to be fit or whose fit methods do nothing.

**property parameters** (*self*)

Returns the parameters which were used to initialize the component

**predict** (*self, X*)

Make predictions using selected features.

**Parameters** **X** (*pd.DataFrame, np.ndarray*) – Data of shape [n\_samples, n\_features]

**Returns** Predicted values

**Return type** pd.Series

**predict\_proba** (*self, X*)

Make probability estimates for labels.

**Parameters** **X** (*pd.DataFrame, or np.ndarray*) – Features

**Returns** Probability estimates

**Return type** pd.Series

**save** (*self, file\_path, pickle\_protocol=cloudpickle.DEFAULT\_PROTOCOL*)

Saves component at file path

**Parameters**

- **file\_path** (*str*) – Location to save file



- **pickle\_protocol** (*int*) – The pickle data stream format.

**Returns** None

```
class evalml.pipelines.components.estimators.classifiers.LogisticRegressionClassifier (penalty,
C=1.0,
multi_
solver:
n_jobs

l,
ran-
dom_s
**kwa
```

Logistic Regression Classifier.

#### Parameters

- **penalty** ({*"l1"*, *"l2"*, *"elasticnet"*, *"none"*}) – The norm used in penalization. Defaults to *"l2"*.
- **C** (*float*) – Inverse of regularization strength. Must be a positive float. Defaults to 1.0.
- **multi\_class** ({*"auto"*, *"ovr"*, *"multinomial"*}) – If the option chosen is *"ovr"*, then a binary problem is fit for each label. For *"multinomial"* the loss minimised is the multinomial loss fit across the entire probability distribution, even when the data is binary. *"multinomial"* is unavailable when *solver="liblinear"*. *"auto"* selects *"ovr"* if the data is binary, or if *solver="liblinear"*, and otherwise selects *"multinomial"*. Defaults to *"auto"*.
- **solver** ({*"newton-cg"*, *"lbfgs"*, *"liblinear"*, *"sag"*, *"saga"*}) – Algorithm to use in the optimization problem. For small datasets, *"liblinear"* is a good choice, whereas *"sag"* and *"saga"* are faster for large ones. For multiclass problems, only *"newton-cg"*, *"sag"*, *"saga"* and *"lbfgs"* handle multinomial loss; *"liblinear"* is limited to one-versus-rest schemes.
  - *"newton-cg"*, *"lbfgs"*, *"sag"* and *"saga"* handle L2 or no penalty
  - *"liblinear"* and *"saga"* also handle L1 penalty
  - *"saga"* also supports *"elasticnet"* penalty
  - *"liblinear"* does not support setting *penalty='none'*

Defaults to *"lbfgs"*.

- **n\_jobs** (*int*) – Number of parallel threads used to run xgboost. Note that creating thread contention will significantly slow down the algorithm. Defaults to -1.
- **random\_seed** (*int*) – Seed for the random number generator. Defaults to 0.

#### Attributes

<b>hyper-parameter_ranges</b>	{ "penalty": ["l2"], "C": Real(0.01, 10), }
<b>model_family</b>	ModelFamily.LINEAR_MODEL
<b>modifies_features</b>	True
<b>modifies_target</b>	False
<b>name</b>	Logistic Regression Classifier
<b>pre_dict_uses_y</b>	False
<b>supported_problem_types</b>	[ ProblemTypes.BINARY, ProblemTypes.MULTICLASS, ProblemTypes.TIME_SERIES_BINARY, ProblemTypes.TIME_SERIES_MULTICLASS, ]

## Methods

<i>clone</i>	Constructs a new component with the same parameters and random state.
<i>default_parameters</i>	Returns the default parameters for this component.
<i>describe</i>	Describe a component and its parameters
<i>feature_importance</i>	Returns importance associated with each feature.
<i>fit</i>	Fits component to data
<i>load</i>	Loads component at file path
<i>needs_fitting</i>	Returns boolean determining if component needs fitting before
<i>parameters</i>	Returns the parameters which were used to initialize the component
<i>predict</i>	Make predictions using selected features.
<i>predict_proba</i>	Make probability estimates for labels.
<i>save</i>	Saves component at file path

### **clone** (*self*)

Constructs a new component with the same parameters and random state.

**Returns** A new instance of this component with identical parameters and random state.

### **default\_parameters** (*cls*)

Returns the default parameters for this component.

Our convention is that `Component.default_parameters == Component().parameters`.

**Returns** default parameters for this component.

**Return type** dict

### **describe** (*self*, *print\_name=False*, *return\_dict=False*)

Describe a component and its parameters

#### Parameters

- **print\_name** (*bool*, *optional*) – whether to print name of component
- **return\_dict** (*bool*, *optional*) – whether to return description as dictionary in the format {"name": name, "parameters": parameters}

**Returns** prints and returns dictionary

**Return type** None or dict

**property feature\_importance** (*self*)

Returns importance associated with each feature.

**Returns** Importance associated with each feature

**Return type** np.ndarray

**fit** (*self*, *X*, *y=None*)

Fits component to data

**Parameters**

- **X** (*list*, *pd.DataFrame* or *np.ndarray*) – The input training data of shape [n\_samples, n\_features]
- **y** (*list*, *pd.Series*, *np.ndarray*, *optional*) – The target training data of length [n\_samples]

**Returns** self

**static load** (*file\_path*)

Loads component at file path

**Parameters** **file\_path** (*str*) – Location to load file

**Returns** ComponentBase object

**needs\_fitting** (*self*)

Returns boolean determining if component needs fitting before calling predict, predict\_proba, transform, or feature\_importances. This can be overridden to False for components that do not need to be fit or whose fit methods do nothing.

**property parameters** (*self*)

Returns the parameters which were used to initialize the component

**predict** (*self*, *X*)

Make predictions using selected features.

**Parameters** **X** (*pd.DataFrame*, *np.ndarray*) – Data of shape [n\_samples, n\_features]

**Returns** Predicted values

**Return type** pd.Series

**predict\_proba** (*self*, *X*)

Make probability estimates for labels.

**Parameters** **X** (*pd.DataFrame*, or *np.ndarray*) – Features

**Returns** Probability estimates

**Return type** pd.Series

**save** (*self*, *file\_path*, *pickle\_protocol=cloudpickle.DEFAULT\_PROTOCOL*)

Saves component at file path

**Parameters**

- **file\_path** (*str*) – Location to save file
- **pickle\_protocol** (*int*) – The pickle data stream format.

**Returns** None

```

class evalml.pipelines.components.estimators.classifiers.RandomForestClassifier(n_estimators=100,
                                                                              max_depth=6,
                                                                              n_jobs=-1,
                                                                              random_state=0,
                                                                              **kwargs)

```

Random Forest Classifier.

#### Parameters

- **n\_estimators** (*float*) – The number of trees in the forest. Defaults to 100.
- **max\_depth** (*int*) – Maximum tree depth for base learners. Defaults to 6.
- **n\_jobs** (*int or None*) – Number of jobs to run in parallel. -1 uses all processes. Defaults to -1.
- **random\_seed** (*int*) – Seed for the random number generator. Defaults to 0.

#### Attributes

<b>hyper-parameter_ranges</b>	{ "n_estimators": Integer(10, 1000), "max_depth": Integer(1, 10), }
<b>model_family</b>	ModelFamily.RANDOM_FOREST
<b>modifies_features</b>	True
<b>modifies_target</b>	False
<b>name</b>	Random Forest Classifier
<b>predict Uses y</b>	False
<b>supported_problem_types</b>	[ ProblemTypes.BINARY, ProblemTypes.MULTICLASS, ProblemTypes.TIME_SERIES_BINARY, ProblemTypes.TIME_SERIES_MULTICLASS, ]

#### Methods

<i>clone</i>	Constructs a new component with the same parameters and random state.
<i>default_parameters</i>	Returns the default parameters for this component.
<i>describe</i>	Describe a component and its parameters
<i>feature_importance</i>	Returns importance associated with each feature.
<i>fit</i>	Fits component to data
<i>load</i>	Loads component at file path
<i>needs_fitting</i>	Returns boolean determining if component needs fitting before
<i>parameters</i>	Returns the parameters which were used to initialize the component
<i>predict</i>	Make predictions using selected features.
<i>predict_proba</i>	Make probability estimates for labels.
<i>save</i>	Saves component at file path

**clone** (*self*)

Constructs a new component with the same parameters and random state.

**Returns** A new instance of this component with identical parameters and random state.

**default\_parameters** (*cls*)

Returns the default parameters for this component.

Our convention is that `Component.default_parameters == Component().parameters`.

**Returns** default parameters for this component.

**Return type** dict

**describe** (*self*, *print\_name=False*, *return\_dict=False*)

Describe a component and its parameters

**Parameters**

- **print\_name** (*bool*, *optional*) – whether to print name of component
- **return\_dict** (*bool*, *optional*) – whether to return description as dictionary in the format {"name": name, "parameters": parameters}

**Returns** prints and returns dictionary

**Return type** None or dict

**property feature\_importance** (*self*)

Returns importance associated with each feature.

**Returns** Importance associated with each feature

**Return type** np.ndarray

**fit** (*self*, *X*, *y=None*)

Fits component to data

**Parameters**

- **X** (*list*, *pd.DataFrame* or *np.ndarray*) – The input training data of shape [n\_samples, n\_features]
- **y** (*list*, *pd.Series*, *np.ndarray*, *optional*) – The target training data of length [n\_samples]

**Returns** self

**static load** (*file\_path*)

Loads component at file path

**Parameters** **file\_path** (*str*) – Location to load file

**Returns** ComponentBase object

**needs\_fitting** (*self*)

Returns boolean determining if component needs fitting before calling predict, predict\_proba, transform, or feature\_importances. This can be overridden to False for components that do not need to be fit or whose fit methods do nothing.

**property parameters** (*self*)

Returns the parameters which were used to initialize the component

**predict** (*self*, *X*)

Make predictions using selected features.

**Parameters** **X** (*pd.DataFrame*, *np.ndarray*) – Data of shape [n\_samples, n\_features]

**Returns** Predicted values

**Return type** `pd.Series`

**predict\_proba** (*self*, *X*)

Make probability estimates for labels.

**Parameters** **X** (*pd.DataFrame*, or *np.ndarray*) – Features

**Returns** Probability estimates

**Return type** `pd.Series`

**save** (*self*, *file\_path*, *pickle\_protocol=cloudpickle.DEFAULT\_PROTOCOL*)

Saves component at file path

**Parameters**

- **file\_path** (*str*) – Location to save file
- **pickle\_protocol** (*int*) – The pickle data stream format.

**Returns** None

```
class evalml.pipelines.components.estimators.classifiers.SVMClassifier (C=1.0,  
                                                                    ker-  
                                                                    nel='rbf',  
                                                                    gamma='scale',  
                                                                    prob-  
                                                                    a-  
                                                                    bil-  
                                                                    ity=True,  
                                                                    ran-  
                                                                    dom_seed=0,  
                                                                    **kwargs)
```

Support Vector Machine Classifier.

**Parameters**

- **C** (*float*) – The regularization parameter. The strength of the regularization is inversely proportional to C. Must be strictly positive. The penalty is a squared l2 penalty. Defaults to 1.0.
- **kernel** (*{ "linear", "poly", "rbf", "sigmoid", "precomputed" }*) – Specifies the kernel type to be used in the algorithm. Defaults to “rbf”.
- **gamma** (*{ "scale", "auto" } or float*) – Kernel coefficient for “rbf”, “poly” and “sigmoid”. Defaults to “scale”. - If gamma=’scale’ (default) is passed then it uses  $1 / (n\_features * X.var())$  as value of gamma - if “auto”, uses  $1 / n\_features$
- **probability** (*boolean*) – Whether to enable probability estimates. Defaults to True.
- **random\_seed** (*int*) – Seed for the random number generator. Defaults to 0.

**Attributes**

<b>hyper-parameter_ranges</b>	{ "C": Real(0, 10), "kernel": ["linear", "poly", "rbf", "sigmoid", "precomputed"], "gamma": ["scale", "auto"], }
<b>model_family</b>	ModelFamily.SVM
<b>modifies_features</b>	True
<b>modifies_target</b>	False
<b>name</b>	SVM Classifier
<b>pre-dict_uses_y</b>	False
<b>supported_problem_types</b>	[ ProblemTypes.BINARY, ProblemTypes.MULTICLASS, ProblemTypes.TIME_SERIES_BINARY, ProblemTypes.TIME_SERIES_MULTICLASS, ]

## Methods

<i>clone</i>	Constructs a new component with the same parameters and random state.
<i>default_parameters</i>	Returns the default parameters for this component.
<i>describe</i>	Describe a component and its parameters
<i>feature_importance</i>	Feature importance only works with linear kernels.
<i>fit</i>	Fits component to data
<i>load</i>	Loads component at file path
<i>needs_fitting</i>	Returns boolean determining if component needs fitting before
<i>parameters</i>	Returns the parameters which were used to initialize the component
<i>predict</i>	Make predictions using selected features.
<i>predict_proba</i>	Make probability estimates for labels.
<i>save</i>	Saves component at file path

### **clone** (*self*)

Constructs a new component with the same parameters and random state.

**Returns** A new instance of this component with identical parameters and random state.

### **default\_parameters** (*cls*)

Returns the default parameters for this component.

Our convention is that `Component.default_parameters == Component().parameters`.

**Returns** default parameters for this component.

**Return type** dict

### **describe** (*self*, *print\_name=False*, *return\_dict=False*)

Describe a component and its parameters

#### Parameters

- **print\_name** (*bool*, *optional*) – whether to print name of component
- **return\_dict** (*bool*, *optional*) – whether to return description as dictionary in the format {"name": name, "parameters": parameters}

**Returns** prints and returns dictionary

**Return type** None or dict

**property feature\_importance** (*self*)

Feature importance only works with linear kernels. If the kernel isn't linear, we return a numpy array of zeros

**fit** (*self*, *X*, *y=None*)

Fits component to data

**Parameters**

- **X** (*list*, *pd.DataFrame* or *np.ndarray*) – The input training data of shape [n\_samples, n\_features]
- **y** (*list*, *pd.Series*, *np.ndarray*, *optional*) – The target training data of length [n\_samples]

**Returns** *self*

**static load** (*file\_path*)

Loads component at file path

**Parameters** **file\_path** (*str*) – Location to load file

**Returns** ComponentBase object

**needs\_fitting** (*self*)

Returns boolean determining if component needs fitting before calling predict, predict\_proba, transform, or feature\_importances. This can be overridden to False for components that do not need to be fit or whose fit methods do nothing.

**property parameters** (*self*)

Returns the parameters which were used to initialize the component

**predict** (*self*, *X*)

Make predictions using selected features.

**Parameters** **X** (*pd.DataFrame*, *np.ndarray*) – Data of shape [n\_samples, n\_features]

**Returns** Predicted values

**Return type** *pd.Series*

**predict\_proba** (*self*, *X*)

Make probability estimates for labels.

**Parameters** **X** (*pd.DataFrame*, or *np.ndarray*) – Features

**Returns** Probability estimates

**Return type** *pd.Series*

**save** (*self*, *file\_path*, *pickle\_protocol=cloudpickle.DEFAULT\_PROTOCOL*)

Saves component at file path

**Parameters**

- **file\_path** (*str*) – Location to save file
- **pickle\_protocol** (*int*) – The pickle data stream format.

**Returns** None



```

class evalml.pipelines.components.estimators.classifiers.XGBoostClassifier(eta=0.1,
                                                                           max_depth=6,
                                                                           min_child_weight=1,
                                                                           n_estimators=100,
                                                                           random_seed=0,
                                                                           n_jobs=-1,
                                                                           **kwargs)

```

XGBoost Classifier.

#### Parameters

- **eta** (*float*) – Boosting learning rate. Defaults to 0.1.
- **max\_depth** (*int*) – Maximum tree depth for base learners. Defaults to 6.
- **min\_child\_weight** (*float*) – Minimum sum of instance weight (hessian) needed in a child. Defaults to 1.0
- **n\_estimators** (*int*) – Number of gradient boosted trees. Equivalent to number of boosting rounds. Defaults to 100.
- **random\_seed** (*int*) – Seed for the random number generator. Defaults to 0.
- **n\_jobs** (*int*) – Number of parallel threads used to run xgboost. Note that creating thread contention will significantly slow down the algorithm. Defaults to -1.

#### Attributes

<b>hyper-parameter_ranges</b>	{ "eta": Real(0.000001, 1), "max_depth": Integer(1, 10), "min_child_weight": Real(1, 10), "n_estimators": Integer(1, 1000), }
<b>model_family</b>	ModelFamily.XGBOOST
<b>modifies_features</b>	True
<b>modifies_target</b>	False
<b>name</b>	XGBoost Classifier
<b>predict Uses y</b>	False
<b>SEED_MAX</b>	None
<b>SEED_MIN</b>	None
<b>supported_problem_types</b>	[ ProblemTypes.BINARY, ProblemTypes.MULTICLASS, ProblemTypes.TIME_SERIES_BINARY, ProblemTypes.TIME_SERIES_MULTICLASS, ]

#### Methods

<i>clone</i>	Constructs a new component with the same parameters and random state.
<i>default_parameters</i>	Returns the default parameters for this component.
<i>describe</i>	Describe a component and its parameters
<i>feature_importance</i>	Returns importance associated with each feature.
<i>fit</i>	Fits component to data
<i>load</i>	Loads component at file path

continues on next page

Table 333 – continued from previous page

<code>needs_fitting</code>	Returns boolean determining if component needs fitting before
<code>parameters</code>	Returns the parameters which were used to initialize the component
<code>predict</code>	Make predictions using selected features.
<code>predict_proba</code>	Make probability estimates for labels.
<code>save</code>	Saves component at file path

**clone** (*self*)

Constructs a new component with the same parameters and random state.

**Returns** A new instance of this component with identical parameters and random state.

**default\_parameters** (*cls*)

Returns the default parameters for this component.

Our convention is that `Component.default_parameters == Component().parameters`.

**Returns** default parameters for this component.

**Return type** dict

**describe** (*self*, *print\_name=False*, *return\_dict=False*)

Describe a component and its parameters

**Parameters**

- **print\_name** (*bool*, *optional*) – whether to print name of component
- **return\_dict** (*bool*, *optional*) – whether to return description as dictionary in the format {"name": name, "parameters": parameters}

**Returns** prints and returns dictionary

**Return type** None or dict

**property feature\_importance** (*self*)

Returns importance associated with each feature.

**Returns** Importance associated with each feature

**Return type** np.ndarray

**fit** (*self*, *X*, *y=None*)

Fits component to data

**Parameters**

- **X** (*list*, *pd.DataFrame* or *np.ndarray*) – The input training data of shape [n\_samples, n\_features]
- **y** (*list*, *pd.Series*, *np.ndarray*, *optional*) – The target training data of length [n\_samples]

**Returns** self

**static load** (*file\_path*)

Loads component at file path

**Parameters** **file\_path** (*str*) – Location to load file

**Returns** ComponentBase object

**needs\_fitting** (*self*)

Returns boolean determining if component needs fitting before calling predict, predict\_proba, transform, or feature\_importances. This can be overridden to False for components that do not need to be fit or whose fit methods do nothing.

**property parameters** (*self*)

Returns the parameters which were used to initialize the component

**predict** (*self*, *X*)

Make predictions using selected features.

**Parameters** *X* (*pd.DataFrame*, *np.ndarray*) – Data of shape [n\_samples, n\_features]

**Returns** Predicted values

**Return type** *pd.Series*

**predict\_proba** (*self*, *X*)

Make probability estimates for labels.

**Parameters** *X* (*pd.DataFrame*, *or np.ndarray*) – Features

**Returns** Probability estimates

**Return type** *pd.Series*

**save** (*self*, *file\_path*, *pickle\_protocol=cloudpickle.DEFAULT\_PROTOCOL*)

Saves component at file path

**Parameters**

- **file\_path** (*str*) – Location to save file
- **pickle\_protocol** (*int*) – The pickle data stream format.

**Returns** None

**regressors****Submodules****arima\_regressor****Module Contents****Classes Summary**


---

*ARIMARegressor*


---

Autoregressive Integrated Moving Average Model.

---

## Contents

**class** evalml.pipelines.components.estimators.regressors.arima\_regressor.**ARIMAREgressor** (date\_index, trend, start\_index, d=0, start\_q, max\_p, max\_d, max\_q, seasonal, n\_jobs, random\_seed, \*\*kwargs)

Autoregressive Integrated Moving Average Model. The three parameters (p, d, q) are the AR order, the degree of differencing, and the MA order. More information here: [https://www.statsmodels.org/devel/generated/statsmodels.tsa.arima\\_model.ARIMA.html](https://www.statsmodels.org/devel/generated/statsmodels.tsa.arima_model.ARIMA.html)

Currently ARIMAREgressor isn't supported via conda install. It's recommended that it be installed via PyPI.

### Parameters

- **date\_index** (*str*) – Specifies the name of the column in X that provides the datetime objects. Defaults to None.
- **trend** (*str*) – Controls the deterministic trend. Options are ['n', 'c', 't', 'ct'] where 'c' is a constant term, 't' indicates a linear trend, and 'ct' is both. Can also be an iterable when defining a polynomial, such as [1, 1, 0, 1].
- **start\_p** (*int*) – Minimum Autoregressive order. Defaults to 2.
- **d** (*int*) – Minimum Differencing degree. Defaults to 0.
- **start\_q** (*int*) – Minimum Moving Average order. Defaults to 2.
- **max\_p** (*int*) – Maximum Autoregressive order. Defaults to 5.
- **max\_d** (*int*) – Maximum Differencing degree. Defaults to 2.
- **max\_q** (*int*) – Maximum Moving Average order. Defaults to 5.
- **seasonal** (*boolean*) – Whether to fit a seasonal model to ARIMA. Defaults to True.
- **n\_jobs** (*int or None*) – Non-negative integer describing level of parallelism used for pipelines. Defaults to -1.
- **random\_seed** (*int*) – Seed for the random number generator. Defaults to 0.

### Attributes

<b>hyper-parameter_ranges</b>	{ "start_p": Integer(1, 3), "d": Integer(0, 2), "start_q": Integer(1, 3), "max_p": Integer(3, 10), "max_d": Integer(2, 5), "max_q": Integer(3, 10), "seasonal": [True, False], }
<b>model_family</b>	ModelFamily.ARIMA
<b>modifies_features</b>	True
<b>modifies_target</b>	False
<b>name</b>	ARIMA Regressor
<b>pre-dict_uses_y</b>	False
<b>supported_problem_types</b>	[ProblemTypes.TIME_SERIES_REGRESSION]

## Methods

<i>clone</i>	Constructs a new component with the same parameters and random state.
<i>default_parameters</i>	Returns the default parameters for this component.
<i>describe</i>	Describe a component and its parameters
<i>feature_importance</i>	Returns array of 0's with a length of 1 as feature_importance is not defined for ARIMA regressor.
<i>fit</i>	Fits component to data
<i>load</i>	Loads component at file path
<i>needs_fitting</i>	Returns boolean determining if component needs fitting before
<i>parameters</i>	Returns the parameters which were used to initialize the component
<i>predict</i>	Make predictions using selected features.
<i>predict_proba</i>	Make probability estimates for labels.
<i>save</i>	Saves component at file path

### **clone** (*self*)

Constructs a new component with the same parameters and random state.

**Returns** A new instance of this component with identical parameters and random state.

### **default\_parameters** (*cls*)

Returns the default parameters for this component.

Our convention is that `Component.default_parameters == Component().parameters`.

**Returns** default parameters for this component.

**Return type** dict

### **describe** (*self*, *print\_name=False*, *return\_dict=False*)

Describe a component and its parameters

#### Parameters

- **print\_name** (*bool*, *optional*) – whether to print name of component
- **return\_dict** (*bool*, *optional*) – whether to return description as dictionary in the format {"name": name, "parameters": parameters}

**Returns** prints and returns dictionary

**Return type** None or dict

**property feature\_importance** (*self*)

Returns array of 0's with a length of 1 as feature\_importance is not defined for ARIMA regressor.

**fit** (*self*, *X*, *y=None*)

Fits component to data

**Parameters**

- **X** (*list*, *pd.DataFrame* or *np.ndarray*) – The input training data of shape [n\_samples, n\_features]
- **y** (*list*, *pd.Series*, *np.ndarray*, *optional*) – The target training data of length [n\_samples]

**Returns** *self*

**static load** (*file\_path*)

Loads component at file path

**Parameters** **file\_path** (*str*) – Location to load file

**Returns** ComponentBase object

**needs\_fitting** (*self*)

Returns boolean determining if component needs fitting before calling predict, predict\_proba, transform, or feature\_importances. This can be overridden to False for components that do not need to be fit or whose fit methods do nothing.

**property parameters** (*self*)

Returns the parameters which were used to initialize the component

**predict** (*self*, *X*, *y=None*)

Make predictions using selected features.

**Parameters** **X** (*pd.DataFrame*, *np.ndarray*) – Data of shape [n\_samples, n\_features]

**Returns** Predicted values

**Return type** *pd.Series*

**predict\_proba** (*self*, *X*)

Make probability estimates for labels.

**Parameters** **X** (*pd.DataFrame*, or *np.ndarray*) – Features

**Returns** Probability estimates

**Return type** *pd.Series*

**save** (*self*, *file\_path*, *pickle\_protocol=cloudpickle.DEFAULT\_PROTOCOL*)

Saves component at file path

**Parameters**

- **file\_path** (*str*) – Location to save file
- **pickle\_protocol** (*int*) – The pickle data stream format.

**Returns** None

## baseline\_regressor

### Module Contents

### Classes Summary

<i>BaselineRegressor</i>	Baseline regressor that uses a simple strategy to make predictions.
--------------------------	---

### Contents

**class** evalml.pipelines.components.estimators.regressors.baseline\_regressor.**BaselineRegressor**

Baseline regressor that uses a simple strategy to make predictions. This is useful as a simple baseline regressor to compare with other regressors.

#### Parameters

- **strategy** (*str*) – Method used to predict. Valid options are “mean”, “median”. Defaults to “mean”.
- **random\_seed** (*int*) – Seed for the random number generator. Defaults to 0.

#### Attributes

<b>hyper-parameter_ranges</b>	{}
<b>model_family</b>	ModelFamily.BASELINE
<b>modifies_features</b>	True
<b>modifies_target</b>	False
<b>name</b>	Baseline Regressor
<b>predict_uses_y</b>	False
<b>supported_problem_types</b>	[ ProblemTypes.REGRESSION, ProblemTypes.TIME_SERIES_REGRESSION,]

#### Methods

<i>clone</i>	Constructs a new component with the same parameters and random state.
<i>default_parameters</i>	Returns the default parameters for this component.
<i>describe</i>	Describe a component and its parameters
<i>feature_importance</i>	Returns importance associated with each feature. Since baseline regressors do not use input features to calculate predictions, returns an array of zeroes.
<i>fit</i>	Fits component to data

continues on next page

Table 337 – continued from previous page

<code>load</code>	Loads component at file path
<code>needs_fitting</code>	Returns boolean determining if component needs fitting before
<code>parameters</code>	Returns the parameters which were used to initialize the component
<code>predict</code>	Make predictions using selected features.
<code>predict_proba</code>	Make probability estimates for labels.
<code>save</code>	Saves component at file path

**clone** (*self*)

Constructs a new component with the same parameters and random state.

**Returns** A new instance of this component with identical parameters and random state.

**default\_parameters** (*cls*)

Returns the default parameters for this component.

Our convention is that `Component.default_parameters == Component().parameters`.

**Returns** default parameters for this component.

**Return type** dict

**describe** (*self*, *print\_name=False*, *return\_dict=False*)

Describe a component and its parameters

**Parameters**

- **print\_name** (*bool*, *optional*) – whether to print name of component
- **return\_dict** (*bool*, *optional*) – whether to return description as dictionary in the format {"name": name, "parameters": parameters}

**Returns** prints and returns dictionary

**Return type** None or dict

**property feature\_importance** (*self*)

Returns importance associated with each feature. Since baseline regressors do not use input features to calculate predictions, returns an array of zeroes.

**Returns** An array of zeroes

**Return type** np.ndarray (float)

**fit** (*self*, *X*, *y=None*)

Fits component to data

**Parameters**

- **X** (*list*, *pd.DataFrame* or *np.ndarray*) – The input training data of shape [n\_samples, n\_features]
- **y** (*list*, *pd.Series*, *np.ndarray*, *optional*) – The target training data of length [n\_samples]

**Returns** self

**static load** (*file\_path*)

Loads component at file path

**Parameters** **file\_path** (*str*) – Location to load file

**Returns** ComponentBase object



**needs\_fitting** (*self*)

Returns boolean determining if component needs fitting before calling predict, predict\_proba, transform, or feature\_importances. This can be overridden to False for components that do not need to be fit or whose fit methods do nothing.

**property parameters** (*self*)

Returns the parameters which were used to initialize the component

**predict** (*self*, *X*)

Make predictions using selected features.

**Parameters** *X* (*pd.DataFrame*, *np.ndarray*) – Data of shape [n\_samples, n\_features]

**Returns** Predicted values

**Return type** *pd.Series*

**predict\_proba** (*self*, *X*)

Make probability estimates for labels.

**Parameters** *X* (*pd.DataFrame*, or *np.ndarray*) – Features

**Returns** Probability estimates

**Return type** *pd.Series*

**save** (*self*, *file\_path*, *pickle\_protocol=cloudpickle.DEFAULT\_PROTOCOL*)

Saves component at file path

**Parameters**

- **file\_path** (*str*) – Location to save file
- **pickle\_protocol** (*int*) – The pickle data stream format.

**Returns** None

**catboost\_regressor****Module Contents****Classes Summary**


---

*CatBoostRegressor*

CatBoost Regressor, a regressor that uses gradient-boosting on decision trees.

---

## Contents

**class** evalml.pipelines.components.estimators.regressors.catboost\_regressor.CatBoostRegressor

CatBoost Regressor, a regressor that uses gradient-boosting on decision trees. CatBoost is an open-source library and natively supports categorical features.

For more information, check out <https://catboost.ai/>

### Parameters

- **n\_estimators** (*float*) – The maximum number of trees to build. Defaults to 10.
- **eta** (*float*) – The learning rate. Defaults to 0.03.
- **max\_depth** (*int*) – The maximum tree depth for base learners. Defaults to 6.
- **bootstrap\_type** (*string*) – Defines the method for sampling the weights of objects. Available methods are ‘Bayesian’, ‘Bernoulli’, ‘MVS’. Defaults to None.
- **silent** (*boolean*) – Whether to use the “silent” logging mode. Defaults to True.
- **allow\_writing\_files** (*boolean*) – Whether to allow writing snapshot files while training. Defaults to False.
- **n\_jobs** (*int or None*) – Number of jobs to run in parallel. -1 uses all processes. Defaults to -1.
- **random\_seed** (*int*) – Seed for the random number generator. Defaults to 0.

### Attributes

<b>hyper-parameter_ranges</b>	{ “n_estimators”: Integer(4, 100), “eta”: Real(0.000001, 1), “max_depth”: Integer(4, 10), }
<b>model_family</b>	ModelFamily.CATBOOST
<b>modifies_features</b>	True
<b>modifies_target</b>	False
<b>name</b>	CatBoost Regressor
<b>pre-dict_uses_y</b>	False
<b>supported_problem_types</b>	[ ProblemTypes.REGRESSION, ProblemTypes.TIME_SERIES_REGRESSION, ]

### Methods

<code>clone</code>	Constructs a new component with the same parameters and random state.
<code>default_parameters</code>	Returns the default parameters for this component.
<code>describe</code>	Describe a component and its parameters
<code>feature_importance</code>	Returns importance associated with each feature.
<code>fit</code>	Fits component to data
<code>load</code>	Loads component at file path
<code>needs_fitting</code>	Returns boolean determining if component needs fitting before
<code>parameters</code>	Returns the parameters which were used to initialize the component
<code>predict</code>	Make predictions using selected features.
<code>predict_proba</code>	Make probability estimates for labels.
<code>save</code>	Saves component at file path

**clone** (*self*)

Constructs a new component with the same parameters and random state.

**Returns** A new instance of this component with identical parameters and random state.

**default\_parameters** (*cls*)

Returns the default parameters for this component.

Our convention is that `Component.default_parameters == Component().parameters`.

**Returns** default parameters for this component.

**Return type** dict

**describe** (*self*, *print\_name=False*, *return\_dict=False*)

Describe a component and its parameters

**Parameters**

- **print\_name** (*bool*, *optional*) – whether to print name of component
- **return\_dict** (*bool*, *optional*) – whether to return description as dictionary in the format {"name": name, "parameters": parameters}

**Returns** prints and returns dictionary

**Return type** None or dict

**property feature\_importance** (*self*)

Returns importance associated with each feature.

**Returns** Importance associated with each feature

**Return type** np.ndarray

**fit** (*self*, *X*, *y=None*)

Fits component to data

**Parameters**

- **X** (*list*, *pd.DataFrame* or *np.ndarray*) – The input training data of shape [n\_samples, n\_features]
- **y** (*list*, *pd.Series*, *np.ndarray*, *optional*) – The target training data of length [n\_samples]

**Returns** self

**static load** (*file\_path*)

Loads component at file path

**Parameters** *file\_path* (*str*) – Location to load file

**Returns** ComponentBase object

**needs\_fitting** (*self*)

Returns boolean determining if component needs fitting before calling predict, predict\_proba, transform, or feature\_importances. This can be overridden to False for components that do not need to be fit or whose fit methods do nothing.

**property parameters** (*self*)

Returns the parameters which were used to initialize the component

**predict** (*self*, *X*)

Make predictions using selected features.

**Parameters** *X* (*pd.DataFrame*, *np.ndarray*) – Data of shape [n\_samples, n\_features]

**Returns** Predicted values

**Return type** *pd.Series*

**predict\_proba** (*self*, *X*)

Make probability estimates for labels.

**Parameters** *X* (*pd.DataFrame*, or *np.ndarray*) – Features

**Returns** Probability estimates

**Return type** *pd.Series*

**save** (*self*, *file\_path*, *pickle\_protocol=cloudpickle.DEFAULT\_PROTOCOL*)

Saves component at file path

**Parameters**

- **file\_path** (*str*) – Location to save file
- **pickle\_protocol** (*int*) – The pickle data stream format.

**Returns** None

## decision\_tree\_regressor

### Module Contents

### Classes Summary

---

*DecisionTreeRegressor*

---

Decision Tree Regressor.

---

## Contents

**class** evalml.pipelines.components.estimators.regressors.decision\_tree\_regressor.**DecisionTreeRegressor**

Decision Tree Regressor.

### Parameters

- **criterion** (*{ "mse", "friedman\_mse", "mae", "poisson" }*) – The function to measure the quality of a split. Supported criteria are:
  - “mse” for the mean squared error, which is equal to variance reduction as feature selection criterion and minimizes the L2 loss using the mean of each terminal node
  - “friedman\_mse”, which uses mean squared error with Friedman’s improvement score for potential splits
  - “mae” for the mean absolute error, which minimizes the L1 loss using the median of each terminal node,
  - “poisson” which uses reduction in Poisson deviance to find splits.
- **max\_features** (*int, float or { "auto", "sqrt", "log2" }*) – The number of features to consider when looking for the best split:
  - If int, then consider max\_features features at each split.
  - If float, then max\_features is a fraction and int(max\_features \* n\_features) features are considered at each split.
  - If “auto”, then max\_features=sqrt(n\_features).
  - If “sqrt”, then max\_features=sqrt(n\_features).
  - If “log2”, then max\_features=log2(n\_features).
  - If None, then max\_features = n\_features.

The search for a split does not stop until at least one valid partition of the node samples is found, even if it requires to effectively inspect more than max\_features features.

- **max\_depth** (*int*) – The maximum depth of the tree. Defaults to 6.
- **min\_samples\_split** (*int or float*) – The minimum number of samples required to split an internal node:
  - If int, then consider min\_samples\_split as the minimum number.
  - If float, then min\_samples\_split is a fraction and ceil(min\_samples\_split \* n\_samples) are the minimum number of samples for each split.

Defaults to 2.

- **min\_weight\_fraction\_leaf** (*float*) – The minimum weighted fraction of the sum total of weights (of all the input samples) required to be at a leaf node. Defaults to 0.0.
- **random\_seed** (*int*) – Seed for the random number generator. Defaults to 0.

## Attributes

<b>hyper-parameter_ranges</b>	{ "criterion": ["mse", "friedman_mse", "mae"], "max_features": ["auto", "sqrt", "log2"], "max_depth": Integer(4, 10), }
<b>model_family</b>	ModelFamily.DECISION_TREE
<b>modifies_features</b>	True
<b>modifies_target</b>	False
<b>name</b>	Decision Tree Regressor
<b>pre_dict_uses_y</b>	False
<b>supported_problem_types</b>	[ ProblemTypes.REGRESSION, ProblemTypes.TIME_SERIES_REGRESSION, ]

## Methods

<i>clone</i>	Constructs a new component with the same parameters and random state.
<i>default_parameters</i>	Returns the default parameters for this component.
<i>describe</i>	Describe a component and its parameters
<i>feature_importance</i>	Returns importance associated with each feature.
<i>fit</i>	Fits component to data
<i>load</i>	Loads component at file path
<i>needs_fitting</i>	Returns boolean determining if component needs fitting before
<i>parameters</i>	Returns the parameters which were used to initialize the component
<i>predict</i>	Make predictions using selected features.
<i>predict_proba</i>	Make probability estimates for labels.
<i>save</i>	Saves component at file path

### **clone** (*self*)

Constructs a new component with the same parameters and random state.

**Returns** A new instance of this component with identical parameters and random state.

### **default\_parameters** (*cls*)

Returns the default parameters for this component.

Our convention is that `Component.default_parameters == Component().parameters`.

**Returns** default parameters for this component.

**Return type** dict

### **describe** (*self*, *print\_name=False*, *return\_dict=False*)

Describe a component and its parameters

#### Parameters

- **print\_name** (*bool*, *optional*) – whether to print name of component
- **return\_dict** (*bool*, *optional*) – whether to return description as dictionary in the format {“name”: name, “parameters”: parameters}

**Returns** prints and returns dictionary

**Return type** None or dict

**property feature\_importance** (*self*)

Returns importance associated with each feature.

**Returns** Importance associated with each feature

**Return type** np.ndarray

**fit** (*self*, *X*, *y=None*)

Fits component to data

**Parameters**

- **X** (*list*, *pd.DataFrame* or *np.ndarray*) – The input training data of shape [n\_samples, n\_features]
- **y** (*list*, *pd.Series*, *np.ndarray*, *optional*) – The target training data of length [n\_samples]

**Returns** self

**static load** (*file\_path*)

Loads component at file path

**Parameters** **file\_path** (*str*) – Location to load file

**Returns** ComponentBase object

**needs\_fitting** (*self*)

Returns boolean determining if component needs fitting before calling predict, predict\_proba, transform, or feature\_importances. This can be overridden to False for components that do not need to be fit or whose fit methods do nothing.

**property parameters** (*self*)

Returns the parameters which were used to initialize the component

**predict** (*self*, *X*)

Make predictions using selected features.

**Parameters** **X** (*pd.DataFrame*, *np.ndarray*) – Data of shape [n\_samples, n\_features]

**Returns** Predicted values

**Return type** pd.Series

**predict\_proba** (*self*, *X*)

Make probability estimates for labels.

**Parameters** **X** (*pd.DataFrame*, or *np.ndarray*) – Features

**Returns** Probability estimates

**Return type** pd.Series

**save** (*self*, *file\_path*, *pickle\_protocol=cloudpickle.DEFAULT\_PROTOCOL*)

Saves component at file path

**Parameters**

- **file\_path** (*str*) – Location to save file
- **pickle\_protocol** (*int*) – The pickle data stream format.

**Returns** None

## elasticnet\_regressor

### Module Contents

### Classes Summary

---

<i>ElasticNetRegressor</i>	Elastic Net Regressor.
----------------------------	------------------------

---

### Contents

**class** evalml.pipelines.components.estimators.regressors.elasticnet\_regressor.**ElasticNetReg**

Elastic Net Regressor.

#### Parameters

- **alpha** (*float*) – Constant that multiplies the penalty terms. Defaults to 0.0001.
- **l1\_ratio** (*float*) – The mixing parameter, with  $0 \leq \text{l1\_ratio} \leq 1$ . Only used if `penalty='elasticnet'`. Setting `l1_ratio=0` is equivalent to using `penalty='l2'`, while setting `l1_ratio=1` is equivalent to using `penalty='l1'`. For  $0 < \text{l1\_ratio} < 1$ , the penalty is a combination of L1 and L2. Defaults to 0.15.
- **max\_iter** (*int*) – The maximum number of iterations. Defaults to 1000.
- **normalize** (*boolean*) – If True, the regressors will be normalized before regression by subtracting the mean and dividing by the l2-norm. Defaults to False.
- **random\_seed** (*int*) – Seed for the random number generator. Defaults to 0.

#### Attributes

hyper-parameter_ranges	{ "alpha": Real(0, 1), "l1_ratio": Real(0, 1), }
model_family	ModelFamily.LINEAR_MODEL
modifies_features	True
modifies_target	False
name	Elastic Net Regressor
pre-dict_uses_y	False
supported_problem_types	[ ProblemTypes.REGRESSION, ProblemTypes.TIME_SERIES_REGRESSION,]

#### Methods



<code>clone</code>	Constructs a new component with the same parameters and random state.
<code>default_parameters</code>	Returns the default parameters for this component.
<code>describe</code>	Describe a component and its parameters
<code>feature_importance</code>	Returns importance associated with each feature.
<code>fit</code>	Fits component to data
<code>load</code>	Loads component at file path
<code>needs_fitting</code>	Returns boolean determining if component needs fitting before
<code>parameters</code>	Returns the parameters which were used to initialize the component
<code>predict</code>	Make predictions using selected features.
<code>predict_proba</code>	Make probability estimates for labels.
<code>save</code>	Saves component at file path

**clone** (*self*)

Constructs a new component with the same parameters and random state.

**Returns** A new instance of this component with identical parameters and random state.

**default\_parameters** (*cls*)

Returns the default parameters for this component.

Our convention is that `Component.default_parameters == Component().parameters`.

**Returns** default parameters for this component.

**Return type** dict

**describe** (*self*, *print\_name=False*, *return\_dict=False*)

Describe a component and its parameters

**Parameters**

- **print\_name** (*bool*, *optional*) – whether to print name of component
- **return\_dict** (*bool*, *optional*) – whether to return description as dictionary in the format {"name": name, "parameters": parameters}

**Returns** prints and returns dictionary

**Return type** None or dict

**property feature\_importance** (*self*)

Returns importance associated with each feature.

**Returns** Importance associated with each feature

**Return type** np.ndarray

**fit** (*self*, *X*, *y=None*)

Fits component to data

**Parameters**

- **X** (*list*, *pd.DataFrame* or *np.ndarray*) – The input training data of shape [n\_samples, n\_features]
- **y** (*list*, *pd.Series*, *np.ndarray*, *optional*) – The target training data of length [n\_samples]

**Returns** self

**static load** (*file\_path*)

Loads component at file path

**Parameters** *file\_path* (*str*) – Location to load file

**Returns** ComponentBase object

**needs\_fitting** (*self*)

Returns boolean determining if component needs fitting before calling predict, predict\_proba, transform, or feature\_importances. This can be overridden to False for components that do not need to be fit or whose fit methods do nothing.

**property parameters** (*self*)

Returns the parameters which were used to initialize the component

**predict** (*self*, *X*)

Make predictions using selected features.

**Parameters** *X* (*pd.DataFrame*, *np.ndarray*) – Data of shape [n\_samples, n\_features]

**Returns** Predicted values

**Return type** *pd.Series*

**predict\_proba** (*self*, *X*)

Make probability estimates for labels.

**Parameters** *X* (*pd.DataFrame*, *or np.ndarray*) – Features

**Returns** Probability estimates

**Return type** *pd.Series*

**save** (*self*, *file\_path*, *pickle\_protocol=cloudpickle.DEFAULT\_PROTOCOL*)

Saves component at file path

**Parameters**

- **file\_path** (*str*) – Location to save file
- **pickle\_protocol** (*int*) – The pickle data stream format.

**Returns** None

**et\_regressor**

**Module Contents**

**Classes Summary**

---

*ExtraTreesRegressor*

Extra Trees Regressor.

---

## Contents

**class** evalml.pipelines.components.estimators.regressors.et\_regressor.**ExtraTreesRegressor** (*n\_*

Extra Trees Regressor.

### Parameters

- **n\_estimators** (*float*) – The number of trees in the forest. Defaults to 100.
- **max\_features** (*int, float or {"auto", "sqrt", "log2"}*) – The number of features to consider when looking for the best split:
  - If *int*, then consider *max\_features* features at each split.
  - If *float*, then *max\_features* is a fraction and  $\text{int}(\text{max\_features} * \text{n\_features})$  features are considered at each split.
  - If “auto”, then  $\text{max\_features} = \text{sqrt}(\text{n\_features})$ .
  - If “sqrt”, then  $\text{max\_features} = \text{sqrt}(\text{n\_features})$ .
  - If “log2”, then  $\text{max\_features} = \text{log2}(\text{n\_features})$ .
  - If *None*, then  $\text{max\_features} = \text{n\_features}$ .

The search for a split does not stop until at least one valid partition of the node samples is found, even if it requires to effectively inspect more than *max\_features* features. Defaults to “auto”.
- **max\_depth** (*int*) – The maximum depth of the tree. Defaults to 6.
- **min\_samples\_split** (*int or float*) – The minimum number of samples required to split an internal node:
  - If *int*, then consider *min\_samples\_split* as the minimum number.
  - If *float*, then *min\_samples\_split* is a fraction and  $\text{ceil}(\text{min\_samples\_split} * \text{n\_samples})$  are the minimum number of samples for each split.
- **to 2.** (*Defaults*) –
- **min\_weight\_fraction\_leaf** (*float*) – The minimum weighted fraction of the sum total of weights (of all the input samples) required to be at a leaf node. Defaults to 0.0.
- **n\_jobs** (*int or None*) – Number of jobs to run in parallel. -1 uses all processes. Defaults to -1.
- **random\_seed** (*int*) – Seed for the random number generator. Defaults to 0.

### Attributes

<b>hyper-parameter_ranges</b>	{ “n_estimators”: Integer(10, 1000), “max_features”: [“auto”, “sqrt”, “log2”], “max_depth”: Integer(4, 10),}
<b>model_family</b>	ModelFamily.EXTRA_TREES
<b>modifies_features</b>	True
<b>modifies_target</b>	False
<b>name</b>	Extra Trees Regressor
<b>pre-dict_uses_y</b>	False
<b>supported_problem_types</b>	[ ProblemTypes.REGRESSION, ProblemTypes.TIME_SERIES_REGRESSION,]

## Methods

<i>clone</i>	Constructs a new component with the same parameters and random state.
<i>default_parameters</i>	Returns the default parameters for this component.
<i>describe</i>	Describe a component and its parameters
<i>feature_importance</i>	Returns importance associated with each feature.
<i>fit</i>	Fits component to data
<i>load</i>	Loads component at file path
<i>needs_fitting</i>	Returns boolean determining if component needs fitting before
<i>parameters</i>	Returns the parameters which were used to initialize the component
<i>predict</i>	Make predictions using selected features.
<i>predict_proba</i>	Make probability estimates for labels.
<i>save</i>	Saves component at file path

### **clone** (*self*)

Constructs a new component with the same parameters and random state.

**Returns** A new instance of this component with identical parameters and random state.

### **default\_parameters** (*cls*)

Returns the default parameters for this component.

Our convention is that `Component.default_parameters == Component().parameters`.

**Returns** default parameters for this component.

**Return type** dict

### **describe** (*self*, *print\_name=False*, *return\_dict=False*)

Describe a component and its parameters

#### Parameters

- **print\_name** (*bool*, *optional*) – whether to print name of component
- **return\_dict** (*bool*, *optional*) – whether to return description as dictionary in the format {“name”: name, “parameters”: parameters}

**Returns** prints and returns dictionary

**Return type** None or dict

**property feature\_importance** (*self*)

Returns importance associated with each feature.

**Returns** Importance associated with each feature

**Return type** np.ndarray

**fit** (*self*, *X*, *y=None*)

Fits component to data

**Parameters**

- **X** (*list*, *pd.DataFrame* or *np.ndarray*) – The input training data of shape [n\_samples, n\_features]
- **y** (*list*, *pd.Series*, *np.ndarray*, *optional*) – The target training data of length [n\_samples]

**Returns** self

**static load** (*file\_path*)

Loads component at file path

**Parameters** **file\_path** (*str*) – Location to load file

**Returns** ComponentBase object

**needs\_fitting** (*self*)

Returns boolean determining if component needs fitting before calling predict, predict\_proba, transform, or feature\_importances. This can be overridden to False for components that do not need to be fit or whose fit methods do nothing.

**property parameters** (*self*)

Returns the parameters which were used to initialize the component

**predict** (*self*, *X*)

Make predictions using selected features.

**Parameters** **X** (*pd.DataFrame*, *np.ndarray*) – Data of shape [n\_samples, n\_features]

**Returns** Predicted values

**Return type** pd.Series

**predict\_proba** (*self*, *X*)

Make probability estimates for labels.

**Parameters** **X** (*pd.DataFrame*, or *np.ndarray*) – Features

**Returns** Probability estimates

**Return type** pd.Series

**save** (*self*, *file\_path*, *pickle\_protocol=cloudpickle.DEFAULT\_PROTOCOL*)

Saves component at file path

**Parameters**

- **file\_path** (*str*) – Location to save file
- **pickle\_protocol** (*int*) – The pickle data stream format.

**Returns** None

## lightgbm\_regressor

### Module Contents

### Classes Summary

---

<i>LightGBMRegressor</i>	LightGBM Regressor.
--------------------------	---------------------

---

### Contents

**class** evalml.pipelines.components.estimators.regressors.lightgbm\_regressor.**LightGBMRegressor**

LightGBM Regressor.

#### Parameters

- **boosting\_type** (*string*) – Type of boosting to use. Defaults to “gbdt”. - ‘gbdt’ uses traditional Gradient Boosting Decision Tree - “dart”, uses Dropouts meet Multiple Additive Regression Trees - “goss”, uses Gradient-based One-Side Sampling - “rf”, uses Random Forest
- **learning\_rate** (*float*) – Boosting learning rate. Defaults to 0.1.
- **n\_estimators** (*int*) – Number of boosted trees to fit. Defaults to 100.
- **max\_depth** (*int*) – Maximum tree depth for base learners, <=0 means no limit. Defaults to 0.
- **num\_leaves** (*int*) – Maximum tree leaves for base learners. Defaults to 31.
- **min\_child\_samples** (*int*) – Minimum number of data needed in a child (leaf). Defaults to 20.
- **bagging\_fraction** (*float*) – LightGBM will randomly select a subset of features on each iteration (tree) without resampling if this is smaller than 1.0. For example, if set to 0.8, LightGBM will select 80% of features before training each tree. This can be used to speed up training and deal with overfitting. Defaults to 0.9.
- **bagging\_freq** (*int*) – Frequency for bagging. 0 means bagging is disabled. k means perform bagging at every k iteration. Every k-th iteration, LightGBM will randomly select

bagging\_fraction \* 100 % of the data to use for the next k iterations. Defaults to 0.

- **n\_jobs** (*int* or *None*) – Number of threads to run in parallel. -1 uses all threads. Defaults to -1.
- **random\_seed** (*int*) – Seed for the random number generator. Defaults to 0.

#### Attributes

<b>hyper-parameter_ranges</b>	{ "learning_rate": Real(0.000001, 1), "boosting_type": ["gbdt", "dart", "goss", "rf"], "n_estimators": Integer(10, 100), "max_depth": Integer(0, 10), "num_leaves": Integer(2, 100), "min_child_samples": Integer(1, 100), "bagging_fraction": Real(0.000001, 1), "bagging_freq": Integer(0, 1), }
<b>model_family</b>	ModelFamily.LIGHTGBM
<b>modifies_features</b>	True
<b>modifies_target</b>	False
<b>name</b>	LightGBM Regressor
<b>pre-dict_uses_y</b>	False
<b>SEED_MAX</b>	SEED_BOUNDS.max_bound
<b>SEED_MIN</b>	0
<b>supported_problem_types</b>	[ProblemTypes.REGRESSION]

#### Methods

<i>clone</i>	Constructs a new component with the same parameters and random state.
<i>default_parameters</i>	Returns the default parameters for this component.
<i>describe</i>	Describe a component and its parameters
<i>feature_importance</i>	Returns importance associated with each feature.
<i>fit</i>	Fits component to data
<i>load</i>	Loads component at file path
<i>needs_fitting</i>	Returns boolean determining if component needs fitting before
<i>parameters</i>	Returns the parameters which were used to initialize the component
<i>predict</i>	Make predictions using selected features.
<i>predict_proba</i>	Make probability estimates for labels.
<i>save</i>	Saves component at file path

#### **clone** (*self*)

Constructs a new component with the same parameters and random state.

**Returns** A new instance of this component with identical parameters and random state.

#### **default\_parameters** (*cls*)

Returns the default parameters for this component.

Our convention is that `Component.default_parameters == Component().parameters`.

**Returns** default parameters for this component.

**Return type** dict

**describe** (*self*, *print\_name=False*, *return\_dict=False*)

Describe a component and its parameters

**Parameters**

- **print\_name** (*bool*, *optional*) – whether to print name of component
- **return\_dict** (*bool*, *optional*) – whether to return description as dictionary in the format {"name": name, "parameters": parameters}

**Returns** prints and returns dictionary

**Return type** None or dict

**property feature\_importance** (*self*)

Returns importance associated with each feature.

**Returns** Importance associated with each feature

**Return type** np.ndarray

**fit** (*self*, *X*, *y=None*)

Fits component to data

**Parameters**

- **X** (*list*, *pd.DataFrame* or *np.ndarray*) – The input training data of shape [n\_samples, n\_features]
- **y** (*list*, *pd.Series*, *np.ndarray*, *optional*) – The target training data of length [n\_samples]

**Returns** self

**static load** (*file\_path*)

Loads component at file path

**Parameters** **file\_path** (*str*) – Location to load file

**Returns** ComponentBase object

**needs\_fitting** (*self*)

Returns boolean determining if component needs fitting before calling predict, predict\_proba, transform, or feature\_importances. This can be overridden to False for components that do not need to be fit or whose fit methods do nothing.

**property parameters** (*self*)

Returns the parameters which were used to initialize the component

**predict** (*self*, *X*)

Make predictions using selected features.

**Parameters** **X** (*pd.DataFrame*, *np.ndarray*) – Data of shape [n\_samples, n\_features]

**Returns** Predicted values

**Return type** pd.Series

**predict\_proba** (*self*, *X*)

Make probability estimates for labels.

**Parameters** **X** (*pd.DataFrame*, or *np.ndarray*) – Features

**Returns** Probability estimates

**Return type** pd.Series



**save** (*self*, *file\_path*, *pickle\_protocol*=*cloudpickle.DEFAULT\_PROTOCOL*)

Saves component at file path

#### Parameters

- **file\_path** (*str*) – Location to save file
- **pickle\_protocol** (*int*) – The pickle data stream format.

**Returns** None

## linear\_regressor

### Module Contents

### Classes Summary

---

*LinearRegressor*

Linear Regressor.

---

### Contents

**class** evalml.pipelines.components.estimators.regressors.linear\_regressor.**LinearRegressor** (*fit*

Linear Regressor.

#### Parameters

- **fit\_intercept** (*boolean*) – Whether to calculate the intercept for this model. If set to False, no intercept will be used in calculations (i.e. data is expected to be centered). Defaults to True.
- **normalize** (*boolean*) – If True, the regressors will be normalized before regression by subtracting the mean and dividing by the l2-norm. This parameter is ignored when *fit\_intercept* is set to False. Defaults to False.
- **n\_jobs** (*int or None*) – Number of jobs to run in parallel. -1 uses all threads. Defaults to -1.
- **random\_seed** (*int*) – Seed for the random number generator. Defaults to 0.

#### Attributes

<b>hyper-parameter_ranges</b>	{ "fit_intercept": [True, False], "normalize": [True, False]}
<b>model_family</b>	ModelFamily.LINEAR_MODEL
<b>modifies_features</b>	True
<b>modifies_target</b>	False
<b>name</b>	Linear Regressor
<b>pre-dict_uses_y</b>	False
<b>supported_problem_types</b>	[ ProblemTypes.REGRESSION, ProblemTypes.TIME_SERIES_REGRESSION,]

## Methods

<i>clone</i>	Constructs a new component with the same parameters and random state.
<i>default_parameters</i>	Returns the default parameters for this component.
<i>describe</i>	Describe a component and its parameters
<i>feature_importance</i>	Returns importance associated with each feature.
<i>fit</i>	Fits component to data
<i>load</i>	Loads component at file path
<i>needs_fitting</i>	Returns boolean determining if component needs fitting before
<i>parameters</i>	Returns the parameters which were used to initialize the component
<i>predict</i>	Make predictions using selected features.
<i>predict_proba</i>	Make probability estimates for labels.
<i>save</i>	Saves component at file path

### **clone** (*self*)

Constructs a new component with the same parameters and random state.

**Returns** A new instance of this component with identical parameters and random state.

### **default\_parameters** (*cls*)

Returns the default parameters for this component.

Our convention is that `Component.default_parameters == Component().parameters`.

**Returns** default parameters for this component.

**Return type** dict

### **describe** (*self*, *print\_name=False*, *return\_dict=False*)

Describe a component and its parameters

#### Parameters

- **print\_name** (*bool*, *optional*) – whether to print name of component
- **return\_dict** (*bool*, *optional*) – whether to return description as dictionary in the format {"name": name, "parameters": parameters}

**Returns** prints and returns dictionary

**Return type** None or dict

**property feature\_importance** (*self*)

Returns importance associated with each feature.

**Returns** Importance associated with each feature

**Return type** np.ndarray

**fit** (*self*, *X*, *y=None*)

Fits component to data

**Parameters**

- **X** (*list*, *pd.DataFrame* or *np.ndarray*) – The input training data of shape [n\_samples, n\_features]
- **y** (*list*, *pd.Series*, *np.ndarray*, *optional*) – The target training data of length [n\_samples]

**Returns** self

**static load** (*file\_path*)

Loads component at file path

**Parameters** **file\_path** (*str*) – Location to load file

**Returns** ComponentBase object

**needs\_fitting** (*self*)

Returns boolean determining if component needs fitting before calling predict, predict\_proba, transform, or feature\_importances. This can be overridden to False for components that do not need to be fit or whose fit methods do nothing.

**property parameters** (*self*)

Returns the parameters which were used to initialize the component

**predict** (*self*, *X*)

Make predictions using selected features.

**Parameters** **X** (*pd.DataFrame*, *np.ndarray*) – Data of shape [n\_samples, n\_features]

**Returns** Predicted values

**Return type** pd.Series

**predict\_proba** (*self*, *X*)

Make probability estimates for labels.

**Parameters** **X** (*pd.DataFrame*, or *np.ndarray*) – Features

**Returns** Probability estimates

**Return type** pd.Series

**save** (*self*, *file\_path*, *pickle\_protocol=cloudpickle.DEFAULT\_PROTOCOL*)

Saves component at file path

**Parameters**

- **file\_path** (*str*) – Location to save file
- **pickle\_protocol** (*int*) – The pickle data stream format.

**Returns** None

## prophet\_regressor

### Module Contents

### Classes Summary

---

<i>ProphetRegressor</i>	Prophet is a procedure for forecasting time series data based on an additive model where non-linear trends are fit with yearly, weekly, and daily seasonality, plus holiday effects.
-------------------------	--

---

### Contents

**class** evalml.pipelines.components.estimators.regressors.prophet\_regressor.**ProphetRegressor**

Prophet is a procedure for forecasting time series data based on an additive model where non-linear trends are fit with yearly, weekly, and daily seasonality, plus holiday effects. It works best with time series that have strong seasonal effects and several seasons of historical data. Prophet is robust to missing data and shifts in the trend, and typically handles outliers well.

More information here: <https://facebook.github.io/prophet/>

#### Attributes

<b>hyper-parameter_ranges</b>	{ "changepoint_prior_scale": Real(0.001, 0.5), "seasonality_prior_scale": Real(0.01, 10), "holidays_prior_scale": Real(0.01, 10), "seasonality_mode": ["additive", "multiplicative"], }
<b>model_family</b>	ModelFamily.PROPHET
<b>modifies_features</b>	True
<b>modifies_target</b>	False
<b>name</b>	Prophet Regressor
<b>pre-dict_uses_y</b>	False
<b>supported_problem_types</b>	[ProblemTypes.TIME_SERIES_REGRESSION]

## Methods

<i>build_prophet_df</i>	
<i>clone</i>	Constructs a new component with the same parameters and random state.
<i>default_parameters</i>	Returns the default parameters for this component.
<i>describe</i>	Describe a component and its parameters
<i>feature_importance</i>	Returns array of 0's with len(1) as feature_importance is not defined for Prophet regressor.
<i>fit</i>	Fits component to data
<i>get_params</i>	
<i>load</i>	Loads component at file path
<i>needs_fitting</i>	Returns boolean determining if component needs fitting before
<i>parameters</i>	Returns the parameters which were used to initialize the component
<i>predict</i>	Make predictions using selected features.
<i>predict_proba</i>	Make probability estimates for labels.
<i>save</i>	Saves component at file path

**static build\_prophet\_df** (*X*, *y=None*, *date\_column='ds'*)

**clone** (*self*)

Constructs a new component with the same parameters and random state.

**Returns** A new instance of this component with identical parameters and random state.

**default\_parameters** (*cls*)

Returns the default parameters for this component.

Our convention is that `Component.default_parameters == Component().parameters`.

**Returns** default parameters for this component.

**Return type** dict

**describe** (*self*, *print\_name=False*, *return\_dict=False*)

Describe a component and its parameters

**Parameters**

- **print\_name** (*bool, optional*) – whether to print name of component
- **return\_dict** (*bool, optional*) – whether to return description as dictionary in the format {"name": name, "parameters": parameters}

**Returns** prints and returns dictionary

**Return type** None or dict

**property feature\_importance** (*self*)

Returns array of 0's with len(1) as feature\_importance is not defined for Prophet regressor.

**fit** (*self, X, y=None*)

Fits component to data

**Parameters**

- **X** (*list, pd.DataFrame or np.ndarray*) – The input training data of shape [n\_samples, n\_features]
- **y** (*list, pd.Series, np.ndarray, optional*) – The target training data of length [n\_samples]

**Returns** self

**get\_params** (*self*)

**static load** (*file\_path*)

Loads component at file path

**Parameters** **file\_path** (*str*) – Location to load file

**Returns** ComponentBase object

**needs\_fitting** (*self*)

Returns boolean determining if component needs fitting before calling predict, predict\_proba, transform, or feature\_importances. This can be overridden to False for components that do not need to be fit or whose fit methods do nothing.

**property parameters** (*self*)

Returns the parameters which were used to initialize the component

**predict** (*self, X, y=None*)

Make predictions using selected features.

**Parameters** **X** (*pd.DataFrame, np.ndarray*) – Data of shape [n\_samples, n\_features]

**Returns** Predicted values

**Return type** pd.Series

**predict\_proba** (*self, X*)

Make probability estimates for labels.

**Parameters** **X** (*pd.DataFrame, or np.ndarray*) – Features

**Returns** Probability estimates

**Return type** pd.Series

**save** (*self, file\_path, pickle\_protocol=cloudpickle.DEFAULT\_PROTOCOL*)

Saves component at file path

**Parameters**

- **file\_path** (*str*) – Location to save file

- **pickle\_protocol** (*int*) – The pickle data stream format.

**Returns** None

## rf\_regressor

### Module Contents

### Classes Summary

---

*RandomForestRegressor*

Random Forest Regressor.

---

### Contents

**class** evalml.pipelines.components.estimators.regressors.rf\_regressor.**RandomForestRegressor**

Random Forest Regressor.

#### Parameters

- **n\_estimators** (*float*) – The number of trees in the forest. Defaults to 100.
- **max\_depth** (*int*) – Maximum tree depth for base learners. Defaults to 6.
- **n\_jobs** (*int or None*) – Number of jobs to run in parallel. -1 uses all processes. Defaults to -1.
- **random\_seed** (*int*) – Seed for the random number generator. Defaults to 0.

#### Attributes

<b>hyper-parameter_ranges</b>	{ “n_estimators”: Integer(10, 1000), “max_depth”: Integer(1, 32), }
<b>model_family</b>	ModelFamily.RANDOM_FOREST
<b>modifies_features</b>	True
<b>modifies_target</b>	False
<b>name</b>	Random Forest Regressor
<b>predict_uses_y</b>	False
<b>supported_problem_types</b>	[ ProblemTypes.REGRESSION, ProblemTypes.TIME_SERIES_REGRESSION, ]

#### Methods

<code>clone</code>	Constructs a new component with the same parameters and random state.
<code>default_parameters</code>	Returns the default parameters for this component.
<code>describe</code>	Describe a component and its parameters
<code>feature_importance</code>	Returns importance associated with each feature.
<code>fit</code>	Fits component to data
<code>load</code>	Loads component at file path
<code>needs_fitting</code>	Returns boolean determining if component needs fitting before
<code>parameters</code>	Returns the parameters which were used to initialize the component
<code>predict</code>	Make predictions using selected features.
<code>predict_proba</code>	Make probability estimates for labels.
<code>save</code>	Saves component at file path

**clone** (*self*)

Constructs a new component with the same parameters and random state.

**Returns** A new instance of this component with identical parameters and random state.

**default\_parameters** (*cls*)

Returns the default parameters for this component.

Our convention is that `Component.default_parameters == Component().parameters`.

**Returns** default parameters for this component.

**Return type** dict

**describe** (*self*, *print\_name=False*, *return\_dict=False*)

Describe a component and its parameters

**Parameters**

- **print\_name** (*bool*, *optional*) – whether to print name of component
- **return\_dict** (*bool*, *optional*) – whether to return description as dictionary in the format {"name": name, "parameters": parameters}

**Returns** prints and returns dictionary

**Return type** None or dict

**property feature\_importance** (*self*)

Returns importance associated with each feature.

**Returns** Importance associated with each feature

**Return type** np.ndarray

**fit** (*self*, *X*, *y=None*)

Fits component to data

**Parameters**

- **X** (*list*, *pd.DataFrame* or *np.ndarray*) – The input training data of shape [n\_samples, n\_features]
- **y** (*list*, *pd.Series*, *np.ndarray*, *optional*) – The target training data of length [n\_samples]

**Returns** self



**static load** (*file\_path*)

Loads component at file path

**Parameters** *file\_path* (*str*) – Location to load file

**Returns** ComponentBase object

**needs\_fitting** (*self*)

Returns boolean determining if component needs fitting before calling predict, predict\_proba, transform, or feature\_importances. This can be overridden to False for components that do not need to be fit or whose fit methods do nothing.

**property parameters** (*self*)

Returns the parameters which were used to initialize the component

**predict** (*self*, *X*)

Make predictions using selected features.

**Parameters** *X* (*pd.DataFrame*, *np.ndarray*) – Data of shape [n\_samples, n\_features]

**Returns** Predicted values

**Return type** *pd.Series*

**predict\_proba** (*self*, *X*)

Make probability estimates for labels.

**Parameters** *X* (*pd.DataFrame*, or *np.ndarray*) – Features

**Returns** Probability estimates

**Return type** *pd.Series*

**save** (*self*, *file\_path*, *pickle\_protocol=cloudpickle.DEFAULT\_PROTOCOL*)

Saves component at file path

**Parameters**

- **file\_path** (*str*) – Location to save file
- **pickle\_protocol** (*int*) – The pickle data stream format.

**Returns** None

## svm\_regressor

### Module Contents

### Classes Summary

---

*SVMRegressor*

---

Support Vector Machine Regressor.

---

## Contents

```
class evalml.pipelines.components.estimators.regressors.svm_regressor.SVMRegressor (C=1.0,
                                                                    kernel='rbf',
                                                                    gamma='scale',
                                                                    random_seed=None,
                                                                    **kwargs)
```

Support Vector Machine Regressor.

### Parameters

- **C** (*float*) – The regularization parameter. The strength of the regularization is inversely proportional to C. Must be strictly positive. The penalty is a squared l2 penalty. Defaults to 1.0.
- **kernel** (*{ "linear", "poly", "rbf", "sigmoid", "precomputed" }*) – Specifies the kernel type to be used in the algorithm. Defaults to “rbf”.
- **gamma** (*{ "scale", "auto" } or float*) – Kernel coefficient for “rbf”, “poly” and “sigmoid”. Defaults to “scale”. - If gamma=’scale’ (default) is passed then it uses  $1 / (n\_features * X.var())$  as value of gamma - if “auto”, uses  $1 / n\_features$
- **random\_seed** (*int*) – Seed for the random number generator. Defaults to 0.

### Attributes

<b>hyper-parameter_ranges</b>	{ “C”: Real(0, 10), “kernel”: [“linear”, “poly”, “rbf”, “sigmoid”, “precomputed”], “gamma”: [“scale”, “auto”], }
<b>model_family</b>	ModelFamily.SVM
<b>modifies_features</b>	True
<b>modifies_target</b>	False
<b>name</b>	SVM Regressor
<b>pre_dict_uses_y</b>	False
<b>supported_problem_types</b>	[ ProblemTypes.REGRESSION, ProblemTypes.TIME_SERIES_REGRESSION, ]

### Methods

<i>clone</i>	Constructs a new component with the same parameters and random state.
<i>default_parameters</i>	Returns the default parameters for this component.
<i>describe</i>	Describe a component and its parameters
<i>feature_importance</i>	Feature importance only works with linear kernels.
<i>fit</i>	Fits component to data
<i>load</i>	Loads component at file path
<i>needs_fitting</i>	Returns boolean determining if component needs fitting before
<i>parameters</i>	Returns the parameters which were used to initialize the component

continues on next page

Table 355 – continued from previous page

<code>predict</code>	Make predictions using selected features.
<code>predict_proba</code>	Make probability estimates for labels.
<code>save</code>	Saves component at file path

**clone** (*self*)

Constructs a new component with the same parameters and random state.

**Returns** A new instance of this component with identical parameters and random state.

**default\_parameters** (*cls*)

Returns the default parameters for this component.

Our convention is that `Component.default_parameters == Component().parameters`.

**Returns** default parameters for this component.

**Return type** dict

**describe** (*self*, *print\_name=False*, *return\_dict=False*)

Describe a component and its parameters

**Parameters**

- **print\_name** (*bool*, *optional*) – whether to print name of component
- **return\_dict** (*bool*, *optional*) – whether to return description as dictionary in the format {"name": name, "parameters": parameters}

**Returns** prints and returns dictionary

**Return type** None or dict

**property feature\_importance** (*self*)

Feature importance only works with linear kernels. If the kernel isn't linear, we return a numpy array of zeros

**fit** (*self*, *X*, *y=None*)

Fits component to data

**Parameters**

- **X** (*list*, *pd.DataFrame* or *np.ndarray*) – The input training data of shape [n\_samples, n\_features]
- **y** (*list*, *pd.Series*, *np.ndarray*, *optional*) – The target training data of length [n\_samples]

**Returns** self

**static load** (*file\_path*)

Loads component at file path

**Parameters** **file\_path** (*str*) – Location to load file

**Returns** ComponentBase object

**needs\_fitting** (*self*)

Returns boolean determining if component needs fitting before calling predict, predict\_proba, transform, or feature\_importances. This can be overridden to False for components that do not need to be fit or whose fit methods do nothing.

**property parameters** (*self*)

Returns the parameters which were used to initialize the component

**predict** (*self*, *X*)

Make predictions using selected features.

**Parameters** **X** (*pd.DataFrame*, *np.ndarray*) – Data of shape [n\_samples, n\_features]

**Returns** Predicted values

**Return type** *pd.Series*

**predict\_proba** (*self*, *X*)

Make probability estimates for labels.

**Parameters** **X** (*pd.DataFrame*, *or np.ndarray*) – Features

**Returns** Probability estimates

**Return type** *pd.Series*

**save** (*self*, *file\_path*, *pickle\_protocol=cloudpickle.DEFAULT\_PROTOCOL*)

Saves component at file path

**Parameters**

- **file\_path** (*str*) – Location to save file
- **pickle\_protocol** (*int*) – The pickle data stream format.

**Returns** None

## time\_series\_baseline\_estimator

### Module Contents

### Classes Summary

---

<i>TimeSeriesBaselineEstimator</i>	Time series estimator that predicts using the naive forecasting approach.
------------------------------------	---

---

### Contents

**class** evalml.pipelines.components.estimators.regressors.time\_series\_baseline\_estimator.**TimeSeriesBaselineEstimator**

Time series estimator that predicts using the naive forecasting approach.

This is useful as a simple baseline estimator for time series problems.

**Parameters**

- **gap** (*int*) – Gap between prediction date and target date and must be a positive integer. If gap is 0, target date will be shifted ahead by 1 time period. Defaults to 1.
- **random\_seed** (*int*) – Seed for the random number generator. Defaults to 0.

**Attributes**

<b>hyper-parameter_ranges</b>	{}
<b>model_family</b>	ModelFamily.BASELINE
<b>modifies_features</b>	True
<b>modifies_target</b>	False
<b>name</b>	Time Series Baseline Estimator
<b>pre-dict_uses_y</b>	True
<b>supported_problem_types</b>	[ProblemTypes.TIME_SERIES_REGRESSION, ProblemTypes.TIME_SERIES_BINARY, ProblemTypes.TIME_SERIES_MULTICLASS,]

## Methods

<i>clone</i>	Constructs a new component with the same parameters and random state.
<i>default_parameters</i>	Returns the default parameters for this component.
<i>describe</i>	Describe a component and its parameters
<i>feature_importance</i>	Returns importance associated with each feature.
<i>fit</i>	Fits component to data
<i>load</i>	Loads component at file path
<i>needs_fitting</i>	Returns boolean determining if component needs fitting before
<i>parameters</i>	Returns the parameters which were used to initialize the component
<i>predict</i>	Make predictions using selected features.
<i>predict_proba</i>	Make probability estimates for labels.
<i>save</i>	Saves component at file path

### **clone** (*self*)

Constructs a new component with the same parameters and random state.

**Returns** A new instance of this component with identical parameters and random state.

### **default\_parameters** (*cls*)

Returns the default parameters for this component.

Our convention is that `Component.default_parameters == Component().parameters`.

**Returns** default parameters for this component.

**Return type** dict

### **describe** (*self*, *print\_name=False*, *return\_dict=False*)

Describe a component and its parameters

#### Parameters

- **print\_name** (*bool*, *optional*) – whether to print name of component
- **return\_dict** (*bool*, *optional*) – whether to return description as dictionary in the format {"name": name, "parameters": parameters}

**Returns** prints and returns dictionary

**Return type** None or dict

**property feature\_importance** (*self*)

Returns importance associated with each feature.

Since baseline estimators do not use input features to calculate predictions, returns an array of zeroes.

**Returns** an array of zeroes

**Return type** np.ndarray (float)

**fit** (*self*, *X*, *y=None*)

Fits component to data

**Parameters**

- **X** (*list*, *pd.DataFrame* or *np.ndarray*) – The input training data of shape [n\_samples, n\_features]
- **y** (*list*, *pd.Series*, *np.ndarray*, *optional*) – The target training data of length [n\_samples]

**Returns** self

**static load** (*file\_path*)

Loads component at file path

**Parameters** **file\_path** (*str*) – Location to load file

**Returns** ComponentBase object

**needs\_fitting** (*self*)

Returns boolean determining if component needs fitting before calling predict, predict\_proba, transform, or feature\_importances. This can be overridden to False for components that do not need to be fit or whose fit methods do nothing.

**property parameters** (*self*)

Returns the parameters which were used to initialize the component

**predict** (*self*, *X*, *y=None*)

Make predictions using selected features.

**Parameters** **X** (*pd.DataFrame*, *np.ndarray*) – Data of shape [n\_samples, n\_features]

**Returns** Predicted values

**Return type** pd.Series

**predict\_proba** (*self*, *X*, *y=None*)

Make probability estimates for labels.

**Parameters** **X** (*pd.DataFrame*, or *np.ndarray*) – Features

**Returns** Probability estimates

**Return type** pd.Series

**save** (*self*, *file\_path*, *pickle\_protocol=cloudpickle.DEFAULT\_PROTOCOL*)

Saves component at file path

**Parameters**

- **file\_path** (*str*) – Location to save file
- **pickle\_protocol** (*int*) – The pickle data stream format.

**Returns** None

## xgboost\_regressor

### Module Contents

### Classes Summary

---

<i>XGBoostRegressor</i>	XGBoost Regressor.
-------------------------	--------------------

---

### Contents

**class** evalml.pipelines.components.estimators.regressors.xgboost\_regressor.XGBoostRegressor

XGBoost Regressor.

#### Parameters

- **eta** (*float*) – Boosting learning rate. Defaults to 0.1.
- **max\_depth** (*int*) – Maximum tree depth for base learners. Defaults to 6.
- **min\_child\_weight** (*float*) – Minimum sum of instance weight (hessian) needed in a child. Defaults to 1.0
- **n\_estimators** (*int*) – Number of gradient boosted trees. Equivalent to number of boosting rounds. Defaults to 100.
- **random\_seed** (*int*) – Seed for the random number generator. Defaults to 0.
- **n\_jobs** (*int*) – Number of parallel threads used to run xgboost. Note that creating thread contention will significantly slow down the algorithm. Defaults to -1.

#### Attributes

<b>hyper-parameter_ranges</b>	{ “eta”: Real(0.000001, 1), “max_depth”: Integer(1, 20), “min_child_weight”: Real(1, 10), “n_estimators”: Integer(1, 1000), }
<b>model_family</b>	ModelFamily.XGBOOST
<b>modifies_features</b>	True
<b>modifies_target</b>	False
<b>name</b>	XGBoost Regressor
<b>pre-dict_uses_y</b>	False
<b>SEED_MAX</b>	None
<b>SEED_MIN</b>	None
<b>supported_problem_types</b>	[ ProblemTypes.REGRESSION, ProblemTypes.TIME_SERIES_REGRESSION, ]

## Methods

<i>clone</i>	Constructs a new component with the same parameters and random state.
<i>default_parameters</i>	Returns the default parameters for this component.
<i>describe</i>	Describe a component and its parameters
<i>feature_importance</i>	Returns importance associated with each feature.
<i>fit</i>	Fits component to data
<i>load</i>	Loads component at file path
<i>needs_fitting</i>	Returns boolean determining if component needs fitting before
<i>parameters</i>	Returns the parameters which were used to initialize the component
<i>predict</i>	Make predictions using selected features.
<i>predict_proba</i>	Make probability estimates for labels.
<i>save</i>	Saves component at file path

### **clone** (*self*)

Constructs a new component with the same parameters and random state.

**Returns** A new instance of this component with identical parameters and random state.

### **default\_parameters** (*cls*)

Returns the default parameters for this component.

Our convention is that `Component.default_parameters == Component().parameters`.

**Returns** default parameters for this component.

**Return type** dict

### **describe** (*self*, *print\_name=False*, *return\_dict=False*)

Describe a component and its parameters

#### Parameters

- **print\_name** (*bool*, *optional*) – whether to print name of component
- **return\_dict** (*bool*, *optional*) – whether to return description as dictionary in the format {“name”: name, “parameters”: parameters}



**Returns** prints and returns dictionary

**Return type** None or dict

**property feature\_importance** (*self*)

Returns importance associated with each feature.

**Returns** Importance associated with each feature

**Return type** np.ndarray

**fit** (*self*, *X*, *y=None*)

Fits component to data

**Parameters**

- **X** (*list*, *pd.DataFrame* or *np.ndarray*) – The input training data of shape [n\_samples, n\_features]
- **y** (*list*, *pd.Series*, *np.ndarray*, *optional*) – The target training data of length [n\_samples]

**Returns** self

**static load** (*file\_path*)

Loads component at file path

**Parameters** **file\_path** (*str*) – Location to load file

**Returns** ComponentBase object

**needs\_fitting** (*self*)

Returns boolean determining if component needs fitting before calling predict, predict\_proba, transform, or feature\_importances. This can be overridden to False for components that do not need to be fit or whose fit methods do nothing.

**property parameters** (*self*)

Returns the parameters which were used to initialize the component

**predict** (*self*, *X*)

Make predictions using selected features.

**Parameters** **X** (*pd.DataFrame*, *np.ndarray*) – Data of shape [n\_samples, n\_features]

**Returns** Predicted values

**Return type** pd.Series

**predict\_proba** (*self*, *X*)

Make probability estimates for labels.

**Parameters** **X** (*pd.DataFrame*, or *np.ndarray*) – Features

**Returns** Probability estimates

**Return type** pd.Series

**save** (*self*, *file\_path*, *pickle\_protocol=cloudpickle.DEFAULT\_PROTOCOL*)

Saves component at file path

**Parameters**

- **file\_path** (*str*) – Location to save file
- **pickle\_protocol** (*int*) – The pickle data stream format.

**Returns** None

## Package Contents

### Classes Summary

<i>ARIMAREgressor</i>	Autoregressive Integrated Moving Average Model.
<i>BaselineRegressor</i>	Baseline regressor that uses a simple strategy to make predictions.
<i>CatBoostRegressor</i>	CatBoost Regressor, a regressor that uses gradient-boosting on decision trees.
<i>DecisionTreeRegressor</i>	Decision Tree Regressor.
<i>ElasticNetRegressor</i>	Elastic Net Regressor.
<i>ExtraTreesRegressor</i>	Extra Trees Regressor.
<i>LightGBMRegressor</i>	LightGBM Regressor.
<i>LinearRegressor</i>	Linear Regressor.
<i>ProphetRegressor</i>	Prophet is a procedure for forecasting time series data based on an additive model where non-linear trends are fit with yearly, weekly, and daily seasonality, plus holiday effects.
<i>RandomForestRegressor</i>	Random Forest Regressor.
<i>SVMRegressor</i>	Support Vector Machine Regressor.
<i>TimeSeriesBaselineEstimator</i>	Time series estimator that predicts using the naive forecasting approach.
<i>XGBoostRegressor</i>	XGBoost Regressor.

### Contents

```
class evalml.pipelines.components.estimators.regressors.ARIMAREgressor (date_index=None,
                                                                    trend=None,
                                                                    start_p=2,
                                                                    d=0,
                                                                    start_q=2,
                                                                    max_p=5,
                                                                    max_d=2,
                                                                    max_q=5,
                                                                    seasonal=True,
                                                                    n_jobs=-1,
                                                                    random_seed=0,
                                                                    **kwargs)
```

Autoregressive Integrated Moving Average Model. The three parameters (p, d, q) are the AR order, the degree of differencing, and the MA order. More information here: [https://www.statsmodels.org/devel/generated/statsmodels.tsa.arima\\_model.ARIMA.html](https://www.statsmodels.org/devel/generated/statsmodels.tsa.arima_model.ARIMA.html)

Currently ARIMAREgressor isn't supported via conda install. It's recommended that it be installed via PyPI.

#### Parameters

- **date\_index** (*str*) – Specifies the name of the column in X that provides the datetime objects. Defaults to None.
- **trend** (*str*) – Controls the deterministic trend. Options are ['n', 'c', 't', 'ct'] where 'c'

is a constant term, ‘t’ indicates a linear trend, and ‘ct’ is both. Can also be an iterable when defining a polynomial, such as [1, 1, 0, 1].

- **start\_p**(*int*) – Minimum Autoregressive order. Defaults to 2.
- **d**(*int*) – Minimum Differencing degree. Defaults to 0.
- **start\_q**(*int*) – Minimum Moving Average order. Defaults to 2.
- **max\_p**(*int*) – Maximum Autoregressive order. Defaults to 5.
- **max\_d**(*int*) – Maximum Differencing degree. Defaults to 2.
- **max\_q**(*int*) – Maximum Moving Average order. Defaults to 5.
- **seasonal**(*boolean*) – Whether to fit a seasonal model to ARIMA. Defaults to True.
- **n\_jobs**(*int or None*) – Non-negative integer describing level of parallelism used for pipelines. Defaults to -1.
- **random\_seed**(*int*) – Seed for the random number generator. Defaults to 0.

#### Attributes

<b>hyper-parameter_ranges</b>	{ “start_p”: Integer(1, 3), “d”: Integer(0, 2), “start_q”: Integer(1, 3), “max_p”: Integer(3, 10), “max_d”: Integer(2, 5), “max_q”: Integer(3, 10), “seasonal”: [True, False], }
<b>model_family</b>	ModelFamily.ARIMA
<b>modifies_features</b>	True
<b>modifies_target</b>	False
<b>name</b>	ARIMA Regressor
<b>predict_uses_y</b>	False
<b>supported_problem_types</b>	[ProblemTypes.TIME_SERIES_REGRESSION]

#### Methods

<i>clone</i>	Constructs a new component with the same parameters and random state.
<i>default_parameters</i>	Returns the default parameters for this component.
<i>describe</i>	Describe a component and its parameters
<i>feature_importance</i>	Returns array of 0’s with a length of 1 as feature_importance is not defined for ARIMA regressor.
<i>fit</i>	Fits component to data
<i>load</i>	Loads component at file path
<i>needs_fitting</i>	Returns boolean determining if component needs fitting before
<i>parameters</i>	Returns the parameters which were used to initialize the component
<i>predict</i>	Make predictions using selected features.
<i>predict_proba</i>	Make probability estimates for labels.
<i>save</i>	Saves component at file path

**clone**(*self*)

Constructs a new component with the same parameters and random state.

**Returns** A new instance of this component with identical parameters and random state.

**default\_parameters** (*cls*)

Returns the default parameters for this component.

Our convention is that `Component.default_parameters == Component().parameters`.

**Returns** default parameters for this component.

**Return type** dict

**describe** (*self*, *print\_name=False*, *return\_dict=False*)

Describe a component and its parameters

**Parameters**

- **print\_name** (*bool*, *optional*) – whether to print name of component
- **return\_dict** (*bool*, *optional*) – whether to return description as dictionary in the format {"name": name, "parameters": parameters}

**Returns** prints and returns dictionary

**Return type** None or dict

**property feature\_importance** (*self*)

Returns array of 0's with a length of 1 as feature\_importance is not defined for ARIMA regressor.

**fit** (*self*, *X*, *y=None*)

Fits component to data

**Parameters**

- **X** (*list*, *pd.DataFrame* or *np.ndarray*) – The input training data of shape [n\_samples, n\_features]
- **y** (*list*, *pd.Series*, *np.ndarray*, *optional*) – The target training data of length [n\_samples]

**Returns** self

**static load** (*file\_path*)

Loads component at file path

**Parameters** **file\_path** (*str*) – Location to load file

**Returns** ComponentBase object

**needs\_fitting** (*self*)

Returns boolean determining if component needs fitting before calling predict, predict\_proba, transform, or feature\_importances. This can be overridden to False for components that do not need to be fit or whose fit methods do nothing.

**property parameters** (*self*)

Returns the parameters which were used to initialize the component

**predict** (*self*, *X*, *y=None*)

Make predictions using selected features.

**Parameters** **X** (*pd.DataFrame*, *np.ndarray*) – Data of shape [n\_samples, n\_features]

**Returns** Predicted values

**Return type** pd.Series

**predict\_proba** (*self*, *X*)

Make probability estimates for labels.

**Parameters** *X* (*pd.DataFrame*, or *np.ndarray*) – Features

**Returns** Probability estimates

**Return type** *pd.Series*

**save** (*self*, *file\_path*, *pickle\_protocol=cloudpickle.DEFAULT\_PROTOCOL*)

Saves component at file path

**Parameters**

- **file\_path** (*str*) – Location to save file
- **pickle\_protocol** (*int*) – The pickle data stream format.

**Returns** None

**class** evalml.pipelines.components.estimators.regressors.**BaselineRegressor** (*strategy='mean', random\_seed=0, \*\*kwargs*)

Baseline regressor that uses a simple strategy to make predictions. This is useful as a simple baseline regressor to compare with other regressors.

**Parameters**

- **strategy** (*str*) – Method used to predict. Valid options are “mean”, “median”. Defaults to “mean”.
- **random\_seed** (*int*) – Seed for the random number generator. Defaults to 0.

**Attributes**

<b>hyper-parameter_ranges</b>	{}
<b>model_family</b>	ModelFamily.BASELINE
<b>modifies_features</b>	True
<b>modifies_target</b>	False
<b>name</b>	Baseline Regressor
<b>predict_uses_y</b>	False
<b>supported_problem_types</b>	[ ProblemTypes.REGRESSION, ProblemTypes.TIME_SERIES_REGRESSION,]

**Methods**

<i>clone</i>	Constructs a new component with the same parameters and random state.
<i>default_parameters</i>	Returns the default parameters for this component.
<i>describe</i>	Describe a component and its parameters
<i>feature_importance</i>	Returns importance associated with each feature. Since baseline regressors do not use input features to calculate predictions, returns an array of zeroes.

continues on next page

Table 362 – continued from previous page

<i>fit</i>	Fits component to data
<i>load</i>	Loads component at file path
<i>needs_fitting</i>	Returns boolean determining if component needs fitting before
<i>parameters</i>	Returns the parameters which were used to initialize the component
<i>predict</i>	Make predictions using selected features.
<i>predict_proba</i>	Make probability estimates for labels.
<i>save</i>	Saves component at file path

**clone** (*self*)

Constructs a new component with the same parameters and random state.

**Returns** A new instance of this component with identical parameters and random state.

**default\_parameters** (*cls*)

Returns the default parameters for this component.

Our convention is that `Component.default_parameters == Component().parameters`.

**Returns** default parameters for this component.

**Return type** dict

**describe** (*self*, *print\_name=False*, *return\_dict=False*)

Describe a component and its parameters

**Parameters**

- **print\_name** (*bool*, *optional*) – whether to print name of component
- **return\_dict** (*bool*, *optional*) – whether to return description as dictionary in the format {"name": name, "parameters": parameters}

**Returns** prints and returns dictionary

**Return type** None or dict

**property feature\_importance** (*self*)

Returns importance associated with each feature. Since baseline regressors do not use input features to calculate predictions, returns an array of zeroes.

**Returns** An array of zeroes

**Return type** np.ndarray (float)

**fit** (*self*, *X*, *y=None*)

Fits component to data

**Parameters**

- **X** (*list*, *pd.DataFrame* or *np.ndarray*) – The input training data of shape [n\_samples, n\_features]
- **y** (*list*, *pd.Series*, *np.ndarray*, *optional*) – The target training data of length [n\_samples]

**Returns** self

**static load** (*file\_path*)

Loads component at file path

**Parameters** **file\_path** (*str*) – Location to load file

**Returns** ComponentBase object

**needs\_fitting** (*self*)

Returns boolean determining if component needs fitting before calling predict, predict\_proba, transform, or feature\_importances. This can be overridden to False for components that do not need to be fit or whose fit methods do nothing.

**property parameters** (*self*)

Returns the parameters which were used to initialize the component

**predict** (*self*, *X*)

Make predictions using selected features.

**Parameters** *X* (*pd.DataFrame*, *np.ndarray*) – Data of shape [n\_samples, n\_features]

**Returns** Predicted values

**Return type** *pd.Series*

**predict\_proba** (*self*, *X*)

Make probability estimates for labels.

**Parameters** *X* (*pd.DataFrame*, *or np.ndarray*) – Features

**Returns** Probability estimates

**Return type** *pd.Series*

**save** (*self*, *file\_path*, *pickle\_protocol=cloudpickle.DEFAULT\_PROTOCOL*)

Saves component at file path

**Parameters**

- **file\_path** (*str*) – Location to save file
- **pickle\_protocol** (*int*) – The pickle data stream format.

**Returns** None

```
class evalml.pipelines.components.estimators.regressors.CatBoostRegressor (n_estimators=10,
                                                                    eta=0.03,
                                                                    max_depth=6,
                                                                    boot-
                                                                    strap_type=None,
                                                                    silent=False,
                                                                    al-
                                                                    low_writing_files=False,
                                                                    ran-
                                                                    dom_seed=0,
                                                                    n_jobs=-
                                                                    1,
                                                                    **kwargs)
```

CatBoost Regressor, a regressor that uses gradient-boosting on decision trees. CatBoost is an open-source library and natively supports categorical features.

For more information, check out <https://catboost.ai/>

**Parameters**

- **n\_estimators** (*float*) – The maximum number of trees to build. Defaults to 10.
- **eta** (*float*) – The learning rate. Defaults to 0.03.
- **max\_depth** (*int*) – The maximum tree depth for base learners. Defaults to 6.

- **bootstrap\_type** (*string*) – Defines the method for sampling the weights of objects. Available methods are ‘Bayesian’, ‘Bernoulli’, ‘MVS’. Defaults to None.
- **silent** (*boolean*) – Whether to use the “silent” logging mode. Defaults to True.
- **allow\_writing\_files** (*boolean*) – Whether to allow writing snapshot files while training. Defaults to False.
- **n\_jobs** (*int or None*) – Number of jobs to run in parallel. -1 uses all processes. Defaults to -1.
- **random\_seed** (*int*) – Seed for the random number generator. Defaults to 0.

#### Attributes

<b>hyper-parameter_ranges</b>	{ “n_estimators”: Integer(4, 100), “eta”: Real(0.000001, 1), “max_depth”: Integer(4, 10), }
<b>model_family</b>	ModelFamily.CATBOOST
<b>modifies_features</b>	True
<b>modifies_target</b>	False
<b>name</b>	CatBoost Regressor
<b>predict_uses_y</b>	False
<b>supported_problem_types</b>	[ ProblemTypes.REGRESSION, ProblemTypes.TIME_SERIES_REGRESSION, ]

#### Methods

<i>clone</i>	Constructs a new component with the same parameters and random state.
<i>default_parameters</i>	Returns the default parameters for this component.
<i>describe</i>	Describe a component and its parameters
<i>feature_importance</i>	Returns importance associated with each feature.
<i>fit</i>	Fits component to data
<i>load</i>	Loads component at file path
<i>needs_fitting</i>	Returns boolean determining if component needs fitting before
<i>parameters</i>	Returns the parameters which were used to initialize the component
<i>predict</i>	Make predictions using selected features.
<i>predict_proba</i>	Make probability estimates for labels.
<i>save</i>	Saves component at file path

#### **clone** (*self*)

Constructs a new component with the same parameters and random state.

**Returns** A new instance of this component with identical parameters and random state.

#### **default\_parameters** (*cls*)

Returns the default parameters for this component.

Our convention is that `Component.default_parameters == Component().parameters`.

**Returns** default parameters for this component.



**Return type** dict

**describe** (*self*, *print\_name=False*, *return\_dict=False*)

Describe a component and its parameters

**Parameters**

- **print\_name** (*bool*, *optional*) – whether to print name of component
- **return\_dict** (*bool*, *optional*) – whether to return description as dictionary in the format {"name": name, "parameters": parameters}

**Returns** prints and returns dictionary

**Return type** None or dict

**property feature\_importance** (*self*)

Returns importance associated with each feature.

**Returns** Importance associated with each feature

**Return type** np.ndarray

**fit** (*self*, *X*, *y=None*)

Fits component to data

**Parameters**

- **X** (*list*, *pd.DataFrame* or *np.ndarray*) – The input training data of shape [n\_samples, n\_features]
- **y** (*list*, *pd.Series*, *np.ndarray*, *optional*) – The target training data of length [n\_samples]

**Returns** self

**static load** (*file\_path*)

Loads component at file path

**Parameters** **file\_path** (*str*) – Location to load file

**Returns** ComponentBase object

**needs\_fitting** (*self*)

Returns boolean determining if component needs fitting before calling predict, predict\_proba, transform, or feature\_importances. This can be overridden to False for components that do not need to be fit or whose fit methods do nothing.

**property parameters** (*self*)

Returns the parameters which were used to initialize the component

**predict** (*self*, *X*)

Make predictions using selected features.

**Parameters** **X** (*pd.DataFrame*, *np.ndarray*) – Data of shape [n\_samples, n\_features]

**Returns** Predicted values

**Return type** pd.Series

**predict\_proba** (*self*, *X*)

Make probability estimates for labels.

**Parameters** **X** (*pd.DataFrame*, or *np.ndarray*) – Features

**Returns** Probability estimates

**Return type** `pd.Series`

**save** (*self*, *file\_path*, *pickle\_protocol*=`cloudpickle.DEFAULT_PROTOCOL`)  
Saves component at file path

**Parameters**

- **file\_path** (*str*) – Location to save file
- **pickle\_protocol** (*int*) – The pickle data stream format.

**Returns** `None`

**class** `evalml.pipelines.components.estimators.regressors.DecisionTreeRegressor` (*criterion*='mse',  
*max\_features*='auto',  
*max\_depth*=6,  
*min\_samples\_split*=2,  
*min\_weight\_fraction*=0,  
*random\_state*=0,  
\*\**kwargs*)

Decision Tree Regressor.

**Parameters**

- **criterion** (`{ "mse", "friedman_mse", "mae", "poisson" }`) – The function to measure the quality of a split. Supported criteria are:
  - “mse” for the mean squared error, which is equal to variance reduction as feature selection criterion and minimizes the L2 loss using the mean of each terminal node
  - “friedman\_mse”, which uses mean squared error with Friedman’s improvement score for potential splits
  - “mae” for the mean absolute error, which minimizes the L1 loss using the median of each terminal node,
  - “poisson” which uses reduction in Poisson deviance to find splits.
- **max\_features** (*int*, *float* or `{ "auto", "sqrt", "log2" }`) – The number of features to consider when looking for the best split:
  - If *int*, then consider `max_features` features at each split.
  - If *float*, then `max_features` is a fraction and `int(max_features * n_features)` features are considered at each split.
  - If “auto”, then `max_features=sqrt(n_features)`.
  - If “sqrt”, then `max_features=sqrt(n_features)`.
  - If “log2”, then `max_features=log2(n_features)`.
  - If *None*, then `max_features = n_features`.

The search for a split does not stop until at least one valid partition of the node samples is found, even if it requires to effectively inspect more than `max_features` features.

- **max\_depth** (*int*) – The maximum depth of the tree. Defaults to 6.
- **min\_samples\_split** (*int* or *float*) – The minimum number of samples required to split an internal node:
  - If *int*, then consider `min_samples_split` as the minimum number.

- If float, then `min_samples_split` is a fraction and `ceil(min_samples_split * n_samples)` are the minimum number of samples for each split.

Defaults to 2.

- **`min_weight_fraction_leaf`** (*float*) – The minimum weighted fraction of the sum total of weights (of all the input samples) required to be at a leaf node. Defaults to 0.0.
- **`random_seed`** (*int*) – Seed for the random number generator. Defaults to 0.

#### Attributes

<b>hyper-parameter_ranges</b>	{ “criterion”: [“mse”, “friedman_mse”, “mae”], “max_features”: [“auto”, “sqrt”, “log2”], “max_depth”: Integer(4, 10),}
<b>model_family</b>	ModelFamily.DECISION_TREE
<b>modifies_features</b>	True
<b>modifies_target</b>	False
<b>name</b>	Decision Tree Regressor
<b>predict_uses_y</b>	False
<b>supported_problem_types</b>	[ ProblemTypes.REGRESSION, ProblemTypes.TIME_SERIES_REGRESSION,]

#### Methods

<i>clone</i>	Constructs a new component with the same parameters and random state.
<i>default_parameters</i>	Returns the default parameters for this component.
<i>describe</i>	Describe a component and its parameters
<i>feature_importance</i>	Returns importance associated with each feature.
<i>fit</i>	Fits component to data
<i>load</i>	Loads component at file path
<i>needs_fitting</i>	Returns boolean determining if component needs fitting before
<i>parameters</i>	Returns the parameters which were used to initialize the component
<i>predict</i>	Make predictions using selected features.
<i>predict_proba</i>	Make probability estimates for labels.
<i>save</i>	Saves component at file path

#### **`clone`** (*self*)

Constructs a new component with the same parameters and random state.

**Returns** A new instance of this component with identical parameters and random state.

#### **`default_parameters`** (*cls*)

Returns the default parameters for this component.

Our convention is that `Component.default_parameters == Component().parameters`.

**Returns** default parameters for this component.

**Return type** dict

**describe** (*self*, *print\_name=False*, *return\_dict=False*)

Describe a component and its parameters

**Parameters**

- **print\_name** (*bool*, *optional*) – whether to print name of component
- **return\_dict** (*bool*, *optional*) – whether to return description as dictionary in the format {"name": name, "parameters": parameters}

**Returns** prints and returns dictionary

**Return type** None or dict

**property feature\_importance** (*self*)

Returns importance associated with each feature.

**Returns** Importance associated with each feature

**Return type** np.ndarray

**fit** (*self*, *X*, *y=None*)

Fits component to data

**Parameters**

- **X** (*list*, *pd.DataFrame* or *np.ndarray*) – The input training data of shape [n\_samples, n\_features]
- **y** (*list*, *pd.Series*, *np.ndarray*, *optional*) – The target training data of length [n\_samples]

**Returns** self

**static load** (*file\_path*)

Loads component at file path

**Parameters** **file\_path** (*str*) – Location to load file

**Returns** ComponentBase object

**needs\_fitting** (*self*)

Returns boolean determining if component needs fitting before calling predict, predict\_proba, transform, or feature\_importances. This can be overridden to False for components that do not need to be fit or whose fit methods do nothing.

**property parameters** (*self*)

Returns the parameters which were used to initialize the component

**predict** (*self*, *X*)

Make predictions using selected features.

**Parameters** **X** (*pd.DataFrame*, *np.ndarray*) – Data of shape [n\_samples, n\_features]

**Returns** Predicted values

**Return type** pd.Series

**predict\_proba** (*self*, *X*)

Make probability estimates for labels.

**Parameters** **X** (*pd.DataFrame*, or *np.ndarray*) – Features

**Returns** Probability estimates

**Return type** pd.Series

**save** (*self*, *file\_path*, *pickle\_protocol*=*cloudpickle.DEFAULT\_PROTOCOL*)

Saves component at file path

#### Parameters

- **file\_path** (*str*) – Location to save file
- **pickle\_protocol** (*int*) – The pickle data stream format.

**Returns** None

```
class evalml.pipelines.components.estimators.regressors.ElasticNetRegressor (alpha=0.0001,  
                                                                    l1_ratio=0.15,  
                                                                    max_iter=1000,  
                                                                    nor-  
                                                                    mal-  
                                                                    ize=False,  
                                                                    ran-  
                                                                    dom_seed=0,  
                                                                    **kwargs)
```

Elastic Net Regressor.

#### Parameters

- **alpha** (*float*) – Constant that multiplies the penalty terms. Defaults to 0.0001.
- **l1\_ratio** (*float*) – The mixing parameter, with  $0 \leq \text{l1\_ratio} \leq 1$ . Only used if `penalty='elasticnet'`. Setting `l1_ratio=0` is equivalent to using `penalty='l2'`, while setting `l1_ratio=1` is equivalent to using `penalty='l1'`. For  $0 < \text{l1\_ratio} < 1$ , the penalty is a combination of L1 and L2. Defaults to 0.15.
- **max\_iter** (*int*) – The maximum number of iterations. Defaults to 1000.
- **normalize** (*boolean*) – If True, the regressors will be normalized before regression by subtracting the mean and dividing by the l2-norm. Defaults to False.
- **random\_seed** (*int*) – Seed for the random number generator. Defaults to 0.

#### Attributes

<b>hyper-parameter_ranges</b>	{ "alpha": Real(0, 1), "l1_ratio": Real(0, 1), }
<b>model_family</b>	ModelFamily.LINEAR_MODEL
<b>modifies_features</b>	True
<b>modifies_target</b>	False
<b>name</b>	Elastic Net Regressor
<b>pre-dict_uses_y</b>	False
<b>supported_problem_types</b>	[ ProblemTypes.REGRESSION, ProblemTypes.TIME_SERIES_REGRESSION, ]

#### Methods

---

*clone*

Constructs a new component with the same parameters and random state.

continues on next page

---

Table 365 – continued from previous page

<code>default_parameters</code>	Returns the default parameters for this component.
<code>describe</code>	Describe a component and its parameters
<code>feature_importance</code>	Returns importance associated with each feature.
<code>fit</code>	Fits component to data
<code>load</code>	Loads component at file path
<code>needs_fitting</code>	Returns boolean determining if component needs fitting before
<code>parameters</code>	Returns the parameters which were used to initialize the component
<code>predict</code>	Make predictions using selected features.
<code>predict_proba</code>	Make probability estimates for labels.
<code>save</code>	Saves component at file path

**clone** (*self*)

Constructs a new component with the same parameters and random state.

**Returns** A new instance of this component with identical parameters and random state.

**default\_parameters** (*cls*)

Returns the default parameters for this component.

Our convention is that `Component.default_parameters == Component().parameters`.

**Returns** default parameters for this component.

**Return type** dict

**describe** (*self*, *print\_name=False*, *return\_dict=False*)

Describe a component and its parameters

**Parameters**

- **print\_name** (*bool*, *optional*) – whether to print name of component
- **return\_dict** (*bool*, *optional*) – whether to return description as dictionary in the format {"name": name, "parameters": parameters}

**Returns** prints and returns dictionary

**Return type** None or dict

**property feature\_importance** (*self*)

Returns importance associated with each feature.

**Returns** Importance associated with each feature

**Return type** np.ndarray

**fit** (*self*, *X*, *y=None*)

Fits component to data

**Parameters**

- **X** (*list*, *pd.DataFrame* or *np.ndarray*) – The input training data of shape [n\_samples, n\_features]
- **y** (*list*, *pd.Series*, *np.ndarray*, *optional*) – The target training data of length [n\_samples]

**Returns** self

**static load** (*file\_path*)

Loads component at file path

**Parameters** *file\_path* (*str*) – Location to load file

**Returns** ComponentBase object

**needs\_fitting** (*self*)

Returns boolean determining if component needs fitting before calling predict, predict\_proba, transform, or feature\_importances. This can be overridden to False for components that do not need to be fit or whose fit methods do nothing.

**property parameters** (*self*)

Returns the parameters which were used to initialize the component

**predict** (*self*, *X*)

Make predictions using selected features.

**Parameters** *X* (*pd.DataFrame*, *np.ndarray*) – Data of shape [n\_samples, n\_features]

**Returns** Predicted values

**Return type** *pd.Series*

**predict\_proba** (*self*, *X*)

Make probability estimates for labels.

**Parameters** *X* (*pd.DataFrame*, *or np.ndarray*) – Features

**Returns** Probability estimates

**Return type** *pd.Series*

**save** (*self*, *file\_path*, *pickle\_protocol=cloudpickle.DEFAULT\_PROTOCOL*)

Saves component at file path

**Parameters**

- **file\_path** (*str*) – Location to save file
- **pickle\_protocol** (*int*) – The pickle data stream format.

**Returns** None

```
class evalml.pipelines.components.estimators.regressors.ExtraTreesRegressor (n_estimators=100,
                                                                    max_features='auto',
                                                                    max_depth=6,
                                                                    min_samples_split=2,
                                                                    min_weight_fraction,
                                                                    n_jobs=-
                                                                    1,
                                                                    ran-
                                                                    dom_seed=0,
                                                                    **kwargs)
```

Extra Trees Regressor.

**Parameters**

- **n\_estimators** (*float*) – The number of trees in the forest. Defaults to 100.
- **max\_features** (*int*, *float* *or* {"auto", "sqrt", "log2"}) – The number of features to consider when looking for the best split:
  - If int, then consider max\_features features at each split.

- If float, then `max_features` is a fraction and `int(max_features * n_features)` features are considered at each split.
- If “auto”, then `max_features=sqrt(n_features)`.
- If “sqrt”, then `max_features=sqrt(n_features)`.
- If “log2”, then `max_features=log2(n_features)`.
- If None, then `max_features = n_features`.

The search for a split does not stop until at least one valid partition of the node samples is found, even if it requires to effectively inspect more than `max_features` features. Defaults to “auto”.

- **`max_depth`** (*int*) – The maximum depth of the tree. Defaults to 6.
- **`min_samples_split`** (*int or float*) – The minimum number of samples required to split an internal node:
  - If int, then consider `min_samples_split` as the minimum number.
  - If float, then `min_samples_split` is a fraction and `ceil(min_samples_split * n_samples)` are the minimum number of samples for each split.
- **`to 2.`** (*Defaults*) –
- **`min_weight_fraction_leaf`** (*float*) – The minimum weighted fraction of the sum total of weights (of all the input samples) required to be at a leaf node. Defaults to 0.0.
- **`n_jobs`** (*int or None*) – Number of jobs to run in parallel. -1 uses all processes. Defaults to -1.
- **`random_seed`** (*int*) – Seed for the random number generator. Defaults to 0.

#### Attributes

<b>hyper-parameter_ranges</b>	{ “n_estimators”: Integer(10, 1000), “max_features”: [“auto”, “sqrt”, “log2”], “max_depth”: Integer(4, 10), }
<b>model_family</b>	ModelFamily.EXTRA_TREES
<b>modifies_features</b>	True
<b>modifies_target</b>	False
<b>name</b>	Extra Trees Regressor
<b>predict_uses_y</b>	False
<b>supported_problem_types</b>	[ ProblemTypes.REGRESSION, ProblemTypes.TIME_SERIES_REGRESSION, ]

#### Methods

<code>clone</code>	Constructs a new component with the same parameters and random state.
<code>default_parameters</code>	Returns the default parameters for this component.
<code>describe</code>	Describe a component and its parameters
<code>feature_importance</code>	Returns importance associated with each feature.
<code>fit</code>	Fits component to data

continues on next page



Table 366 – continued from previous page

<code>load</code>	Loads component at file path
<code>needs_fitting</code>	Returns boolean determining if component needs fitting before
<code>parameters</code>	Returns the parameters which were used to initialize the component
<code>predict</code>	Make predictions using selected features.
<code>predict_proba</code>	Make probability estimates for labels.
<code>save</code>	Saves component at file path

**clone** (*self*)

Constructs a new component with the same parameters and random state.

**Returns** A new instance of this component with identical parameters and random state.

**default\_parameters** (*cls*)

Returns the default parameters for this component.

Our convention is that `Component.default_parameters == Component().parameters`.

**Returns** default parameters for this component.

**Return type** dict

**describe** (*self*, *print\_name=False*, *return\_dict=False*)

Describe a component and its parameters

**Parameters**

- **print\_name** (*bool*, *optional*) – whether to print name of component
- **return\_dict** (*bool*, *optional*) – whether to return description as dictionary in the format {"name": name, "parameters": parameters}

**Returns** prints and returns dictionary

**Return type** None or dict

**property feature\_importance** (*self*)

Returns importance associated with each feature.

**Returns** Importance associated with each feature

**Return type** np.ndarray

**fit** (*self*, *X*, *y=None*)

Fits component to data

**Parameters**

- **X** (*list*, *pd.DataFrame* or *np.ndarray*) – The input training data of shape [n\_samples, n\_features]
- **y** (*list*, *pd.Series*, *np.ndarray*, *optional*) – The target training data of length [n\_samples]

**Returns** self

**static load** (*file\_path*)

Loads component at file path

**Parameters** **file\_path** (*str*) – Location to load file

**Returns** ComponentBase object

**needs\_fitting** (*self*)

Returns boolean determining if component needs fitting before calling predict, predict\_proba, transform, or feature\_importances. This can be overridden to False for components that do not need to be fit or whose fit methods do nothing.

**property parameters** (*self*)

Returns the parameters which were used to initialize the component

**predict** (*self*, *X*)

Make predictions using selected features.

**Parameters** *X* (*pd.DataFrame*, *np.ndarray*) – Data of shape [n\_samples, n\_features]

**Returns** Predicted values

**Return type** *pd.Series*

**predict\_proba** (*self*, *X*)

Make probability estimates for labels.

**Parameters** *X* (*pd.DataFrame*, or *np.ndarray*) – Features

**Returns** Probability estimates

**Return type** *pd.Series*

**save** (*self*, *file\_path*, *pickle\_protocol=cloudpickle.DEFAULT\_PROTOCOL*)

Saves component at file path

**Parameters**

- **file\_path** (*str*) – Location to save file
- **pickle\_protocol** (*int*) – The pickle data stream format.

**Returns** None

```
class evalml.pipelines.components.estimators.regressors.LightGBMRegressor (boosting_type='gbdt',  
                                                                    learn-  
                                                                    ing_rate=0.1,  
                                                                    n_estimators=20,  
                                                                    max_depth=0,  
                                                                    num_leaves=31,  
                                                                    min_child_samples=20,  
                                                                    bag-  
                                                                    ging_fraction=0.9,  
                                                                    bag-  
                                                                    ging_freq=0,  
                                                                    n_jobs=-  
  
                                                                    1,  
                                                                    ran-  
                                                                    dom_seed=0,  
                                                                    **kwargs)
```

LightGBM Regressor.

**Parameters**

- **boosting\_type** (*string*) – Type of boosting to use. Defaults to “gbdt”. - ‘gbdt’ uses traditional Gradient Boosting Decision Tree - “dart”, uses Dropouts meet Multiple Additive Regression Trees - “goss”, uses Gradient-based One-Side Sampling - “rf”, uses Random Forest

- **learning\_rate** (*float*) – Boosting learning rate. Defaults to 0.1.
- **n\_estimators** (*int*) – Number of boosted trees to fit. Defaults to 100.
- **max\_depth** (*int*) – Maximum tree depth for base learners,  $\leq 0$  means no limit. Defaults to 0.
- **num\_leaves** (*int*) – Maximum tree leaves for base learners. Defaults to 31.
- **min\_child\_samples** (*int*) – Minimum number of data needed in a child (leaf). Defaults to 20.
- **bagging\_fraction** (*float*) – LightGBM will randomly select a subset of features on each iteration (tree) without resampling if this is smaller than 1.0. For example, if set to 0.8, LightGBM will select 80% of features before training each tree. This can be used to speed up training and deal with overfitting. Defaults to 0.9.
- **bagging\_freq** (*int*) – Frequency for bagging. 0 means bagging is disabled. k means perform bagging at every k iteration. Every k-th iteration, LightGBM will randomly select  $\text{bagging\_fraction} * 100 \%$  of the data to use for the next k iterations. Defaults to 0.
- **n\_jobs** (*int or None*) – Number of threads to run in parallel. -1 uses all threads. Defaults to -1.
- **random\_seed** (*int*) – Seed for the random number generator. Defaults to 0.

#### Attributes

<b>hyper-parameter_ranges</b>	{ "learning_rate": Real(0.000001, 1), "boosting_type": ["gbdt", "dart", "goss", "rf"], "n_estimators": Integer(10, 100), "max_depth": Integer(0, 10), "num_leaves": Integer(2, 100), "min_child_samples": Integer(1, 100), "bagging_fraction": Real(0.000001, 1), "bagging_freq": Integer(0, 1), }
<b>model_family</b>	ModelFamily.LIGHTGBM
<b>modifies_features</b>	True
<b>modifies_target</b>	False
<b>name</b>	LightGBM Regressor
<b>predict_uses_y</b>	False
<b>SEED_MAX</b>	SEED_BOUNDS.max_bound
<b>SEED_MIN</b>	0
<b>supported_problem_types</b>	[ProblemTypes.REGRESSION]

#### Methods

<code>clone</code>	Constructs a new component with the same parameters and random state.
<code>default_parameters</code>	Returns the default parameters for this component.
<code>describe</code>	Describe a component and its parameters
<code>feature_importance</code>	Returns importance associated with each feature.
<code>fit</code>	Fits component to data
<code>load</code>	Loads component at file path
<code>needs_fitting</code>	Returns boolean determining if component needs fitting before

continues on next page

Table 367 – continued from previous page

<i>parameters</i>	Returns the parameters which were used to initialize the component
<i>predict</i>	Make predictions using selected features.
<i>predict_proba</i>	Make probability estimates for labels.
<i>save</i>	Saves component at file path

**clone** (*self*)

Constructs a new component with the same parameters and random state.

**Returns** A new instance of this component with identical parameters and random state.

**default\_parameters** (*cls*)

Returns the default parameters for this component.

Our convention is that `Component.default_parameters == Component().parameters`.

**Returns** default parameters for this component.

**Return type** dict

**describe** (*self*, *print\_name=False*, *return\_dict=False*)

Describe a component and its parameters

**Parameters**

- **print\_name** (*bool*, *optional*) – whether to print name of component
- **return\_dict** (*bool*, *optional*) – whether to return description as dictionary in the format {"name": name, "parameters": parameters}

**Returns** prints and returns dictionary

**Return type** None or dict

**property feature\_importance** (*self*)

Returns importance associated with each feature.

**Returns** Importance associated with each feature

**Return type** np.ndarray

**fit** (*self*, *X*, *y=None*)

Fits component to data

**Parameters**

- **X** (*list*, *pd.DataFrame* or *np.ndarray*) – The input training data of shape [n\_samples, n\_features]
- **y** (*list*, *pd.Series*, *np.ndarray*, *optional*) – The target training data of length [n\_samples]

**Returns** self

**static load** (*file\_path*)

Loads component at file path

**Parameters** **file\_path** (*str*) – Location to load file

**Returns** ComponentBase object

**needs\_fitting** (*self*)

Returns boolean determining if component needs fitting before calling predict, predict\_proba, transform,

or `feature_importances`. This can be overridden to `False` for components that do not need to be fit or whose fit methods do nothing.

**property parameters** (*self*)

Returns the parameters which were used to initialize the component

**predict** (*self*, *X*)

Make predictions using selected features.

**Parameters** *X* (*pd.DataFrame*, *np.ndarray*) – Data of shape [n\_samples, n\_features]

**Returns** Predicted values

**Return type** *pd.Series*

**predict\_proba** (*self*, *X*)

Make probability estimates for labels.

**Parameters** *X* (*pd.DataFrame*, or *np.ndarray*) – Features

**Returns** Probability estimates

**Return type** *pd.Series*

**save** (*self*, *file\_path*, *pickle\_protocol=cloudpickle.DEFAULT\_PROTOCOL*)

Saves component at file path

**Parameters**

- **file\_path** (*str*) – Location to save file
- **pickle\_protocol** (*int*) – The pickle data stream format.

**Returns** `None`

```
class evalml.pipelines.components.estimators.regressors.LinearRegressor (fit_intercept=True,
                                                                    normalize=False,
                                                                    n_jobs=-1,
                                                                    random_seed=0,
                                                                    **kwargs)
```

Linear Regressor.

**Parameters**

- **fit\_intercept** (*boolean*) – Whether to calculate the intercept for this model. If set to `False`, no intercept will be used in calculations (i.e. data is expected to be centered). Defaults to `True`.
- **normalize** (*boolean*) – If `True`, the regressors will be normalized before regression by subtracting the mean and dividing by the l2-norm. This parameter is ignored when `fit_intercept` is set to `False`. Defaults to `False`.
- **n\_jobs** (*int* or *None*) – Number of jobs to run in parallel. -1 uses all threads. Defaults to -1.
- **random\_seed** (*int*) – Seed for the random number generator. Defaults to 0.

**Attributes**

<b>hyper-parameter_ranges</b>	{ “fit_intercept”: [True, False], “normalize”: [True, False]}
<b>model_family</b>	ModelFamily.LINEAR_MODEL
<b>modifies_features</b>	True
<b>modifies_target</b>	False
<b>name</b>	Linear Regressor
<b>pre-dict_uses_y</b>	False
<b>supported_problem_types</b>	[ ProblemTypes.REGRESSION, ProblemTypes.TIME_SERIES_REGRESSION,]

## Methods

<i>clone</i>	Constructs a new component with the same parameters and random state.
<i>default_parameters</i>	Returns the default parameters for this component.
<i>describe</i>	Describe a component and its parameters
<i>feature_importance</i>	Returns importance associated with each feature.
<i>fit</i>	Fits component to data
<i>load</i>	Loads component at file path
<i>needs_fitting</i>	Returns boolean determining if component needs fitting before
<i>parameters</i>	Returns the parameters which were used to initialize the component
<i>predict</i>	Make predictions using selected features.
<i>predict_proba</i>	Make probability estimates for labels.
<i>save</i>	Saves component at file path

### **clone** (*self*)

Constructs a new component with the same parameters and random state.

**Returns** A new instance of this component with identical parameters and random state.

### **default\_parameters** (*cls*)

Returns the default parameters for this component.

Our convention is that `Component.default_parameters == Component().parameters`.

**Returns** default parameters for this component.

**Return type** dict

### **describe** (*self*, *print\_name=False*, *return\_dict=False*)

Describe a component and its parameters

#### Parameters

- **print\_name** (*bool*, *optional*) – whether to print name of component
- **return\_dict** (*bool*, *optional*) – whether to return description as dictionary in the format {“name”: name, “parameters”: parameters}

**Returns** prints and returns dictionary

**Return type** None or dict

**property feature\_importance** (*self*)

Returns importance associated with each feature.

**Returns** Importance associated with each feature

**Return type** np.ndarray

**fit** (*self*, *X*, *y=None*)

Fits component to data

**Parameters**

- **X** (*list*, *pd.DataFrame* or *np.ndarray*) – The input training data of shape [n\_samples, n\_features]
- **y** (*list*, *pd.Series*, *np.ndarray*, *optional*) – The target training data of length [n\_samples]

**Returns** self

**static load** (*file\_path*)

Loads component at file path

**Parameters** **file\_path** (*str*) – Location to load file

**Returns** ComponentBase object

**needs\_fitting** (*self*)

Returns boolean determining if component needs fitting before calling predict, predict\_proba, transform, or feature\_importances. This can be overridden to False for components that do not need to be fit or whose fit methods do nothing.

**property parameters** (*self*)

Returns the parameters which were used to initialize the component

**predict** (*self*, *X*)

Make predictions using selected features.

**Parameters** **X** (*pd.DataFrame*, *np.ndarray*) – Data of shape [n\_samples, n\_features]

**Returns** Predicted values

**Return type** pd.Series

**predict\_proba** (*self*, *X*)

Make probability estimates for labels.

**Parameters** **X** (*pd.DataFrame*, or *np.ndarray*) – Features

**Returns** Probability estimates

**Return type** pd.Series

**save** (*self*, *file\_path*, *pickle\_protocol=cloudpickle.DEFAULT\_PROTOCOL*)

Saves component at file path

**Parameters**

- **file\_path** (*str*) – Location to save file
- **pickle\_protocol** (*int*) – The pickle data stream format.

**Returns** None

```
class evalml.pipelines.components.estimators.regressors.ProphetRegressor (date_column='ds',
                                                                           change-
                                                                           point_prior_scale=0.05,
                                                                           sea-
                                                                           son-
                                                                           al-
                                                                           ity_prior_scale=10,
                                                                           hol-
                                                                           i-
                                                                           days_prior_scale=10,
                                                                           sea-
                                                                           son-
                                                                           al-
                                                                           ity_mode='additive',
                                                                           ran-
                                                                           dom_seed=0,
                                                                           stan_backend='CMDSTAN',
                                                                           **kwargs)
```

Prophet is a procedure for forecasting time series data based on an additive model where non-linear trends are fit with yearly, weekly, and daily seasonality, plus holiday effects. It works best with time series that have strong seasonal effects and several seasons of historical data. Prophet is robust to missing data and shifts in the trend, and typically handles outliers well.

More information here: <https://facebook.github.io/prophet/>

Attributes

hyper-parameter_ranges	{ "changepoint_prior_scale": Real(0.001, 0.5), "seasonality_prior_scale": Real(0.01, 10), "holidays_prior_scale": Real(0.01, 10), "seasonality_mode": ["additive", "multiplicative"], }
model_family	ModelFamily.PROPHET
modifies_features	True
modifies_target	False
name	Prophet Regressor
pre-dict_uses_y	False
supported_problem_types	[ProblemTypes.TIME_SERIES_REGRESSION]

Methods

<i>build_prophet_df</i>	
<i>clone</i>	Constructs a new component with the same parameters and random state.
<i>default_parameters</i>	Returns the default parameters for this component.
<i>describe</i>	Describe a component and its parameters
<i>feature_importance</i>	Returns array of 0's with len(1) as feature_importance is not defined for Prophet regressor.
<i>fit</i>	Fits component to data

continues on next page



Table 369 – continued from previous page

<i>get_params</i>	
<i>load</i>	Loads component at file path
<i>needs_fitting</i>	Returns boolean determining if component needs fitting before
<i>parameters</i>	Returns the parameters which were used to initialize the component
<i>predict</i>	Make predictions using selected features.
<i>predict_proba</i>	Make probability estimates for labels.
<i>save</i>	Saves component at file path

**static build\_prophet\_df** (*X*, *y=None*, *date\_column='ds'*)

**clone** (*self*)

Constructs a new component with the same parameters and random state.

**Returns** A new instance of this component with identical parameters and random state.

**default\_parameters** (*cls*)

Returns the default parameters for this component.

Our convention is that `Component.default_parameters == Component().parameters`.

**Returns** default parameters for this component.

**Return type** dict

**describe** (*self*, *print\_name=False*, *return\_dict=False*)

Describe a component and its parameters

**Parameters**

- **print\_name** (*bool*, *optional*) – whether to print name of component
- **return\_dict** (*bool*, *optional*) – whether to return description as dictionary in the format {"name": name, "parameters": parameters}

**Returns** prints and returns dictionary

**Return type** None or dict

**property feature\_importance** (*self*)

Returns array of 0's with len(1) as feature\_importance is not defined for Prophet regressor.

**fit** (*self*, *X*, *y=None*)

Fits component to data

**Parameters**

- **X** (*list*, *pd.DataFrame* or *np.ndarray*) – The input training data of shape [n\_samples, n\_features]
- **y** (*list*, *pd.Series*, *np.ndarray*, *optional*) – The target training data of length [n\_samples]

**Returns** self

**get\_params** (*self*)

**static load** (*file\_path*)

Loads component at file path

**Parameters** **file\_path** (*str*) – Location to load file

**Returns** ComponentBase object

**needs\_fitting** (*self*)

Returns boolean determining if component needs fitting before calling predict, predict\_proba, transform, or feature\_importances. This can be overridden to False for components that do not need to be fit or whose fit methods do nothing.

**property parameters** (*self*)

Returns the parameters which were used to initialize the component

**predict** (*self*, *X*, *y=None*)

Make predictions using selected features.

**Parameters** *X* (*pd.DataFrame*, *np.ndarray*) – Data of shape [n\_samples, n\_features]

**Returns** Predicted values

**Return type** *pd.Series*

**predict\_proba** (*self*, *X*)

Make probability estimates for labels.

**Parameters** *X* (*pd.DataFrame*, *or np.ndarray*) – Features

**Returns** Probability estimates

**Return type** *pd.Series*

**save** (*self*, *file\_path*, *pickle\_protocol=cloudpickle.DEFAULT\_PROTOCOL*)

Saves component at file path

**Parameters**

- **file\_path** (*str*) – Location to save file
- **pickle\_protocol** (*int*) – The pickle data stream format.

**Returns** None

```
class evalml.pipelines.components.estimators.regressors.RandomForestRegressor (n_estimators=100,  
                                                                    max_depth=6,  
                                                                    n_jobs=-1,  
                                                                    1,  
                                                                    ran-  
                                                                    dom_seed=0,  
                                                                    **kwargs)
```

Random Forest Regressor.

**Parameters**

- **n\_estimators** (*float*) – The number of trees in the forest. Defaults to 100.
- **max\_depth** (*int*) – Maximum tree depth for base learners. Defaults to 6.
- **n\_jobs** (*int or None*) – Number of jobs to run in parallel. -1 uses all processes. Defaults to -1.
- **random\_seed** (*int*) – Seed for the random number generator. Defaults to 0.

**Attributes**

<b>hyper-parameter_ranges</b>	{ "n_estimators": Integer(10, 1000), "max_depth": Integer(1, 32), }
<b>model_family</b>	ModelFamily.RANDOM_FOREST
<b>modifies_features</b>	True
<b>modifies_target</b>	False
<b>name</b>	Random Forest Regressor
<b>pre-dict_uses_y</b>	False
<b>supported_problem_types</b>	[ ProblemTypes.REGRESSION, ProblemTypes.TIME_SERIES_REGRESSION, ]

## Methods

<i>clone</i>	Constructs a new component with the same parameters and random state.
<i>default_parameters</i>	Returns the default parameters for this component.
<i>describe</i>	Describe a component and its parameters
<i>feature_importance</i>	Returns importance associated with each feature.
<i>fit</i>	Fits component to data
<i>load</i>	Loads component at file path
<i>needs_fitting</i>	Returns boolean determining if component needs fitting before
<i>parameters</i>	Returns the parameters which were used to initialize the component
<i>predict</i>	Make predictions using selected features.
<i>predict_proba</i>	Make probability estimates for labels.
<i>save</i>	Saves component at file path

### **clone** (*self*)

Constructs a new component with the same parameters and random state.

**Returns** A new instance of this component with identical parameters and random state.

### **default\_parameters** (*cls*)

Returns the default parameters for this component.

Our convention is that `Component.default_parameters == Component().parameters`.

**Returns** default parameters for this component.

**Return type** dict

### **describe** (*self*, *print\_name=False*, *return\_dict=False*)

Describe a component and its parameters

#### Parameters

- **print\_name** (*bool*, *optional*) – whether to print name of component
- **return\_dict** (*bool*, *optional*) – whether to return description as dictionary in the format {“name”: name, “parameters”: parameters}

**Returns** prints and returns dictionary

**Return type** None or dict

**property feature\_importance** (*self*)

Returns importance associated with each feature.

**Returns** Importance associated with each feature

**Return type** np.ndarray

**fit** (*self*, *X*, *y=None*)

Fits component to data

**Parameters**

- **X** (*list*, *pd.DataFrame* or *np.ndarray*) – The input training data of shape [n\_samples, n\_features]
- **y** (*list*, *pd.Series*, *np.ndarray*, *optional*) – The target training data of length [n\_samples]

**Returns** self

**static load** (*file\_path*)

Loads component at file path

**Parameters** **file\_path** (*str*) – Location to load file

**Returns** ComponentBase object

**needs\_fitting** (*self*)

Returns boolean determining if component needs fitting before calling predict, predict\_proba, transform, or feature\_importances. This can be overridden to False for components that do not need to be fit or whose fit methods do nothing.

**property parameters** (*self*)

Returns the parameters which were used to initialize the component

**predict** (*self*, *X*)

Make predictions using selected features.

**Parameters** **X** (*pd.DataFrame*, *np.ndarray*) – Data of shape [n\_samples, n\_features]

**Returns** Predicted values

**Return type** pd.Series

**predict\_proba** (*self*, *X*)

Make probability estimates for labels.

**Parameters** **X** (*pd.DataFrame*, or *np.ndarray*) – Features

**Returns** Probability estimates

**Return type** pd.Series

**save** (*self*, *file\_path*, *pickle\_protocol=cloudpickle.DEFAULT\_PROTOCOL*)

Saves component at file path

**Parameters**

- **file\_path** (*str*) – Location to save file
- **pickle\_protocol** (*int*) – The pickle data stream format.

**Returns** None

```

class evalml.pipelines.components.estimators.regressors.SVMRegressor(C=1.0,
                                                                    ker-
                                                                    nel='rbf',
                                                                    gamma='scale',
                                                                    ran-
                                                                    dom_seed=0,
                                                                    **kwargs)

```

Support Vector Machine Regressor.

### Parameters

- **C** (*float*) – The regularization parameter. The strength of the regularization is inversely proportional to C. Must be strictly positive. The penalty is a squared l2 penalty. Defaults to 1.0.
- **kernel** (*{ "linear", "poly", "rbf", "sigmoid", "precomputed" }*) – Specifies the kernel type to be used in the algorithm. Defaults to “rbf”.
- **gamma** (*{ "scale", "auto" } or float*) – Kernel coefficient for “rbf”, “poly” and “sigmoid”. Defaults to “scale”. - If gamma=’scale’ (default) is passed then it uses  $1 / (n\_features * X.var())$  as value of gamma - if “auto”, uses  $1 / n\_features$
- **random\_seed** (*int*) – Seed for the random number generator. Defaults to 0.

### Attributes

<b>hyper-parameter_ranges</b>	{ “C”: Real(0, 10), “kernel”: [“linear”, “poly”, “rbf”, “sigmoid”, “precomputed”], “gamma”: [“scale”, “auto”], }
<b>model_family</b>	ModelFamily.SVM
<b>modifies_features</b>	True
<b>modifies_target</b>	False
<b>name</b>	SVM Regressor
<b>predict_uses_y</b>	False
<b>supported_problem_types</b>	[ ProblemTypes.REGRESSION, ProblemTypes.TIME_SERIES_REGRESSION, ]

### Methods

<i>clone</i>	Constructs a new component with the same parameters and random state.
<i>default_parameters</i>	Returns the default parameters for this component.
<i>describe</i>	Describe a component and its parameters
<i>feature_importance</i>	Feature importance only works with linear kernels.
<i>fit</i>	Fits component to data
<i>load</i>	Loads component at file path
<i>needs_fitting</i>	Returns boolean determining if component needs fitting before
<i>parameters</i>	Returns the parameters which were used to initialize the component
<i>predict</i>	Make predictions using selected features.
<i>predict_proba</i>	Make probability estimates for labels.

continues on next page

Table 371 – continued from previous page

<i>save</i>	Saves component at file path
<b>clone</b> ( <i>self</i> )	
Constructs a new component with the same parameters and random state.	
<b>Returns</b> A new instance of this component with identical parameters and random state.	
<b>default_parameters</b> ( <i>cls</i> )	
Returns the default parameters for this component.	
Our convention is that <code>Component.default_parameters == Component().parameters</code> .	
<b>Returns</b> default parameters for this component.	
<b>Return type</b> dict	
<b>describe</b> ( <i>self</i> , <i>print_name=False</i> , <i>return_dict=False</i> )	
Describe a component and its parameters	
<b>Parameters</b>	
<ul style="list-style-type: none"> <li>• <b>print_name</b> (<i>bool</i>, <i>optional</i>) – whether to print name of component</li> <li>• <b>return_dict</b> (<i>bool</i>, <i>optional</i>) – whether to return description as dictionary in the format {"name": name, "parameters": parameters}</li> </ul>	
<b>Returns</b> prints and returns dictionary	
<b>Return type</b> None or dict	
<b>property feature_importance</b> ( <i>self</i> )	
Feature importance only works with linear kernels. If the kernel isn't linear, we return a numpy array of zeros	
<b>fit</b> ( <i>self</i> , <i>X</i> , <i>y=None</i> )	
Fits component to data	
<b>Parameters</b>	
<ul style="list-style-type: none"> <li>• <b>X</b> (<i>list</i>, <i>pd.DataFrame</i> or <i>np.ndarray</i>) – The input training data of shape [n_samples, n_features]</li> <li>• <b>y</b> (<i>list</i>, <i>pd.Series</i>, <i>np.ndarray</i>, <i>optional</i>) – The target training data of length [n_samples]</li> </ul>	
<b>Returns</b> self	
<b>static load</b> ( <i>file_path</i> )	
Loads component at file path	
<b>Parameters</b> <b>file_path</b> ( <i>str</i> ) – Location to load file	
<b>Returns</b> ComponentBase object	
<b>needs_fitting</b> ( <i>self</i> )	
Returns boolean determining if component needs fitting before calling predict, predict_proba, transform, or feature_importances. This can be overridden to False for components that do not need to be fit or whose fit methods do nothing.	
<b>property parameters</b> ( <i>self</i> )	
Returns the parameters which were used to initialize the component	
<b>predict</b> ( <i>self</i> , <i>X</i> )	
Make predictions using selected features.	

**Parameters** **X** (*pd.DataFrame*, *np.ndarray*) – Data of shape [n\_samples, n\_features]

**Returns** Predicted values

**Return type** *pd.Series*

**predict\_proba** (*self*, *X*)

Make probability estimates for labels.

**Parameters** **X** (*pd.DataFrame*, or *np.ndarray*) – Features

**Returns** Probability estimates

**Return type** *pd.Series*

**save** (*self*, *file\_path*, *pickle\_protocol=cloudpickle.DEFAULT\_PROTOCOL*)

Saves component at file path

**Parameters**

- **file\_path** (*str*) – Location to save file
- **pickle\_protocol** (*int*) – The pickle data stream format.

**Returns** None

**class** evalml.pipelines.components.estimators.regressors.**TimeSeriesBaselineEstimator** (*gap=1*,  
*random\_seed*,  
*\*\*kwargs*)

Time series estimator that predicts using the naive forecasting approach.

This is useful as a simple baseline estimator for time series problems.

**Parameters**

- **gap** (*int*) – Gap between prediction date and target date and must be a positive integer. If gap is 0, target date will be shifted ahead by 1 time period. Defaults to 1.
- **random\_seed** (*int*) – Seed for the random number generator. Defaults to 0.

**Attributes**

<b>hyper-parameter_ranges</b>	{}
<b>model_family</b>	ModelFamily.BASELINE
<b>modifies_features</b>	True
<b>modifies_target</b>	False
<b>name</b>	Time Series Baseline Estimator
<b>predict_uses_y</b>	True
<b>supported_problem_types</b>	[ProblemTypes.TIME_SERIES_REGRESSION, ProblemTypes.TIME_SERIES_BINARY, ProblemTypes.TIME_SERIES_MULTICLASS,]

**Methods**

<code>clone</code>	Constructs a new component with the same parameters and random state.
<code>default_parameters</code>	Returns the default parameters for this component.
<code>describe</code>	Describe a component and its parameters
<code>feature_importance</code>	Returns importance associated with each feature.
<code>fit</code>	Fits component to data
<code>load</code>	Loads component at file path
<code>needs_fitting</code>	Returns boolean determining if component needs fitting before
<code>parameters</code>	Returns the parameters which were used to initialize the component
<code>predict</code>	Make predictions using selected features.
<code>predict_proba</code>	Make probability estimates for labels.
<code>save</code>	Saves component at file path

**clone** (*self*)

Constructs a new component with the same parameters and random state.

**Returns** A new instance of this component with identical parameters and random state.

**default\_parameters** (*cls*)

Returns the default parameters for this component.

Our convention is that `Component.default_parameters == Component().parameters`.

**Returns** default parameters for this component.

**Return type** dict

**describe** (*self*, *print\_name=False*, *return\_dict=False*)

Describe a component and its parameters

**Parameters**

- **print\_name** (*bool*, *optional*) – whether to print name of component
- **return\_dict** (*bool*, *optional*) – whether to return description as dictionary in the format {"name": name, "parameters": parameters}

**Returns** prints and returns dictionary

**Return type** None or dict

**property feature\_importance** (*self*)

Returns importance associated with each feature.

Since baseline estimators do not use input features to calculate predictions, returns an array of zeroes.

**Returns** an array of zeroes

**Return type** np.ndarray (float)

**fit** (*self*, *X*, *y=None*)

Fits component to data

**Parameters**

- **X** (*list*, *pd.DataFrame* or *np.ndarray*) – The input training data of shape [n\_samples, n\_features]
- **y** (*list*, *pd.Series*, *np.ndarray*, *optional*) – The target training data of length [n\_samples]



**Returns** `self`

**static load** (*file\_path*)

Loads component at file path

**Parameters** `file_path` (*str*) – Location to load file

**Returns** ComponentBase object

**needs\_fitting** (*self*)

Returns boolean determining if component needs fitting before calling predict, predict\_proba, transform, or feature\_importances. This can be overridden to False for components that do not need to be fit or whose fit methods do nothing.

**property parameters** (*self*)

Returns the parameters which were used to initialize the component

**predict** (*self*, *X*, *y=None*)

Make predictions using selected features.

**Parameters** `X` (*pd.DataFrame*, *np.ndarray*) – Data of shape [n\_samples, n\_features]

**Returns** Predicted values

**Return type** `pd.Series`

**predict\_proba** (*self*, *X*, *y=None*)

Make probability estimates for labels.

**Parameters** `X` (*pd.DataFrame*, or *np.ndarray*) – Features

**Returns** Probability estimates

**Return type** `pd.Series`

**save** (*self*, *file\_path*, *pickle\_protocol=cloudpickle.DEFAULT\_PROTOCOL*)

Saves component at file path

**Parameters**

- **file\_path** (*str*) – Location to save file
- **pickle\_protocol** (*int*) – The pickle data stream format.

**Returns** `None`

```
class evalml.pipelines.components.estimators.regressors.XGBoostRegressor (eta=0.1,
                                                                    max_depth=6,
                                                                    min_child_weight=1,
                                                                    n_estimators=100,
                                                                    random_state=0,
                                                                    n_jobs=-1,
                                                                    **kwargs)
```

XGBoost Regressor.

**Parameters**

- **eta** (*float*) – Boosting learning rate. Defaults to 0.1.
- **max\_depth** (*int*) – Maximum tree depth for base learners. Defaults to 6.
- **min\_child\_weight** (*float*) – Minimum sum of instance weight (hessian) needed in a child. Defaults to 1.0

- **n\_estimators** (*int*) – Number of gradient boosted trees. Equivalent to number of boosting rounds. Defaults to 100.
- **random\_seed** (*int*) – Seed for the random number generator. Defaults to 0.
- **n\_jobs** (*int*) – Number of parallel threads used to run xgboost. Note that creating thread contention will significantly slow down the algorithm. Defaults to -1.

#### Attributes

<b>hyper-parameter_ranges</b>	{ "eta": Real(0.000001, 1), "max_depth": Integer(1, 20), "min_child_weight": Real(1, 10), "n_estimators": Integer(1, 1000), }
<b>model_family</b>	ModelFamily.XGBOOST
<b>modifies_features</b>	True
<b>modifies_target</b>	False
<b>name</b>	XGBoost Regressor
<b>pre-dict_uses_y</b>	False
<b>SEED_MAX</b>	None
<b>SEED_MIN</b>	None
<b>supported_problem_types</b>	[ ProblemTypes.REGRESSION, ProblemTypes.TIME_SERIES_REGRESSION, ]

#### Methods

<i>clone</i>	Constructs a new component with the same parameters and random state.
<i>default_parameters</i>	Returns the default parameters for this component.
<i>describe</i>	Describe a component and its parameters
<i>feature_importance</i>	Returns importance associated with each feature.
<i>fit</i>	Fits component to data
<i>load</i>	Loads component at file path
<i>needs_fitting</i>	Returns boolean determining if component needs fitting before
<i>parameters</i>	Returns the parameters which were used to initialize the component
<i>predict</i>	Make predictions using selected features.
<i>predict_proba</i>	Make probability estimates for labels.
<i>save</i>	Saves component at file path

#### **clone** (*self*)

Constructs a new component with the same parameters and random state.

**Returns** A new instance of this component with identical parameters and random state.

#### **default\_parameters** (*cls*)

Returns the default parameters for this component.

Our convention is that `Component.default_parameters == Component().parameters`.

**Returns** default parameters for this component.

**Return type** dict

**describe** (*self*, *print\_name=False*, *return\_dict=False*)

Describe a component and its parameters

**Parameters**

- **print\_name** (*bool*, *optional*) – whether to print name of component
- **return\_dict** (*bool*, *optional*) – whether to return description as dictionary in the format {"name": name, "parameters": parameters}

**Returns** prints and returns dictionary

**Return type** None or dict

**property feature\_importance** (*self*)

Returns importance associated with each feature.

**Returns** Importance associated with each feature

**Return type** np.ndarray

**fit** (*self*, *X*, *y=None*)

Fits component to data

**Parameters**

- **X** (*list*, *pd.DataFrame* or *np.ndarray*) – The input training data of shape [n\_samples, n\_features]
- **y** (*list*, *pd.Series*, *np.ndarray*, *optional*) – The target training data of length [n\_samples]

**Returns** self

**static load** (*file\_path*)

Loads component at file path

**Parameters** **file\_path** (*str*) – Location to load file

**Returns** ComponentBase object

**needs\_fitting** (*self*)

Returns boolean determining if component needs fitting before calling predict, predict\_proba, transform, or feature\_importances. This can be overridden to False for components that do not need to be fit or whose fit methods do nothing.

**property parameters** (*self*)

Returns the parameters which were used to initialize the component

**predict** (*self*, *X*)

Make predictions using selected features.

**Parameters** **X** (*pd.DataFrame*, or *np.ndarray*) – Data of shape [n\_samples, n\_features]

**Returns** Predicted values

**Return type** pd.Series

**predict\_proba** (*self*, *X*)

Make probability estimates for labels.

**Parameters** **X** (*pd.DataFrame*, or *np.ndarray*) – Features

**Returns** Probability estimates

**Return type** pd.Series

**save** (*self*, *file\_path*, *pickle\_protocol*=*cloudpickle.DEFAULT\_PROTOCOL*)  
Saves component at file path

**Parameters**

- **file\_path** (*str*) – Location to save file
- **pickle\_protocol** (*int*) – The pickle data stream format.

**Returns** None

## Submodules

### estimator

## Module Contents

## Classes Summary

---

<i>Estimator</i>	A component that fits and predicts given data.
------------------	--

---

## Contents

**class** evalml.pipelines.components.estimators.estimator.**Estimator** (*parameters*=None, *component\_obj*=None, *random\_seed*=0, *\*\*kwargs*)

A component that fits and predicts given data.

To implement a new Estimator, define your own class which is a subclass of Estimator, including a name and a list of acceptable ranges for any parameters to be tuned during the automl search (hyperparameters). Define an `__init__` method which sets up any necessary state and objects. Make sure your `__init__` only uses standard keyword arguments and calls `super().__init__()` with a parameters dict. You may also override the `fit`, `transform`, `fit_transform` and other methods in this class if appropriate.

To see some examples, check out the definitions of any Estimator component.

**Parameters**

- **parameters** (*dict*) – Dictionary of parameters for the component. Defaults to None.
- **component\_obj** (*obj*) – Third-party objects useful in component implementation. Defaults to None.
- **random\_seed** (*int*) – Seed for the random number generator. Defaults to 0.

**Attributes**

<b>model_family</b>	ModelFamily.NONE
<b>modifies_features</b>	True
<b>modifies_target</b>	False
<b>pre_dict_uses_y</b>	False

## Methods

<i>clone</i>	Constructs a new component with the same parameters and random state.
<i>default_parameters</i>	Returns the default parameters for this component.
<i>describe</i>	Describe a component and its parameters
<i>feature_importance</i>	Returns importance associated with each feature.
<i>fit</i>	Fits component to data
<i>load</i>	Loads component at file path
<i>name</i>	Returns string name of this component
<i>needs_fitting</i>	Returns boolean determining if component needs fitting before
<i>parameters</i>	Returns the parameters which were used to initialize the component
<i>predict</i>	Make predictions using selected features.
<i>predict_proba</i>	Make probability estimates for labels.
<i>save</i>	Saves component at file path
<i>supported_problem_types</i>	Problem types this estimator supports

### **clone** (*self*)

Constructs a new component with the same parameters and random state.

**Returns** A new instance of this component with identical parameters and random state.

### **default\_parameters** (*cls*)

Returns the default parameters for this component.

Our convention is that `Component.default_parameters == Component().parameters`.

**Returns** default parameters for this component.

**Return type** dict

### **describe** (*self*, *print\_name=False*, *return\_dict=False*)

Describe a component and its parameters

#### Parameters

- **print\_name** (*bool*, *optional*) – whether to print name of component
- **return\_dict** (*bool*, *optional*) – whether to return description as dictionary in the format {"name": name, "parameters": parameters}

**Returns** prints and returns dictionary

**Return type** None or dict

### **property feature\_importance** (*self*)

Returns importance associated with each feature.

**Returns** Importance associated with each feature

**Return type** np.ndarray

**fit** (*self*, *X*, *y=None*)

Fits component to data

**Parameters**

- **X** (*list*, *pd.DataFrame* or *np.ndarray*) – The input training data of shape [n\_samples, n\_features]
- **y** (*list*, *pd.Series*, *np.ndarray*, *optional*) – The target training data of length [n\_samples]

**Returns** self

**static load** (*file\_path*)

Loads component at file path

**Parameters** **file\_path** (*str*) – Location to load file

**Returns** ComponentBase object

**property name** (*cls*)

Returns string name of this component

**needs\_fitting** (*self*)

Returns boolean determining if component needs fitting before calling predict, predict\_proba, transform, or feature\_importances. This can be overridden to False for components that do not need to be fit or whose fit methods do nothing.

**property parameters** (*self*)

Returns the parameters which were used to initialize the component

**predict** (*self*, *X*)

Make predictions using selected features.

**Parameters** **X** (*pd.DataFrame*, or *np.ndarray*) – Data of shape [n\_samples, n\_features]

**Returns** Predicted values

**Return type** pd.Series

**predict\_proba** (*self*, *X*)

Make probability estimates for labels.

**Parameters** **X** (*pd.DataFrame*, or *np.ndarray*) – Features

**Returns** Probability estimates

**Return type** pd.Series

**save** (*self*, *file\_path*, *pickle\_protocol=cloudpickle.DEFAULT\_PROTOCOL*)

Saves component at file path

**Parameters**

- **file\_path** (*str*) – Location to save file
- **pickle\_protocol** (*int*) – The pickle data stream format.

**Returns** None

**property supported\_problem\_types** (*cls*)

Problem types this estimator supports

## Package Contents

### Classes Summary

<i>ARIMAREgressor</i>	Autoregressive Integrated Moving Average Model.
<i>BaselineClassifier</i>	Classifier that predicts using the specified strategy.
<i>BaselineRegressor</i>	Baseline regressor that uses a simple strategy to make predictions.
<i>CatBoostClassifier</i>	CatBoost Classifier, a classifier that uses gradient-boosting on decision trees.
<i>CatBoostRegressor</i>	CatBoost Regressor, a regressor that uses gradient-boosting on decision trees.
<i>DecisionTreeClassifier</i>	Decision Tree Classifier.
<i>DecisionTreeRegressor</i>	Decision Tree Regressor.
<i>ElasticNetClassifier</i>	Elastic Net Classifier. Uses Logistic Regression with elasticnet penalty as the base estimator.
<i>ElasticNetRegressor</i>	Elastic Net Regressor.
<i>Estimator</i>	A component that fits and predicts given data.
<i>ExtraTreesClassifier</i>	Extra Trees Classifier.
<i>ExtraTreesRegressor</i>	Extra Trees Regressor.
<i>KNeighborsClassifier</i>	K-Nearest Neighbors Classifier.
<i>LightGBMClassifier</i>	LightGBM Classifier.
<i>LightGBMRegressor</i>	LightGBM Regressor.
<i>LinearRegressor</i>	Linear Regressor.
<i>LogisticRegressionClassifier</i>	Logistic Regression Classifier.
<i>ProphetRegressor</i>	Prophet is a procedure for forecasting time series data based on an additive model where non-linear trends are fit with yearly, weekly, and daily seasonality, plus holiday effects.
<i>RandomForestClassifier</i>	Random Forest Classifier.
<i>RandomForestRegressor</i>	Random Forest Regressor.
<i>SVMClassifier</i>	Support Vector Machine Classifier.
<i>SVMRegressor</i>	Support Vector Machine Regressor.
<i>TimeSeriesBaselineEstimator</i>	Time series estimator that predicts using the naive forecasting approach.
<i>XGBoostClassifier</i>	XGBoost Classifier.
<i>XGBoostRegressor</i>	XGBoost Regressor.

## Contents

```
class evalml.pipelines.components.estimators.ARIMAREgressor (date_index=None,
                                                            trend=None,
                                                            start_p=2,
                                                            d=0, start_q=2,
                                                            max_p=5, max_d=2,
                                                            max_q=5, seasonal=True,
                                                            n_jobs=-1, random_seed=0,
                                                            **kwargs)
```

Autoregressive Integrated Moving Average Model. The three parameters (p, d, q) are the AR order, the degree of differencing, and the MA order. More information here: [https://www.statsmodels.org/devel/generated/statsmodels.tsa.arima\\_model.ARIMA.html](https://www.statsmodels.org/devel/generated/statsmodels.tsa.arima_model.ARIMA.html)

Currently ARIMAREgressor isn't supported via conda install. It's recommended that it be installed via PyPI.

### Parameters

- **date\_index** (*str*) – Specifies the name of the column in X that provides the datetime objects. Defaults to None.
- **trend** (*str*) – Controls the deterministic trend. Options are ['n', 'c', 't', 'ct'] where 'c' is a constant term, 't' indicates a linear trend, and 'ct' is both. Can also be an iterable when defining a polynomial, such as [1, 1, 0, 1].
- **start\_p** (*int*) – Minimum Autoregressive order. Defaults to 2.
- **d** (*int*) – Minimum Differencing degree. Defaults to 0.
- **start\_q** (*int*) – Minimum Moving Average order. Defaults to 2.
- **max\_p** (*int*) – Maximum Autoregressive order. Defaults to 5.
- **max\_d** (*int*) – Maximum Differencing degree. Defaults to 2.
- **max\_q** (*int*) – Maximum Moving Average order. Defaults to 5.
- **seasonal** (*boolean*) – Whether to fit a seasonal model to ARIMA. Defaults to True.
- **n\_jobs** (*int or None*) – Non-negative integer describing level of parallelism used for pipelines. Defaults to -1.
- **random\_seed** (*int*) – Seed for the random number generator. Defaults to 0.

### Attributes



<b>hyper-parameter_ranges</b>	{ "start_p": Integer(1, 3), "d": Integer(0, 2), "start_q": Integer(1, 3), "max_p": Integer(3, 10), "max_d": Integer(2, 5), "max_q": Integer(3, 10), "seasonal": [True, False], }
<b>model_family</b>	ModelFamily.ARIMA
<b>modifies_features</b>	True
<b>modifies_target</b>	False
<b>name</b>	ARIMA Regressor
<b>pre-dict_uses_y</b>	False
<b>supported_problem_types</b>	[ProblemTypes.TIME_SERIES_REGRESSION]

## Methods

<i>clone</i>	Constructs a new component with the same parameters and random state.
<i>default_parameters</i>	Returns the default parameters for this component.
<i>describe</i>	Describe a component and its parameters
<i>feature_importance</i>	Returns array of 0's with a length of 1 as feature_importance is not defined for ARIMA regressor.
<i>fit</i>	Fits component to data
<i>load</i>	Loads component at file path
<i>needs_fitting</i>	Returns boolean determining if component needs fitting before
<i>parameters</i>	Returns the parameters which were used to initialize the component
<i>predict</i>	Make predictions using selected features.
<i>predict_proba</i>	Make probability estimates for labels.
<i>save</i>	Saves component at file path

### **clone** (*self*)

Constructs a new component with the same parameters and random state.

**Returns** A new instance of this component with identical parameters and random state.

### **default\_parameters** (*cls*)

Returns the default parameters for this component.

Our convention is that `Component.default_parameters == Component().parameters`.

**Returns** default parameters for this component.

**Return type** dict

### **describe** (*self*, *print\_name=False*, *return\_dict=False*)

Describe a component and its parameters

#### Parameters

- **print\_name** (*bool*, *optional*) – whether to print name of component
- **return\_dict** (*bool*, *optional*) – whether to return description as dictionary in the format {"name": name, "parameters": parameters}

**Returns** prints and returns dictionary

**Return type** None or dict

**property feature\_importance** (*self*)

Returns array of 0's with a length of 1 as feature\_importance is not defined for ARIMA regressor.

**fit** (*self*, *X*, *y=None*)

Fits component to data

**Parameters**

- **X** (*list*, *pd.DataFrame* or *np.ndarray*) – The input training data of shape [n\_samples, n\_features]
- **y** (*list*, *pd.Series*, *np.ndarray*, *optional*) – The target training data of length [n\_samples]

**Returns** *self*

**static load** (*file\_path*)

Loads component at file path

**Parameters** **file\_path** (*str*) – Location to load file

**Returns** ComponentBase object

**needs\_fitting** (*self*)

Returns boolean determining if component needs fitting before calling predict, predict\_proba, transform, or feature\_importances. This can be overridden to False for components that do not need to be fit or whose fit methods do nothing.

**property parameters** (*self*)

Returns the parameters which were used to initialize the component

**predict** (*self*, *X*, *y=None*)

Make predictions using selected features.

**Parameters** **X** (*pd.DataFrame*, *np.ndarray*) – Data of shape [n\_samples, n\_features]

**Returns** Predicted values

**Return type** *pd.Series*

**predict\_proba** (*self*, *X*)

Make probability estimates for labels.

**Parameters** **X** (*pd.DataFrame*, or *np.ndarray*) – Features

**Returns** Probability estimates

**Return type** *pd.Series*

**save** (*self*, *file\_path*, *pickle\_protocol=cloudpickle.DEFAULT\_PROTOCOL*)

Saves component at file path

**Parameters**

- **file\_path** (*str*) – Location to save file
- **pickle\_protocol** (*int*) – The pickle data stream format.

**Returns** None

```
class evalml.pipelines.components.estimators.BaselineClassifier (strategy='mode',
                                                                ran-
                                                                dom_seed=0,
                                                                **kwargs)
```

Classifier that predicts using the specified strategy.

This is useful as a simple baseline classifier to compare with other classifiers.

### Parameters

- **strategy** (*str*) – Method used to predict. Valid options are “mode”, “random” and “random\_weighted”. Defaults to “mode”.
- **random\_seed** (*int*) – Seed for the random number generator. Defaults to 0.

### Attributes

<b>hyper-parameter_ranges</b>	{}
<b>model_family</b>	ModelFamily.BASELINE
<b>modifies_features</b>	True
<b>modifies_target</b>	False
<b>name</b>	Baseline Classifier
<b>predict_uses_y</b>	False
<b>supported_problem_types</b>	[ProblemTypes.BINARY, ProblemTypes.MULTICLASS]

### Methods

<i>classes_</i>	Returns class labels. Will return None before fitting.
<i>clone</i>	Constructs a new component with the same parameters and random state.
<i>default_parameters</i>	Returns the default parameters for this component.
<i>describe</i>	Describe a component and its parameters
<i>feature_importance</i>	Returns importance associated with each feature. Since baseline classifiers do not use input features to calculate predictions, returns an array of zeroes.
<i>fit</i>	Fits component to data
<i>load</i>	Loads component at file path
<i>needs_fitting</i>	Returns boolean determining if component needs fitting before
<i>parameters</i>	Returns the parameters which were used to initialize the component
<i>predict</i>	Make predictions using selected features.
<i>predict_proba</i>	Make probability estimates for labels.
<i>save</i>	Saves component at file path

#### **property classes\_** (*self*)

Returns class labels. Will return None before fitting.

**Returns** Class names

**Return type** list[str] or list(float)

#### **clone** (*self*)

Constructs a new component with the same parameters and random state.

**Returns** A new instance of this component with identical parameters and random state.

**default\_parameters** (*cls*)

Returns the default parameters for this component.

Our convention is that `Component.default_parameters == Component().parameters`.

**Returns** default parameters for this component.

**Return type** dict

**describe** (*self*, *print\_name=False*, *return\_dict=False*)

Describe a component and its parameters

**Parameters**

- **print\_name** (*bool*, *optional*) – whether to print name of component
- **return\_dict** (*bool*, *optional*) – whether to return description as dictionary in the format {"name": name, "parameters": parameters}

**Returns** prints and returns dictionary

**Return type** None or dict

**property feature\_importance** (*self*)

Returns importance associated with each feature. Since baseline classifiers do not use input features to calculate predictions, returns an array of zeroes.

**Returns** An array of zeroes

**Return type** np.ndarray (float)

**fit** (*self*, *X*, *y=None*)

Fits component to data

**Parameters**

- **X** (*list*, *pd.DataFrame* or *np.ndarray*) – The input training data of shape [n\_samples, n\_features]
- **y** (*list*, *pd.Series*, *np.ndarray*, *optional*) – The target training data of length [n\_samples]

**Returns** self

**static load** (*file\_path*)

Loads component at file path

**Parameters** **file\_path** (*str*) – Location to load file

**Returns** ComponentBase object

**needs\_fitting** (*self*)

Returns boolean determining if component needs fitting before calling predict, predict\_proba, transform, or feature\_importances. This can be overridden to False for components that do not need to be fit or whose fit methods do nothing.

**property parameters** (*self*)

Returns the parameters which were used to initialize the component

**predict** (*self*, *X*)

Make predictions using selected features.

**Parameters** **X** (*pd.DataFrame*, *np.ndarray*) – Data of shape [n\_samples, n\_features]

**Returns** Predicted values

**Return type** pd.Series

**predict\_proba** (*self*, *X*)

Make probability estimates for labels.

**Parameters** **X** (*pd.DataFrame*, or *np.ndarray*) – Features

**Returns** Probability estimates

**Return type** *pd.Series*

**save** (*self*, *file\_path*, *pickle\_protocol=cloudpickle.DEFAULT\_PROTOCOL*)

Saves component at file path

**Parameters**

- **file\_path** (*str*) – Location to save file
- **pickle\_protocol** (*int*) – The pickle data stream format.

**Returns** *None*

**class** `evalml.pipelines.components.estimators.BaselineRegressor` (*strategy='mean'*,  
*random\_seed=0*,  
*\*\*kwargs*)

Baseline regressor that uses a simple strategy to make predictions. This is useful as a simple baseline regressor to compare with other regressors.

**Parameters**

- **strategy** (*str*) – Method used to predict. Valid options are “mean”, “median”. Defaults to “mean”.
- **random\_seed** (*int*) – Seed for the random number generator. Defaults to 0.

**Attributes**

<b>hyper-parameter_ranges</b>	{}
<b>model_family</b>	ModelFamily.BASELINE
<b>modifies_features</b>	True
<b>modifies_target</b>	False
<b>name</b>	Baseline Regressor
<b>predict_uses_y</b>	False
<b>supported_problem_types</b>	[ ProblemTypes.REGRESSION, ProblemTypes.TIME_SERIES_REGRESSION,]

**Methods**

<code>clone</code>	Constructs a new component with the same parameters and random state.
<code>default_parameters</code>	Returns the default parameters for this component.
<code>describe</code>	Describe a component and its parameters
<code>feature_importance</code>	Returns importance associated with each feature. Since baseline regressors do not use input features to calculate predictions, returns an array of zeroes.

continues on next page

Table 379 – continued from previous page

<i>fit</i>	Fits component to data
<i>load</i>	Loads component at file path
<i>needs_fitting</i>	Returns boolean determining if component needs fitting before
<i>parameters</i>	Returns the parameters which were used to initialize the component
<i>predict</i>	Make predictions using selected features.
<i>predict_proba</i>	Make probability estimates for labels.
<i>save</i>	Saves component at file path

**clone** (*self*)

Constructs a new component with the same parameters and random state.

**Returns** A new instance of this component with identical parameters and random state.

**default\_parameters** (*cls*)

Returns the default parameters for this component.

Our convention is that `Component.default_parameters == Component().parameters`.

**Returns** default parameters for this component.

**Return type** dict

**describe** (*self*, *print\_name=False*, *return\_dict=False*)

Describe a component and its parameters

**Parameters**

- **print\_name** (*bool*, *optional*) – whether to print name of component
- **return\_dict** (*bool*, *optional*) – whether to return description as dictionary in the format {"name": name, "parameters": parameters}

**Returns** prints and returns dictionary

**Return type** None or dict

**property feature\_importance** (*self*)

Returns importance associated with each feature. Since baseline regressors do not use input features to calculate predictions, returns an array of zeroes.

**Returns** An array of zeroes

**Return type** np.ndarray (float)

**fit** (*self*, *X*, *y=None*)

Fits component to data

**Parameters**

- **X** (*list*, *pd.DataFrame* or *np.ndarray*) – The input training data of shape [n\_samples, n\_features]
- **y** (*list*, *pd.Series*, *np.ndarray*, *optional*) – The target training data of length [n\_samples]

**Returns** self

**static load** (*file\_path*)

Loads component at file path

**Parameters** **file\_path** (*str*) – Location to load file

**Returns** ComponentBase object

**needs\_fitting** (*self*)

Returns boolean determining if component needs fitting before calling predict, predict\_proba, transform, or feature\_importances. This can be overridden to False for components that do not need to be fit or whose fit methods do nothing.

**property parameters** (*self*)

Returns the parameters which were used to initialize the component

**predict** (*self*, *X*)

Make predictions using selected features.

**Parameters** *X* (*pd.DataFrame*, *np.ndarray*) – Data of shape [n\_samples, n\_features]

**Returns** Predicted values

**Return type** *pd.Series*

**predict\_proba** (*self*, *X*)

Make probability estimates for labels.

**Parameters** *X* (*pd.DataFrame*, or *np.ndarray*) – Features

**Returns** Probability estimates

**Return type** *pd.Series*

**save** (*self*, *file\_path*, *pickle\_protocol=cloudpickle.DEFAULT\_PROTOCOL*)

Saves component at file path

**Parameters**

- **file\_path** (*str*) – Location to save file
- **pickle\_protocol** (*int*) – The pickle data stream format.

**Returns** None

```
class evalml.pipelines.components.estimators.CatBoostClassifier (n_estimators=10,
                                                                eta=0.03,
                                                                max_depth=6,
                                                                boot-
                                                                strap_type=None,
                                                                silent=True,
                                                                al-
                                                                low_writing_files=False,
                                                                ran-
                                                                dom_seed=0,
                                                                n_jobs=-1,
                                                                **kwargs)
```

CatBoost Classifier, a classifier that uses gradient-boosting on decision trees. CatBoost is an open-source library and natively supports categorical features.

For more information, check out <https://catboost.ai/>

**Parameters**

- **n\_estimators** (*float*) – The maximum number of trees to build. Defaults to 10.
- **eta** (*float*) – The learning rate. Defaults to 0.03.
- **max\_depth** (*int*) – The maximum tree depth for base learners. Defaults to 6.

- **bootstrap\_type** (*string*) – Defines the method for sampling the weights of objects. Available methods are ‘Bayesian’, ‘Bernoulli’, ‘MVS’. Defaults to None.
- **silent** (*boolean*) – Whether to use the “silent” logging mode. Defaults to True.
- **allow\_writing\_files** (*boolean*) – Whether to allow writing snapshot files while training. Defaults to False.
- **n\_jobs** (*int or None*) – Number of jobs to run in parallel. -1 uses all processes. Defaults to -1.
- **random\_seed** (*int*) – Seed for the random number generator. Defaults to 0.

#### Attributes

<b>hyper-parameter_ranges</b>	{ “n_estimators”: Integer(4, 100), “eta”: Real(0.000001, 1), “max_depth”: Integer(4, 10), }
<b>model_family</b>	ModelFamily.CATBOOST
<b>modifies_features</b>	True
<b>modifies_target</b>	False
<b>name</b>	CatBoost Classifier
<b>pre-dict_uses_y</b>	False
<b>supported_problem_types</b>	[ ProblemTypes.BINARY, ProblemTypes.MULTICLASS, ProblemTypes.TIME_SERIES_BINARY, ProblemTypes.TIME_SERIES_MULTICLASS, ]

#### Methods

<i>clone</i>	Constructs a new component with the same parameters and random state.
<i>default_parameters</i>	Returns the default parameters for this component.
<i>describe</i>	Describe a component and its parameters
<i>feature_importance</i>	Returns importance associated with each feature.
<i>fit</i>	Fits component to data
<i>load</i>	Loads component at file path
<i>needs_fitting</i>	Returns boolean determining if component needs fitting before
<i>parameters</i>	Returns the parameters which were used to initialize the component
<i>predict</i>	Make predictions using selected features.
<i>predict_proba</i>	Make probability estimates for labels.
<i>save</i>	Saves component at file path

#### **clone** (*self*)

Constructs a new component with the same parameters and random state.

**Returns** A new instance of this component with identical parameters and random state.

#### **default\_parameters** (*cls*)

Returns the default parameters for this component.

Our convention is that `Component.default_parameters == Component().parameters`.

**Returns** default parameters for this component.



**Return type** dict

**describe** (*self*, *print\_name=False*, *return\_dict=False*)

Describe a component and its parameters

**Parameters**

- **print\_name** (*bool*, *optional*) – whether to print name of component
- **return\_dict** (*bool*, *optional*) – whether to return description as dictionary in the format {"name": name, "parameters": parameters}

**Returns** prints and returns dictionary

**Return type** None or dict

**property feature\_importance** (*self*)

Returns importance associated with each feature.

**Returns** Importance associated with each feature

**Return type** np.ndarray

**fit** (*self*, *X*, *y=None*)

Fits component to data

**Parameters**

- **X** (*list*, *pd.DataFrame* or *np.ndarray*) – The input training data of shape [n\_samples, n\_features]
- **y** (*list*, *pd.Series*, *np.ndarray*, *optional*) – The target training data of length [n\_samples]

**Returns** self

**static load** (*file\_path*)

Loads component at file path

**Parameters** **file\_path** (*str*) – Location to load file

**Returns** ComponentBase object

**needs\_fitting** (*self*)

Returns boolean determining if component needs fitting before calling predict, predict\_proba, transform, or feature\_importances. This can be overridden to False for components that do not need to be fit or whose fit methods do nothing.

**property parameters** (*self*)

Returns the parameters which were used to initialize the component

**predict** (*self*, *X*)

Make predictions using selected features.

**Parameters** **X** (*pd.DataFrame*, *np.ndarray*) – Data of shape [n\_samples, n\_features]

**Returns** Predicted values

**Return type** pd.Series

**predict\_proba** (*self*, *X*)

Make probability estimates for labels.

**Parameters** **X** (*pd.DataFrame*, or *np.ndarray*) – Features

**Returns** Probability estimates

**Return type** `pd.Series`

**save** (*self*, *file\_path*, *pickle\_protocol*=`cloudpickle.DEFAULT_PROTOCOL`)  
Saves component at file path

**Parameters**

- **file\_path** (*str*) – Location to save file
- **pickle\_protocol** (*int*) – The pickle data stream format.

**Returns** `None`

```
class evalml.pipelines.components.estimators.CatBoostRegressor (n_estimators=10,  
                                                             eta=0.03,  
                                                             max_depth=6,  
                                                             boot-  
                                                             strap_type=None,  
                                                             silent=False, al-  
                                                             low_writing_files=False,  
                                                             ran-  
                                                             dom_seed=0,  
                                                             n_jobs=-1,  
                                                             **kwargs)
```

CatBoost Regressor, a regressor that uses gradient-boosting on decision trees. CatBoost is an open-source library and natively supports categorical features.

For more information, check out <https://catboost.ai/>

**Parameters**

- **n\_estimators** (*float*) – The maximum number of trees to build. Defaults to 10.
- **eta** (*float*) – The learning rate. Defaults to 0.03.
- **max\_depth** (*int*) – The maximum tree depth for base learners. Defaults to 6.
- **bootstrap\_type** (*string*) – Defines the method for sampling the weights of objects. Available methods are ‘Bayesian’, ‘Bernoulli’, ‘MVS’. Defaults to None.
- **silent** (*boolean*) – Whether to use the “silent” logging mode. Defaults to True.
- **allow\_writing\_files** (*boolean*) – Whether to allow writing snapshot files while training. Defaults to False.
- **n\_jobs** (*int or None*) – Number of jobs to run in parallel. -1 uses all processes. Defaults to -1.
- **random\_seed** (*int*) – Seed for the random number generator. Defaults to 0.

**Attributes**

<b>hyper-parameter_ranges</b>	{ “n_estimators”: Integer(4, 100), “eta”: Real(0.000001, 1), “max_depth”: Integer(4, 10), }
<b>model_family</b>	ModelFamily.CATBOOST
<b>modifies_features</b>	True
<b>modifies_target</b>	False
<b>name</b>	CatBoost Regressor
<b>pre-dict_uses_y</b>	False
<b>supported_problem_types</b>	[ ProblemTypes.REGRESSION, ProblemTypes.TIME_SERIES_REGRESSION, ]

## Methods

<i>clone</i>	Constructs a new component with the same parameters and random state.
<i>default_parameters</i>	Returns the default parameters for this component.
<i>describe</i>	Describe a component and its parameters
<i>feature_importance</i>	Returns importance associated with each feature.
<i>fit</i>	Fits component to data
<i>load</i>	Loads component at file path
<i>needs_fitting</i>	Returns boolean determining if component needs fitting before
<i>parameters</i>	Returns the parameters which were used to initialize the component
<i>predict</i>	Make predictions using selected features.
<i>predict_proba</i>	Make probability estimates for labels.
<i>save</i>	Saves component at file path

### **clone** (*self*)

Constructs a new component with the same parameters and random state.

**Returns** A new instance of this component with identical parameters and random state.

### **default\_parameters** (*cls*)

Returns the default parameters for this component.

Our convention is that `Component.default_parameters == Component().parameters`.

**Returns** default parameters for this component.

**Return type** dict

### **describe** (*self*, *print\_name=False*, *return\_dict=False*)

Describe a component and its parameters

#### Parameters

- **print\_name** (*bool*, *optional*) – whether to print name of component
- **return\_dict** (*bool*, *optional*) – whether to return description as dictionary in the format {“name”: name, “parameters”: parameters}

**Returns** prints and returns dictionary

**Return type** None or dict

**property feature\_importance** (*self*)

Returns importance associated with each feature.

**Returns** Importance associated with each feature

**Return type** np.ndarray

**fit** (*self*, *X*, *y=None*)

Fits component to data

**Parameters**

- **X** (*list*, *pd.DataFrame* or *np.ndarray*) – The input training data of shape [n\_samples, n\_features]
- **y** (*list*, *pd.Series*, *np.ndarray*, *optional*) – The target training data of length [n\_samples]

**Returns** self

**static load** (*file\_path*)

Loads component at file path

**Parameters** **file\_path** (*str*) – Location to load file

**Returns** ComponentBase object

**needs\_fitting** (*self*)

Returns boolean determining if component needs fitting before calling predict, predict\_proba, transform, or feature\_importances. This can be overridden to False for components that do not need to be fit or whose fit methods do nothing.

**property parameters** (*self*)

Returns the parameters which were used to initialize the component

**predict** (*self*, *X*)

Make predictions using selected features.

**Parameters** **X** (*pd.DataFrame*, *np.ndarray*) – Data of shape [n\_samples, n\_features]

**Returns** Predicted values

**Return type** pd.Series

**predict\_proba** (*self*, *X*)

Make probability estimates for labels.

**Parameters** **X** (*pd.DataFrame*, or *np.ndarray*) – Features

**Returns** Probability estimates

**Return type** pd.Series

**save** (*self*, *file\_path*, *pickle\_protocol=cloudpickle.DEFAULT\_PROTOCOL*)

Saves component at file path

**Parameters**

- **file\_path** (*str*) – Location to save file
- **pickle\_protocol** (*int*) – The pickle data stream format.

**Returns** None

```
class evalml.pipelines.components.estimators.DecisionTreeClassifier(criterion='gini',  
                                                                    max_features='auto',  
                                                                    max_depth=6,  
                                                                    min_samples_split=2,  
                                                                    min_weight_fraction_leaf=0.0,  
                                                                    ran-  
                                                                    dom_seed=0,  
                                                                    **kwargs)
```

Decision Tree Classifier.

### Parameters

- **criterion** (*{ "gini", "entropy" }*) – The function to measure the quality of a split. Supported criteria are “gini” for the Gini impurity and “entropy” for the information gain. Defaults to “gini”.
- **max\_features** (*int, float or { "auto", "sqrt", "log2" }*) – The number of features to consider when looking for the best split:
  - If int, then consider max\_features features at each split.
  - If float, then max\_features is a fraction and  $\text{int}(\text{max\_features} * \text{n\_features})$  features are considered at each split.
  - If “auto”, then  $\text{max\_features} = \sqrt{\text{n\_features}}$ .
  - If “sqrt”, then  $\text{max\_features} = \sqrt{\text{n\_features}}$ .
  - If “log2”, then  $\text{max\_features} = \log_2(\text{n\_features})$ .
  - If None, then  $\text{max\_features} = \text{n\_features}$ .

The search for a split does not stop until at least one valid partition of the node samples is found, even if it requires to effectively inspect more than max\_features features. Defaults to “auto”.

- **max\_depth** (*int*) – The maximum depth of the tree. Defaults to 6.
- **min\_samples\_split** (*int or float*) – The minimum number of samples required to split an internal node:
  - If int, then consider min\_samples\_split as the minimum number.
  - If float, then min\_samples\_split is a fraction and  $\text{ceil}(\text{min\_samples\_split} * \text{n\_samples})$  are the minimum number of samples for each split.

Defaults to 2.

- **min\_weight\_fraction\_leaf** (*float*) – The minimum weighted fraction of the sum total of weights (of all the input samples) required to be at a leaf node. Defaults to 0.0.
- **random\_seed** (*int*) – Seed for the random number generator. Defaults to 0.

### Attributes

<b>hyper-parameter_ranges</b>	{ "criterion": ["gini", "entropy"], "max_features": ["auto", "sqrt", "log2"], "max_depth": Integer(4, 10), }
<b>model_family</b>	ModelFamily.DECISION_TREE
<b>modifies_features</b>	True
<b>modifies_target</b>	False
<b>name</b>	Decision Tree Classifier
<b>pre-dict_uses_y</b>	False
<b>supported_problem_types</b>	[ ProblemTypes.BINARY, ProblemTypes.MULTICLASS, ProblemTypes.TIME_SERIES_BINARY, ProblemTypes.TIME_SERIES_MULTICLASS, ]

## Methods

<i>clone</i>	Constructs a new component with the same parameters and random state.
<i>default_parameters</i>	Returns the default parameters for this component.
<i>describe</i>	Describe a component and its parameters
<i>feature_importance</i>	Returns importance associated with each feature.
<i>fit</i>	Fits component to data
<i>load</i>	Loads component at file path
<i>needs_fitting</i>	Returns boolean determining if component needs fitting before
<i>parameters</i>	Returns the parameters which were used to initialize the component
<i>predict</i>	Make predictions using selected features.
<i>predict_proba</i>	Make probability estimates for labels.
<i>save</i>	Saves component at file path

### **clone** (*self*)

Constructs a new component with the same parameters and random state.

**Returns** A new instance of this component with identical parameters and random state.

### **default\_parameters** (*cls*)

Returns the default parameters for this component.

Our convention is that `Component.default_parameters == Component().parameters`.

**Returns** default parameters for this component.

**Return type** dict

### **describe** (*self*, *print\_name=False*, *return\_dict=False*)

Describe a component and its parameters

#### Parameters

- **print\_name** (*bool*, *optional*) – whether to print name of component
- **return\_dict** (*bool*, *optional*) – whether to return description as dictionary in the format {"name": name, "parameters": parameters}

**Returns** prints and returns dictionary

**Return type** None or dict

**property feature\_importance** (*self*)

Returns importance associated with each feature.

**Returns** Importance associated with each feature

**Return type** np.ndarray

**fit** (*self*, *X*, *y=None*)

Fits component to data

**Parameters**

- **X** (*list*, *pd.DataFrame* or *np.ndarray*) – The input training data of shape [n\_samples, n\_features]
- **y** (*list*, *pd.Series*, *np.ndarray*, *optional*) – The target training data of length [n\_samples]

**Returns** self

**static load** (*file\_path*)

Loads component at file path

**Parameters** **file\_path** (*str*) – Location to load file

**Returns** ComponentBase object

**needs\_fitting** (*self*)

Returns boolean determining if component needs fitting before calling predict, predict\_proba, transform, or feature\_importances. This can be overridden to False for components that do not need to be fit or whose fit methods do nothing.

**property parameters** (*self*)

Returns the parameters which were used to initialize the component

**predict** (*self*, *X*)

Make predictions using selected features.

**Parameters** **X** (*pd.DataFrame*, *np.ndarray*) – Data of shape [n\_samples, n\_features]

**Returns** Predicted values

**Return type** pd.Series

**predict\_proba** (*self*, *X*)

Make probability estimates for labels.

**Parameters** **X** (*pd.DataFrame*, or *np.ndarray*) – Features

**Returns** Probability estimates

**Return type** pd.Series

**save** (*self*, *file\_path*, *pickle\_protocol=cloudpickle.DEFAULT\_PROTOCOL*)

Saves component at file path

**Parameters**

- **file\_path** (*str*) – Location to save file
- **pickle\_protocol** (*int*) – The pickle data stream format.

**Returns** None

```
class evalml.pipelines.components.estimators.DecisionTreeRegressor (criterion='mse',  
                                                                    max_features='auto',  
                                                                    max_depth=6,  
                                                                    min_samples_split=2,  
                                                                    min_weight_fraction_leaf=0.0,  
                                                                    ran-  
                                                                    dom_seed=0,  
                                                                    **kwargs)
```

Decision Tree Regressor.

### Parameters

- **criterion** (*{ "mse", "friedman\_mse", "mae", "poisson" }*) – The function to measure the quality of a split. Supported criteria are:
  - “mse” for the mean squared error, which is equal to variance reduction as feature selection criterion and minimizes the L2 loss using the mean of each terminal node
  - “friedman\_mse”, which uses mean squared error with Friedman’s improvement score for potential splits
  - “mae” for the mean absolute error, which minimizes the L1 loss using the median of each terminal node,
  - “poisson” which uses reduction in Poisson deviance to find splits.
- **max\_features** (*int, float or { "auto", "sqrt", "log2" }*) – The number of features to consider when looking for the best split:
  - If int, then consider max\_features features at each split.
  - If float, then max\_features is a fraction and int(max\_features \* n\_features) features are considered at each split.
  - If “auto”, then max\_features=sqrt(n\_features).
  - If “sqrt”, then max\_features=sqrt(n\_features).
  - If “log2”, then max\_features=log2(n\_features).
  - If None, then max\_features = n\_features.

The search for a split does not stop until at least one valid partition of the node samples is found, even if it requires to effectively inspect more than max\_features features.

- **max\_depth** (*int*) – The maximum depth of the tree. Defaults to 6.
- **min\_samples\_split** (*int or float*) – The minimum number of samples required to split an internal node:
  - If int, then consider min\_samples\_split as the minimum number.
  - If float, then min\_samples\_split is a fraction and ceil(min\_samples\_split \* n\_samples) are the minimum number of samples for each split.

Defaults to 2.

- **min\_weight\_fraction\_leaf** (*float*) – The minimum weighted fraction of the sum total of weights (of all the input samples) required to be at a leaf node. Defaults to 0.0.
- **random\_seed** (*int*) – Seed for the random number generator. Defaults to 0.

### Attributes



<b>hyper-parameter_ranges</b>	{ "criterion": ["mse", "friedman_mse", "mae"], "max_features": ["auto", "sqrt", "log2"], "max_depth": Integer(4, 10), }
<b>model_family</b>	ModelFamily.DECISION_TREE
<b>modifies_features</b>	True
<b>modifies_target</b>	False
<b>name</b>	Decision Tree Regressor
<b>pre-dict_uses_y</b>	False
<b>supported_problem_types</b>	[ ProblemTypes.REGRESSION, ProblemTypes.TIME_SERIES_REGRESSION, ]

## Methods

<i>clone</i>	Constructs a new component with the same parameters and random state.
<i>default_parameters</i>	Returns the default parameters for this component.
<i>describe</i>	Describe a component and its parameters
<i>feature_importance</i>	Returns importance associated with each feature.
<i>fit</i>	Fits component to data
<i>load</i>	Loads component at file path
<i>needs_fitting</i>	Returns boolean determining if component needs fitting before
<i>parameters</i>	Returns the parameters which were used to initialize the component
<i>predict</i>	Make predictions using selected features.
<i>predict_proba</i>	Make probability estimates for labels.
<i>save</i>	Saves component at file path

### **clone** (*self*)

Constructs a new component with the same parameters and random state.

**Returns** A new instance of this component with identical parameters and random state.

### **default\_parameters** (*cls*)

Returns the default parameters for this component.

Our convention is that `Component.default_parameters == Component().parameters`.

**Returns** default parameters for this component.

**Return type** dict

### **describe** (*self*, *print\_name=False*, *return\_dict=False*)

Describe a component and its parameters

#### Parameters

- **print\_name** (*bool*, *optional*) – whether to print name of component
- **return\_dict** (*bool*, *optional*) – whether to return description as dictionary in the format {"name": name, "parameters": parameters}

**Returns** prints and returns dictionary

**Return type** None or dict

**property feature\_importance** (*self*)

Returns importance associated with each feature.

**Returns** Importance associated with each feature

**Return type** np.ndarray

**fit** (*self*, *X*, *y=None*)

Fits component to data

**Parameters**

- **X** (*list*, *pd.DataFrame* or *np.ndarray*) – The input training data of shape [n\_samples, n\_features]
- **y** (*list*, *pd.Series*, *np.ndarray*, *optional*) – The target training data of length [n\_samples]

**Returns** *self*

**static load** (*file\_path*)

Loads component at file path

**Parameters** **file\_path** (*str*) – Location to load file

**Returns** ComponentBase object

**needs\_fitting** (*self*)

Returns boolean determining if component needs fitting before calling predict, predict\_proba, transform, or feature\_importances. This can be overridden to False for components that do not need to be fit or whose fit methods do nothing.

**property parameters** (*self*)

Returns the parameters which were used to initialize the component

**predict** (*self*, *X*)

Make predictions using selected features.

**Parameters** **X** (*pd.DataFrame*, *np.ndarray*) – Data of shape [n\_samples, n\_features]

**Returns** Predicted values

**Return type** pd.Series

**predict\_proba** (*self*, *X*)

Make probability estimates for labels.

**Parameters** **X** (*pd.DataFrame*, or *np.ndarray*) – Features

**Returns** Probability estimates

**Return type** pd.Series

**save** (*self*, *file\_path*, *pickle\_protocol=cloudpickle.DEFAULT\_PROTOCOL*)

Saves component at file path

**Parameters**

- **file\_path** (*str*) – Location to save file
- **pickle\_protocol** (*int*) – The pickle data stream format.

**Returns** None

```
class evalml.pipelines.components.estimators.ElasticNetClassifier (penalty='elasticnet',
                                                                C=1.0,
                                                                l1_ratio=0.15,
                                                                multi_class='auto',
                                                                solver='saga',
                                                                n_jobs=-1,
                                                                random_seed=0,
                                                                **kwargs)
```

Elastic Net Classifier. Uses Logistic Regression with elasticnet penalty as the base estimator.

### Parameters

- **penalty** (`{ "l1", "l2", "elasticnet", "none" }`) – The norm used in penalization. Defaults to “elasticnet”.
- **C** (`float`) – Inverse of regularization strength. Must be a positive float. Defaults to 1.0.
- **l1\_ratio** (`float`) – The mixing parameter, with  $0 \leq \text{l1\_ratio} \leq 1$ . Only used if `penalty='elasticnet'`. Setting `l1_ratio=0` is equivalent to using `penalty='l2'`, while setting `l1_ratio=1` is equivalent to using `penalty='l1'`. For  $0 < \text{l1\_ratio} < 1$ , the penalty is a combination of L1 and L2. Defaults to 0.15.
- **multi\_class** (`{ "auto", "ovr", "multinomial" }`) – If the option chosen is “ovr”, then a binary problem is fit for each label. For “multinomial” the loss minimised is the multinomial loss fit across the entire probability distribution, even when the data is binary. “multinomial” is unavailable when `solver="liblinear"`. “auto” selects “ovr” if the data is binary, or if `solver="liblinear"`, and otherwise selects “multinomial”. Defaults to “auto”.
- **solver** (`{ "newton-cg", "lbfgs", "liblinear", "sag", "saga" }`) – Algorithm to use in the optimization problem. For small datasets, “liblinear” is a good choice, whereas “sag” and “saga” are faster for large ones. For multiclass problems, only “newton-cg”, “sag”, “saga” and “lbfgs” handle multinomial loss; “liblinear” is limited to one-versus-rest schemes.
  - “newton-cg”, “lbfgs”, “sag” and “saga” handle L2 or no penalty
  - “liblinear” and “saga” also handle L1 penalty
  - “saga” also supports “elasticnet” penalty
  - “liblinear” does not support setting `penalty='none'`
 Defaults to “saga”.
- **n\_jobs** (`int`) – Number of parallel threads used to run xgboost. Note that creating thread contention will significantly slow down the algorithm. Defaults to -1.
- **random\_seed** (`int`) – Seed for the random number generator. Defaults to 0.

### Attributes

<b>hyper-parameter_ranges</b>	{ "C": Real(0.01, 10), "l1_ratio": Real(0, 1)}
<b>model_family</b>	ModelFamily.LINEAR_MODEL
<b>modifies_features</b>	True
<b>modifies_target</b>	False
<b>name</b>	Elastic Net Classifier
<b>pre-dict_uses_y</b>	False
<b>supported_problem_types</b>	[ ProblemTypes.BINARY, ProblemTypes.MULTICLASS, ProblemTypes.TIME_SERIES_BINARY, ProblemTypes.TIME_SERIES_MULTICLASS,]

## Methods

<i>clone</i>	Constructs a new component with the same parameters and random state.
<i>default_parameters</i>	Returns the default parameters for this component.
<i>describe</i>	Describe a component and its parameters
<i>feature_importance</i>	Returns importance associated with each feature.
<i>fit</i>	Fits component to data
<i>load</i>	Loads component at file path
<i>needs_fitting</i>	Returns boolean determining if component needs fitting before
<i>parameters</i>	Returns the parameters which were used to initialize the component
<i>predict</i>	Make predictions using selected features.
<i>predict_proba</i>	Make probability estimates for labels.
<i>save</i>	Saves component at file path

### **clone** (*self*)

Constructs a new component with the same parameters and random state.

**Returns** A new instance of this component with identical parameters and random state.

### **default\_parameters** (*cls*)

Returns the default parameters for this component.

Our convention is that `Component.default_parameters == Component().parameters`.

**Returns** default parameters for this component.

**Return type** dict

### **describe** (*self*, *print\_name=False*, *return\_dict=False*)

Describe a component and its parameters

#### Parameters

- **print\_name** (*bool*, *optional*) – whether to print name of component
- **return\_dict** (*bool*, *optional*) – whether to return description as dictionary in the format {"name": name, "parameters": parameters}

**Returns** prints and returns dictionary

**Return type** None or dict

**property feature\_importance** (*self*)

Returns importance associated with each feature.

**Returns** Importance associated with each feature

**Return type** np.ndarray

**fit** (*self*, *X*, *y*)

Fits component to data

**Parameters**

- **X** (*list*, *pd.DataFrame* or *np.ndarray*) – The input training data of shape [n\_samples, n\_features]
- **y** (*list*, *pd.Series*, *np.ndarray*, *optional*) – The target training data of length [n\_samples]

**Returns** self

**static load** (*file\_path*)

Loads component at file path

**Parameters** **file\_path** (*str*) – Location to load file

**Returns** ComponentBase object

**needs\_fitting** (*self*)

Returns boolean determining if component needs fitting before calling predict, predict\_proba, transform, or feature\_importances. This can be overridden to False for components that do not need to be fit or whose fit methods do nothing.

**property parameters** (*self*)

Returns the parameters which were used to initialize the component

**predict** (*self*, *X*)

Make predictions using selected features.

**Parameters** **X** (*pd.DataFrame*, *np.ndarray*) – Data of shape [n\_samples, n\_features]

**Returns** Predicted values

**Return type** pd.Series

**predict\_proba** (*self*, *X*)

Make probability estimates for labels.

**Parameters** **X** (*pd.DataFrame*, or *np.ndarray*) – Features

**Returns** Probability estimates

**Return type** pd.Series

**save** (*self*, *file\_path*, *pickle\_protocol=cloudpickle.DEFAULT\_PROTOCOL*)

Saves component at file path

**Parameters**

- **file\_path** (*str*) – Location to save file
- **pickle\_protocol** (*int*) – The pickle data stream format.

**Returns** None

```
class evalml.pipelines.components.estimators.ElasticNetRegressor(alpha=0.0001,
                                                                l1_ratio=0.15,
                                                                max_iter=1000,
                                                                normalize=False,
                                                                random_seed=0,
                                                                **kwargs)
```

Elastic Net Regressor.

### Parameters

- **alpha** (*float*) – Constant that multiplies the penalty terms. Defaults to 0.0001.
- **l1\_ratio** (*float*) – The mixing parameter, with  $0 \leq \text{l1\_ratio} \leq 1$ . Only used if `penalty='elasticnet'`. Setting `l1_ratio=0` is equivalent to using `penalty='l2'`, while setting `l1_ratio=1` is equivalent to using `penalty='l1'`. For  $0 < \text{l1\_ratio} < 1$ , the penalty is a combination of L1 and L2. Defaults to 0.15.
- **max\_iter** (*int*) – The maximum number of iterations. Defaults to 1000.
- **normalize** (*boolean*) – If True, the regressors will be normalized before regression by subtracting the mean and dividing by the l2-norm. Defaults to False.
- **random\_seed** (*int*) – Seed for the random number generator. Defaults to 0.

### Attributes

<b>hyper-parameter_ranges</b>	{ "alpha": Real(0, 1), "l1_ratio": Real(0, 1), }
<b>model_family</b>	ModelFamily.LINEAR_MODEL
<b>modifies_features</b>	True
<b>modifies_target</b>	False
<b>name</b>	Elastic Net Regressor
<b>predict_uses_y</b>	False
<b>supported_problem_types</b>	[ ProblemTypes.REGRESSION, ProblemTypes.TIME_SERIES_REGRESSION, ]

### Methods

<code>clone</code>	Constructs a new component with the same parameters and random state.
<code>default_parameters</code>	Returns the default parameters for this component.
<code>describe</code>	Describe a component and its parameters
<code>feature_importance</code>	Returns importance associated with each feature.
<code>fit</code>	Fits component to data
<code>load</code>	Loads component at file path
<code>needs_fitting</code>	Returns boolean determining if component needs fitting before
<code>parameters</code>	Returns the parameters which were used to initialize the component

continues on next page

Table 385 – continued from previous page

<code>predict</code>	Make predictions using selected features.
<code>predict_proba</code>	Make probability estimates for labels.
<code>save</code>	Saves component at file path

**clone** (*self*)

Constructs a new component with the same parameters and random state.

**Returns** A new instance of this component with identical parameters and random state.

**default\_parameters** (*cls*)

Returns the default parameters for this component.

Our convention is that `Component.default_parameters == Component().parameters`.

**Returns** default parameters for this component.

**Return type** dict

**describe** (*self*, *print\_name=False*, *return\_dict=False*)

Describe a component and its parameters

**Parameters**

- **print\_name** (*bool*, *optional*) – whether to print name of component
- **return\_dict** (*bool*, *optional*) – whether to return description as dictionary in the format {"name": name, "parameters": parameters}

**Returns** prints and returns dictionary

**Return type** None or dict

**property feature\_importance** (*self*)

Returns importance associated with each feature.

**Returns** Importance associated with each feature

**Return type** np.ndarray

**fit** (*self*, *X*, *y=None*)

Fits component to data

**Parameters**

- **X** (*list*, *pd.DataFrame* or *np.ndarray*) – The input training data of shape [n\_samples, n\_features]
- **y** (*list*, *pd.Series*, *np.ndarray*, *optional*) – The target training data of length [n\_samples]

**Returns** self

**static load** (*file\_path*)

Loads component at file path

**Parameters** **file\_path** (*str*) – Location to load file

**Returns** ComponentBase object

**needs\_fitting** (*self*)

Returns boolean determining if component needs fitting before calling predict, predict\_proba, transform, or feature\_importances. This can be overridden to False for components that do not need to be fit or whose fit methods do nothing.

**property parameters** (*self*)

Returns the parameters which were used to initialize the component

**predict** (*self*, *X*)

Make predictions using selected features.

**Parameters** *X* (*pd.DataFrame*, *np.ndarray*) – Data of shape [n\_samples, n\_features]

**Returns** Predicted values

**Return type** *pd.Series*

**predict\_proba** (*self*, *X*)

Make probability estimates for labels.

**Parameters** *X* (*pd.DataFrame*, *or np.ndarray*) – Features

**Returns** Probability estimates

**Return type** *pd.Series*

**save** (*self*, *file\_path*, *pickle\_protocol=cloudpickle.DEFAULT\_PROTOCOL*)

Saves component at file path

**Parameters**

- **file\_path** (*str*) – Location to save file
- **pickle\_protocol** (*int*) – The pickle data stream format.

**Returns** None

**class** evalml.pipelines.components.estimators.**Estimator** (*parameters=None*, *component\_obj=None*, *random\_seed=0*, *\*\*kwargs*)

A component that fits and predicts given data.

To implement a new Estimator, define your own class which is a subclass of Estimator, including a name and a list of acceptable ranges for any parameters to be tuned during the automl search (hyperparameters). Define an `__init__` method which sets up any necessary state and objects. Make sure your `__init__` only uses standard keyword arguments and calls `super().__init__()` with a parameters dict. You may also override the `fit`, `transform`, `fit_transform` and other methods in this class if appropriate.

To see some examples, check out the definitions of any Estimator component.

**Parameters**

- **parameters** (*dict*) – Dictionary of parameters for the component. Defaults to None.
- **component\_obj** (*obj*) – Third-party objects useful in component implementation. Defaults to None.
- **random\_seed** (*int*) – Seed for the random number generator. Defaults to 0.

**Attributes**

<b>model_family</b>	ModelFamily.NONE
<b>modifies_features</b>	True
<b>modifies_target</b>	False
<b>pre-dict_uses_y</b>	False



## Methods

<code>clone</code>	Constructs a new component with the same parameters and random state.
<code>default_parameters</code>	Returns the default parameters for this component.
<code>describe</code>	Describe a component and its parameters
<code>feature_importance</code>	Returns importance associated with each feature.
<code>fit</code>	Fits component to data
<code>load</code>	Loads component at file path
<code>name</code>	Returns string name of this component
<code>needs_fitting</code>	Returns boolean determining if component needs fitting before
<code>parameters</code>	Returns the parameters which were used to initialize the component
<code>predict</code>	Make predictions using selected features.
<code>predict_proba</code>	Make probability estimates for labels.
<code>save</code>	Saves component at file path
<code>supported_problem_types</code>	Problem types this estimator supports

### **clone** (*self*)

Constructs a new component with the same parameters and random state.

**Returns** A new instance of this component with identical parameters and random state.

### **default\_parameters** (*cls*)

Returns the default parameters for this component.

Our convention is that `Component.default_parameters == Component().parameters`.

**Returns** default parameters for this component.

**Return type** dict

### **describe** (*self*, *print\_name=False*, *return\_dict=False*)

Describe a component and its parameters

#### Parameters

- **print\_name** (*bool*, *optional*) – whether to print name of component
- **return\_dict** (*bool*, *optional*) – whether to return description as dictionary in the format {"name": name, "parameters": parameters}

**Returns** prints and returns dictionary

**Return type** None or dict

### **property feature\_importance** (*self*)

Returns importance associated with each feature.

**Returns** Importance associated with each feature

**Return type** np.ndarray

### **fit** (*self*, *X*, *y=None*)

Fits component to data

#### Parameters

- **X** (*list*, *pd.DataFrame* or *np.ndarray*) – The input training data of shape [n\_samples, n\_features]

- **y** (*list, pd.Series, np.ndarray, optional*) – The target training data of length [n\_samples]

**Returns** self

**static load** (*file\_path*)

Loads component at file path

**Parameters** **file\_path** (*str*) – Location to load file

**Returns** ComponentBase object

**property name** (*cls*)

Returns string name of this component

**needs\_fitting** (*self*)

Returns boolean determining if component needs fitting before calling predict, predict\_proba, transform, or feature\_importances. This can be overridden to False for components that do not need to be fit or whose fit methods do nothing.

**property parameters** (*self*)

Returns the parameters which were used to initialize the component

**predict** (*self, X*)

Make predictions using selected features.

**Parameters** **X** (*pd.DataFrame, np.ndarray*) – Data of shape [n\_samples, n\_features]

**Returns** Predicted values

**Return type** pd.Series

**predict\_proba** (*self, X*)

Make probability estimates for labels.

**Parameters** **X** (*pd.DataFrame, or np.ndarray*) – Features

**Returns** Probability estimates

**Return type** pd.Series

**save** (*self, file\_path, pickle\_protocol=cloudpickle.DEFAULT\_PROTOCOL*)

Saves component at file path

**Parameters**

- **file\_path** (*str*) – Location to save file
- **pickle\_protocol** (*int*) – The pickle data stream format.

**Returns** None

**property supported\_problem\_types** (*cls*)

Problem types this estimator supports

```
class evalml.pipelines.components.estimators.ExtraTreesClassifier (n_estimators=100,  
                                                                max_features='auto',  
                                                                max_depth=6,  
                                                                min_samples_split=2,  
                                                                min_weight_fraction_leaf=0.0,  
                                                                n_jobs=-  
                                                                1,      ran-  
                                                                dom_seed=0,  
                                                                **kwargs)
```

Extra Trees Classifier.

## Parameters

- **n\_estimators** (*float*) – The number of trees in the forest. Defaults to 100.
- **max\_features** (*int, float or {"auto", "sqrt", "log2"}*) – The number of features to consider when looking for the best split:
  - If *int*, then consider `max_features` features at each split.
  - If *float*, then `max_features` is a fraction and `int(max_features * n_features)` features are considered at each split.
  - If “auto”, then `max_features=sqrt(n_features)`.
  - If “sqrt”, then `max_features=sqrt(n_features)`.
  - If “log2”, then `max_features=log2(n_features)`.
  - If *None*, then `max_features = n_features`.

The search for a split does not stop until at least one valid partition of the node samples is found, even if it requires to effectively inspect more than `max_features` features. Defaults to “auto”.

- **max\_depth** (*int*) – The maximum depth of the tree. Defaults to 6.
- **min\_samples\_split** (*int or float*) – The minimum number of samples required to split an internal node:
  - If *int*, then consider `min_samples_split` as the minimum number.
  - If *float*, then `min_samples_split` is a fraction and `ceil(min_samples_split * n_samples)` are the minimum number of samples for each split.
- **to 2.** (*Defaults*) –
- **min\_weight\_fraction\_leaf** (*float*) – The minimum weighted fraction of the sum total of weights (of all the input samples) required to be at a leaf node. Defaults to 0.0.
- **n\_jobs** (*int or None*) – Number of jobs to run in parallel. -1 uses all processes. Defaults to -1.
- **random\_seed** (*int*) – Seed for the random number generator. Defaults to 0.

## Attributes

<b>hyper-parameter_ranges</b>	{ “n_estimators”: Integer(10, 1000), “max_features”: [“auto”, “sqrt”, “log2”], “max_depth”: Integer(4, 10), }
<b>model_family</b>	ModelFamily.EXTRA_TREES
<b>modifies_features</b>	True
<b>modifies_target</b>	False
<b>name</b>	Extra Trees Classifier
<b>predict_uses_y</b>	False
<b>supported_problem_types</b>	[ ProblemTypes.BINARY, ProblemTypes.MULTICLASS, ProblemTypes.TIME_SERIES_BINARY, ProblemTypes.TIME_SERIES_MULTICLASS, ]

## Methods

<code>clone</code>	Constructs a new component with the same parameters and random state.
<code>default_parameters</code>	Returns the default parameters for this component.
<code>describe</code>	Describe a component and its parameters
<code>feature_importance</code>	Returns importance associated with each feature.
<code>fit</code>	Fits component to data
<code>load</code>	Loads component at file path
<code>needs_fitting</code>	Returns boolean determining if component needs fitting before
<code>parameters</code>	Returns the parameters which were used to initialize the component
<code>predict</code>	Make predictions using selected features.
<code>predict_proba</code>	Make probability estimates for labels.
<code>save</code>	Saves component at file path

**clone** (*self*)

Constructs a new component with the same parameters and random state.

**Returns** A new instance of this component with identical parameters and random state.

**default\_parameters** (*cls*)

Returns the default parameters for this component.

Our convention is that `Component.default_parameters == Component().parameters`.

**Returns** default parameters for this component.

**Return type** dict

**describe** (*self*, *print\_name=False*, *return\_dict=False*)

Describe a component and its parameters

**Parameters**

- **print\_name** (*bool*, *optional*) – whether to print name of component
- **return\_dict** (*bool*, *optional*) – whether to return description as dictionary in the format {"name": name, "parameters": parameters}

**Returns** prints and returns dictionary

**Return type** None or dict

**property feature\_importance** (*self*)

Returns importance associated with each feature.

**Returns** Importance associated with each feature

**Return type** np.ndarray

**fit** (*self*, *X*, *y=None*)

Fits component to data

**Parameters**

- **X** (*list*, *pd.DataFrame* or *np.ndarray*) – The input training data of shape [n\_samples, n\_features]
- **y** (*list*, *pd.Series*, *np.ndarray*, *optional*) – The target training data of length [n\_samples]

**Returns** self

**static load** (*file\_path*)

Loads component at file path

**Parameters** *file\_path* (*str*) – Location to load file

**Returns** ComponentBase object

**needs\_fitting** (*self*)

Returns boolean determining if component needs fitting before calling predict, predict\_proba, transform, or feature\_importances. This can be overridden to False for components that do not need to be fit or whose fit methods do nothing.

**property parameters** (*self*)

Returns the parameters which were used to initialize the component

**predict** (*self*, *X*)

Make predictions using selected features.

**Parameters** *X* (*pd.DataFrame*, *np.ndarray*) – Data of shape [n\_samples, n\_features]

**Returns** Predicted values

**Return type** *pd.Series*

**predict\_proba** (*self*, *X*)

Make probability estimates for labels.

**Parameters** *X* (*pd.DataFrame*, or *np.ndarray*) – Features

**Returns** Probability estimates

**Return type** *pd.Series*

**save** (*self*, *file\_path*, *pickle\_protocol=cloudpickle.DEFAULT\_PROTOCOL*)

Saves component at file path

**Parameters**

- **file\_path** (*str*) – Location to save file
- **pickle\_protocol** (*int*) – The pickle data stream format.

**Returns** None

```
class evalml.pipelines.components.estimators.ExtraTreesRegressor (n_estimators=100,
                                                                max_features='auto',
                                                                max_depth=6,
                                                                min_samples_split=2,
                                                                min_weight_fraction_leaf=0.0,
                                                                n_jobs=-1,
                                                                random_seed=0,
                                                                **kwargs)
```

Extra Trees Regressor.

**Parameters**

- **n\_estimators** (*float*) – The number of trees in the forest. Defaults to 100.
- **max\_features** (*int*, *float* or {"auto", "sqrt", "log2"}) – The number of features to consider when looking for the best split:
  - If int, then consider max\_features features at each split.
  - If float, then max\_features is a fraction and int(max\_features \* n\_features) features are considered at each split.

- If “auto”, then `max_features=sqrt(n_features)`.
- If “sqrt”, then `max_features=sqrt(n_features)`.
- If “log2”, then `max_features=log2(n_features)`.
- If None, then `max_features = n_features`.

The search for a split does not stop until at least one valid partition of the node samples is found, even if it requires to effectively inspect more than `max_features` features. Defaults to “auto”.

- **`max_depth`** (*int*) – The maximum depth of the tree. Defaults to 6.
- **`min_samples_split`** (*int or float*) – The minimum number of samples required to split an internal node:
  - If int, then consider `min_samples_split` as the minimum number.
  - If float, then `min_samples_split` is a fraction and `ceil(min_samples_split * n_samples)` are the minimum number of samples for each split.
- **`to 2.`** (*Defaults*) –
- **`min_weight_fraction_leaf`** (*float*) – The minimum weighted fraction of the sum total of weights (of all the input samples) required to be at a leaf node. Defaults to 0.0.
- **`n_jobs`** (*int or None*) – Number of jobs to run in parallel. -1 uses all processes. Defaults to -1.
- **`random_seed`** (*int*) – Seed for the random number generator. Defaults to 0.

#### Attributes

<b>hyper-parameter_ranges</b>	{ “n_estimators”: Integer(10, 1000), “max_features”: [“auto”, “sqrt”, “log2”], “max_depth”: Integer(4, 10), }
<b>model_family</b>	ModelFamily.EXTRA_TREES
<b>modifies_features</b>	True
<b>modifies_target</b>	False
<b>name</b>	Extra Trees Regressor
<b>predict_uses_y</b>	False
<b>supported_problem_types</b>	[ ProblemTypes.REGRESSION, ProblemTypes.TIME_SERIES_REGRESSION, ]

#### Methods

<code>clone</code>	Constructs a new component with the same parameters and random state.
<code>default_parameters</code>	Returns the default parameters for this component.
<code>describe</code>	Describe a component and its parameters
<code>feature_importance</code>	Returns importance associated with each feature.
<code>fit</code>	Fits component to data
<code>load</code>	Loads component at file path

continues on next page

Table 388 – continued from previous page

<i>needs_fitting</i>	Returns boolean determining if component needs fitting before
<i>parameters</i>	Returns the parameters which were used to initialize the component
<i>predict</i>	Make predictions using selected features.
<i>predict_proba</i>	Make probability estimates for labels.
<i>save</i>	Saves component at file path

**clone** (*self*)

Constructs a new component with the same parameters and random state.

**Returns** A new instance of this component with identical parameters and random state.

**default\_parameters** (*cls*)

Returns the default parameters for this component.

Our convention is that `Component.default_parameters == Component().parameters`.

**Returns** default parameters for this component.

**Return type** dict

**describe** (*self*, *print\_name=False*, *return\_dict=False*)

Describe a component and its parameters

**Parameters**

- **print\_name** (*bool*, *optional*) – whether to print name of component
- **return\_dict** (*bool*, *optional*) – whether to return description as dictionary in the format {"name": name, "parameters": parameters}

**Returns** prints and returns dictionary

**Return type** None or dict

**property feature\_importance** (*self*)

Returns importance associated with each feature.

**Returns** Importance associated with each feature

**Return type** np.ndarray

**fit** (*self*, *X*, *y=None*)

Fits component to data

**Parameters**

- **X** (*list*, *pd.DataFrame* or *np.ndarray*) – The input training data of shape [n\_samples, n\_features]
- **y** (*list*, *pd.Series*, *np.ndarray*, *optional*) – The target training data of length [n\_samples]

**Returns** self

**static load** (*file\_path*)

Loads component at file path

**Parameters** **file\_path** (*str*) – Location to load file

**Returns** ComponentBase object

**needs\_fitting** (*self*)

Returns boolean determining if component needs fitting before calling predict, predict\_proba, transform, or feature\_importances. This can be overridden to False for components that do not need to be fit or whose fit methods do nothing.

**property parameters** (*self*)

Returns the parameters which were used to initialize the component

**predict** (*self*, *X*)

Make predictions using selected features.

**Parameters** *X* (*pd.DataFrame*, *np.ndarray*) – Data of shape [n\_samples, n\_features]

**Returns** Predicted values

**Return type** *pd.Series*

**predict\_proba** (*self*, *X*)

Make probability estimates for labels.

**Parameters** *X* (*pd.DataFrame*, or *np.ndarray*) – Features

**Returns** Probability estimates

**Return type** *pd.Series*

**save** (*self*, *file\_path*, *pickle\_protocol=cloudpickle.DEFAULT\_PROTOCOL*)

Saves component at file path

**Parameters**

- **file\_path** (*str*) – Location to save file
- **pickle\_protocol** (*int*) – The pickle data stream format.

**Returns** None

```
class evalml.pipelines.components.estimators.KNeighborsClassifier (n_neighbors=5,  
                                                                weights='uniform',  
                                                                algo-  
                                                                rithm='auto',  
                                                                leaf_size=30,  
                                                                p=2, ran-  
                                                                dom_seed=0,  
                                                                **kwargs)
```

K-Nearest Neighbors Classifier.

**Parameters**

- **n\_neighbors** (*int*) – Number of neighbors to use by default. Defaults to 5.
- **weights** ({*'uniform'*, *'distance'*} or *callable*) – Weight function used in prediction. Can be:
  - *'uniform'* : uniform weights. All points in each neighborhood are weighted equally.
  - *'distance'* : weight points by the inverse of their distance. in this case, closer neighbors of a query point will have a greater influence than neighbors which are further away.
  - [*callable*] : a user-defined function which accepts an array of distances, and returns an array of the same shape containing the weights.

Defaults to “uniform”.

- **algorithm** ({*'auto'*, *'ball\_tree'*, *'kd\_tree'*, *'brute'*}) – Algorithm used to compute the nearest neighbors:



- ‘ball\_tree’ will use BallTree
- ‘kd\_tree’ will use KDTree
- ‘brute’ will use a brute-force search.

‘auto’ will attempt to decide the most appropriate algorithm based on the values passed to fit method. Defaults to “auto”. Note: fitting on sparse input will override the setting of this parameter, using brute force.

- **leaf\_size** (*int*) – Leaf size passed to BallTree or KDTree. This can affect the speed of the construction and query, as well as the memory required to store the tree. The optimal value depends on the nature of the problem. Defaults to 30.
- **p** (*int*) – Power parameter for the Minkowski metric. When  $p = 1$ , this is equivalent to using manhattan\_distance (l1), and euclidean\_distance (l2) for  $p = 2$ . For arbitrary  $p$ , minkowski\_distance (l\_p) is used. Defaults to 2.
- **random\_seed** (*int*) – Seed for the random number generator. Defaults to 0.

#### Attributes

<b>hyper-parameter_ranges</b>	{ “n_neighbors”: Integer(2, 12), “weights”: [“uniform”, “distance”], “algorithm”: [“auto”, “ball_tree”, “kd_tree”, “brute”], “leaf_size”: Integer(10, 30), “p”: Integer(1, 5),}
<b>model_family</b>	ModelFamily.K_NEIGHBORS
<b>modifies_features</b>	True
<b>modifies_target</b>	False
<b>name</b>	KNN Classifier
<b>predict Uses y</b>	False
<b>supported_problem_types</b>	[ ProblemTypes.BINARY, ProblemTypes.MULTICLASS, ProblemTypes.TIME_SERIES_BINARY, ProblemTypes.TIME_SERIES_MULTICLASS,]

#### Methods

<i>clone</i>	Constructs a new component with the same parameters and random state.
<i>default_parameters</i>	Returns the default parameters for this component.
<i>describe</i>	Describe a component and its parameters
<i>feature_importance</i>	Returns array of 0’s matching the input number of features as feature_importance is
<i>fit</i>	Fits component to data
<i>load</i>	Loads component at file path
<i>needs_fitting</i>	Returns boolean determining if component needs fitting before
<i>parameters</i>	Returns the parameters which were used to initialize the component
<i>predict</i>	Make predictions using selected features.
<i>predict_proba</i>	Make probability estimates for labels.
<i>save</i>	Saves component at file path

#### **clone** (*self*)

Constructs a new component with the same parameters and random state.

**Returns** A new instance of this component with identical parameters and random state.

**default\_parameters** (*cls*)

Returns the default parameters for this component.

Our convention is that `Component.default_parameters == Component().parameters`.

**Returns** default parameters for this component.

**Return type** dict

**describe** (*self*, *print\_name=False*, *return\_dict=False*)

Describe a component and its parameters

**Parameters**

- **print\_name** (*bool*, *optional*) – whether to print name of component
- **return\_dict** (*bool*, *optional*) – whether to return description as dictionary in the format {"name": name, "parameters": parameters}

**Returns** prints and returns dictionary

**Return type** None or dict

**property feature\_importance** (*self*)

Returns array of 0's matching the input number of features as `feature_importance` is not defined for KNN classifiers.

**fit** (*self*, *X*, *y=None*)

Fits component to data

**Parameters**

- **X** (*list*, *pd.DataFrame* or *np.ndarray*) – The input training data of shape [n\_samples, n\_features]
- **y** (*list*, *pd.Series*, *np.ndarray*, *optional*) – The target training data of length [n\_samples]

**Returns** self

**static load** (*file\_path*)

Loads component at file path

**Parameters** **file\_path** (*str*) – Location to load file

**Returns** ComponentBase object

**needs\_fitting** (*self*)

Returns boolean determining if component needs fitting before calling `predict`, `predict_proba`, `transform`, or `feature_importances`. This can be overridden to `False` for components that do not need to be fit or whose fit methods do nothing.

**property parameters** (*self*)

Returns the parameters which were used to initialize the component

**predict** (*self*, *X*)

Make predictions using selected features.

**Parameters** **X** (*pd.DataFrame*, *np.ndarray*) – Data of shape [n\_samples, n\_features]

**Returns** Predicted values

**Return type** `pd.Series`

**predict\_proba** (*self*, *X*)

Make probability estimates for labels.

**Parameters** *X* (*pd.DataFrame*, or *np.ndarray*) – Features

**Returns** Probability estimates

**Return type** *pd.Series*

**save** (*self*, *file\_path*, *pickle\_protocol=cloudpickle.DEFAULT\_PROTOCOL*)

Saves component at file path

**Parameters**

- **file\_path** (*str*) – Location to save file
- **pickle\_protocol** (*int*) – The pickle data stream format.

**Returns** None

```
class evalml.pipelines.components.estimators.LightGBMClassifier (boosting_type='gbdt',
                                                                learn-
                                                                ing_rate=0.1,
                                                                n_estimators=100,
                                                                max_depth=0,
                                                                num_leaves=31,
                                                                min_child_samples=20,
                                                                bag-
                                                                ging_fraction=0.9,
                                                                bag-
                                                                ging_freq=0,
                                                                n_jobs=-
                                                                1,
                                                                ran-
                                                                dom_seed=0,
                                                                **kwargs)
```

LightGBM Classifier.

**Parameters**

- **boosting\_type** (*string*) – Type of boosting to use. Defaults to “gbdt”. - ‘gbdt’ uses traditional Gradient Boosting Decision Tree - “dart”, uses Dropouts meet Multiple Additive Regression Trees - “goss”, uses Gradient-based One-Side Sampling - “rf”, uses Random Forest
- **learning\_rate** (*float*) – Boosting learning rate. Defaults to 0.1.
- **n\_estimators** (*int*) – Number of boosted trees to fit. Defaults to 100.
- **max\_depth** (*int*) – Maximum tree depth for base learners, <=0 means no limit. Defaults to 0.
- **num\_leaves** (*int*) – Maximum tree leaves for base learners. Defaults to 31.
- **min\_child\_samples** (*int*) – Minimum number of data needed in a child (leaf). Defaults to 20.
- **bagging\_fraction** (*float*) – LightGBM will randomly select a subset of features on each iteration (tree) without resampling if this is smaller than 1.0. For example, if set to 0.8, LightGBM will select 80% of features before training each tree. This can be used to speed up training and deal with overfitting. Defaults to 0.9.

- **bagging\_freq** (*int*) – Frequency for bagging. 0 means bagging is disabled. k means perform bagging at every k iteration. Every k-th iteration, LightGBM will randomly select `bagging_fraction * 100 %` of the data to use for the next k iterations. Defaults to 0.
- **n\_jobs** (*int or None*) – Number of threads to run in parallel. -1 uses all threads. Defaults to -1.
- **random\_seed** (*int*) – Seed for the random number generator. Defaults to 0.

#### Attributes

<b>hyper-parameter_ranges</b>	{ "learning_rate": Real(0.000001, 1), "boosting_type": ["gbdt", "dart", "goss", "rf"], "n_estimators": Integer(10, 100), "max_depth": Integer(0, 10), "num_leaves": Integer(2, 100), "min_child_samples": Integer(1, 100), "bagging_fraction": Real(0.000001, 1), "bagging_freq": Integer(0, 1), }
<b>model_family</b>	ModelFamily.LIGHTGBM
<b>modifies_features</b>	True
<b>modifies_target</b>	False
<b>name</b>	LightGBM Classifier
<b>pre-dict_uses_y</b>	False
<b>SEED_MAX</b>	SEED_BOUNDS.max_bound
<b>SEED_MIN</b>	0
<b>supported_problem_types</b>	[ ProblemTypes.BINARY, ProblemTypes.MULTICLASS, ProblemTypes.TIME_SERIES_BINARY, ProblemTypes.TIME_SERIES_MULTICLASS, ]

#### Methods

<code>clone</code>	Constructs a new component with the same parameters and random state.
<code>default_parameters</code>	Returns the default parameters for this component.
<code>describe</code>	Describe a component and its parameters
<code>feature_importance</code>	Returns importance associated with each feature.
<code>fit</code>	Fits component to data
<code>load</code>	Loads component at file path
<code>needs_fitting</code>	Returns boolean determining if component needs fitting before
<code>parameters</code>	Returns the parameters which were used to initialize the component
<code>predict</code>	Make predictions using selected features.
<code>predict_proba</code>	Make probability estimates for labels.
<code>save</code>	Saves component at file path

#### **clone** (*self*)

Constructs a new component with the same parameters and random state.

**Returns** A new instance of this component with identical parameters and random state.

#### **default\_parameters** (*cls*)

Returns the default parameters for this component.

Our convention is that `Component.default_parameters == Component().parameters`.

**Returns** default parameters for this component.

**Return type** dict

**describe** (*self*, *print\_name=False*, *return\_dict=False*)

Describe a component and its parameters

**Parameters**

- **print\_name** (*bool*, *optional*) – whether to print name of component
- **return\_dict** (*bool*, *optional*) – whether to return description as dictionary in the format {"name": name, "parameters": parameters}

**Returns** prints and returns dictionary

**Return type** None or dict

**property feature\_importance** (*self*)

Returns importance associated with each feature.

**Returns** Importance associated with each feature

**Return type** np.ndarray

**fit** (*self*, *X*, *y=None*)

Fits component to data

**Parameters**

- **X** (*list*, *pd.DataFrame* or *np.ndarray*) – The input training data of shape [n\_samples, n\_features]
- **y** (*list*, *pd.Series*, *np.ndarray*, *optional*) – The target training data of length [n\_samples]

**Returns** self

**static load** (*file\_path*)

Loads component at file path

**Parameters** **file\_path** (*str*) – Location to load file

**Returns** ComponentBase object

**needs\_fitting** (*self*)

Returns boolean determining if component needs fitting before calling predict, predict\_proba, transform, or feature\_importances. This can be overridden to False for components that do not need to be fit or whose fit methods do nothing.

**property parameters** (*self*)

Returns the parameters which were used to initialize the component

**predict** (*self*, *X*)

Make predictions using selected features.

**Parameters** **X** (*pd.DataFrame*, *np.ndarray*) – Data of shape [n\_samples, n\_features]

**Returns** Predicted values

**Return type** pd.Series

**predict\_proba** (*self*, *X*)

Make probability estimates for labels.

**Parameters** **X** (*pd.DataFrame*, or *np.ndarray*) – Features

**Returns** Probability estimates

**Return type** `pd.Series`

**save** (*self*, *file\_path*, *pickle\_protocol*=`cloudpickle.DEFAULT_PROTOCOL`)  
Saves component at file path

**Parameters**

- **file\_path** (*str*) – Location to save file
- **pickle\_protocol** (*int*) – The pickle data stream format.

**Returns** `None`

```
class evalml.pipelines.components.estimators.LightGBMRegressor (boosting_type='gbdt',  
                                                             learn-  
                                                             ing_rate=0.1,  
                                                             n_estimators=20,  
                                                             max_depth=0,  
                                                             num_leaves=31,  
                                                             min_child_samples=20,  
                                                             bag-  
                                                             ging_fraction=0.9,  
                                                             bag-  
                                                             ging_freq=0,  
                                                             n_jobs=- 1, ran-  
                                                             dom_seed=0,  
                                                             **kwargs)
```

LightGBM Regressor.

**Parameters**

- **boosting\_type** (*string*) – Type of boosting to use. Defaults to “gbdt”. - ‘gbdt’ uses traditional Gradient Boosting Decision Tree - “dart”, uses Dropouts meet Multiple Additive Regression Trees - “goss”, uses Gradient-based One-Side Sampling - “rf”, uses Random Forest
- **learning\_rate** (*float*) – Boosting learning rate. Defaults to 0.1.
- **n\_estimators** (*int*) – Number of boosted trees to fit. Defaults to 100.
- **max\_depth** (*int*) – Maximum tree depth for base learners, <=0 means no limit. Defaults to 0.
- **num\_leaves** (*int*) – Maximum tree leaves for base learners. Defaults to 31.
- **min\_child\_samples** (*int*) – Minimum number of data needed in a child (leaf). Defaults to 20.
- **bagging\_fraction** (*float*) – LightGBM will randomly select a subset of features on each iteration (tree) without resampling if this is smaller than 1.0. For example, if set to 0.8, LightGBM will select 80% of features before training each tree. This can be used to speed up training and deal with overfitting. Defaults to 0.9.
- **bagging\_freq** (*int*) – Frequency for bagging. 0 means bagging is disabled. k means perform bagging at every k iteration. Every k-th iteration, LightGBM will randomly select `bagging_fraction * 100 %` of the data to use for the next k iterations. Defaults to 0.
- **n\_jobs** (*int or None*) – Number of threads to run in parallel. -1 uses all threads. Defaults to -1.
- **random\_seed** (*int*) – Seed for the random number generator. Defaults to 0.

## Attributes

<b>hyper-paramer-ranges</b>	{ "learning_rate": Real(0.000001, 1), "boosting_type": ["gbdt", "dart", "goss", "rf"], "n_estimators": Integer(10, 100), "max_depth": Integer(0, 10), "num_leaves": Integer(2, 100), "min_child_samples": Integer(1, 100), "bagging_fraction": Real(0.000001, 1), "bagging_freq": Integer(0, 1),}
<b>model_family</b>	ModelFamily.LIGHTGBM
<b>modifies_features</b>	True
<b>modifies_target</b>	False
<b>name</b>	LightGBM Regressor
<b>pre-dict_uses_y</b>	False
<b>SEED_MAX</b>	SEED_BOUNDS.max_bound
<b>SEED_MIN</b>	0
<b>supported_problem_types</b>	[ProblemTypes.REGRESSION]

## Methods

<i>clone</i>	Constructs a new component with the same parameters and random state.
<i>default_parameters</i>	Returns the default parameters for this component.
<i>describe</i>	Describe a component and its parameters
<i>feature_importance</i>	Returns importance associated with each feature.
<i>fit</i>	Fits component to data
<i>load</i>	Loads component at file path
<i>needs_fitting</i>	Returns boolean determining if component needs fitting before
<i>parameters</i>	Returns the parameters which were used to initialize the component
<i>predict</i>	Make predictions using selected features.
<i>predict_proba</i>	Make probability estimates for labels.
<i>save</i>	Saves component at file path

**clone** (*self*)

Constructs a new component with the same parameters and random state.

**Returns** A new instance of this component with identical parameters and random state.

**default\_parameters** (*cls*)

Returns the default parameters for this component.

Our convention is that `Component.default_parameters == Component().parameters`.

**Returns** default parameters for this component.

**Return type** dict

**describe** (*self*, *print\_name=False*, *return\_dict=False*)

Describe a component and its parameters

**Parameters**

- **print\_name** (*bool*, *optional*) – whether to print name of component

- **return\_dict** (*bool, optional*) – whether to return description as dictionary in the format {"name": name, "parameters": parameters}

**Returns** prints and returns dictionary

**Return type** None or dict

**property feature\_importance** (*self*)

Returns importance associated with each feature.

**Returns** Importance associated with each feature

**Return type** np.ndarray

**fit** (*self, X, y=None*)

Fits component to data

**Parameters**

- **X** (*list, pd.DataFrame or np.ndarray*) – The input training data of shape [n\_samples, n\_features]
- **y** (*list, pd.Series, np.ndarray, optional*) – The target training data of length [n\_samples]

**Returns** self

**static load** (*file\_path*)

Loads component at file path

**Parameters** **file\_path** (*str*) – Location to load file

**Returns** ComponentBase object

**needs\_fitting** (*self*)

Returns boolean determining if component needs fitting before calling predict, predict\_proba, transform, or feature\_importances. This can be overridden to False for components that do not need to be fit or whose fit methods do nothing.

**property parameters** (*self*)

Returns the parameters which were used to initialize the component

**predict** (*self, X*)

Make predictions using selected features.

**Parameters** **X** (*pd.DataFrame, np.ndarray*) – Data of shape [n\_samples, n\_features]

**Returns** Predicted values

**Return type** pd.Series

**predict\_proba** (*self, X*)

Make probability estimates for labels.

**Parameters** **X** (*pd.DataFrame, or np.ndarray*) – Features

**Returns** Probability estimates

**Return type** pd.Series

**save** (*self, file\_path, pickle\_protocol=cloudpickle.DEFAULT\_PROTOCOL*)

Saves component at file path

**Parameters**

- **file\_path** (*str*) – Location to save file



- **pickle\_protocol** (*int*) – The pickle data stream format.

**Returns** None

```
class evalml.pipelines.components.estimators.LinearRegressor (fit_intercept=True,  
                                                         normalize=False,  
                                                         n_jobs=-1, random_seed=0,  
                                                         **kwargs)
```

Linear Regressor.

#### Parameters

- **fit\_intercept** (*boolean*) – Whether to calculate the intercept for this model. If set to False, no intercept will be used in calculations (i.e. data is expected to be centered). Defaults to True.
- **normalize** (*boolean*) – If True, the regressors will be normalized before regression by subtracting the mean and dividing by the l2-norm. This parameter is ignored when fit\_intercept is set to False. Defaults to False.
- **n\_jobs** (*int or None*) – Number of jobs to run in parallel. -1 uses all threads. Defaults to -1.
- **random\_seed** (*int*) – Seed for the random number generator. Defaults to 0.

#### Attributes

<b>hyper-parameter_ranges</b>	{ “fit_intercept”: [True, False], “normalize”: [True, False]}
<b>model_family</b>	ModelFamily.LINEAR_MODEL
<b>modifies_features</b>	True
<b>modifies_target</b>	False
<b>name</b>	Linear Regressor
<b>predict Uses y</b>	False
<b>supported_problem_types</b>	[ ProblemTypes.REGRESSION, ProblemTypes.TIME_SERIES_REGRESSION,]

#### Methods

<i>clone</i>	Constructs a new component with the same parameters and random state.
<i>default_parameters</i>	Returns the default parameters for this component.
<i>describe</i>	Describe a component and its parameters
<i>feature_importance</i>	Returns importance associated with each feature.
<i>fit</i>	Fits component to data
<i>load</i>	Loads component at file path
<i>needs_fitting</i>	Returns boolean determining if component needs fitting before
<i>parameters</i>	Returns the parameters which were used to initialize the component
<i>predict</i>	Make predictions using selected features.

continues on next page

Table 392 – continued from previous page

<code>predict_proba</code>	Make probability estimates for labels.
<code>save</code>	Saves component at file path

**clone** (*self*)

Constructs a new component with the same parameters and random state.

**Returns** A new instance of this component with identical parameters and random state.

**default\_parameters** (*cls*)

Returns the default parameters for this component.

Our convention is that `Component.default_parameters == Component().parameters`.

**Returns** default parameters for this component.

**Return type** dict

**describe** (*self*, *print\_name=False*, *return\_dict=False*)

Describe a component and its parameters

**Parameters**

- **print\_name** (*bool*, *optional*) – whether to print name of component
- **return\_dict** (*bool*, *optional*) – whether to return description as dictionary in the format {"name": name, "parameters": parameters}

**Returns** prints and returns dictionary

**Return type** None or dict

**property feature\_importance** (*self*)

Returns importance associated with each feature.

**Returns** Importance associated with each feature

**Return type** np.ndarray

**fit** (*self*, *X*, *y=None*)

Fits component to data

**Parameters**

- **X** (*list*, *pd.DataFrame* or *np.ndarray*) – The input training data of shape [n\_samples, n\_features]
- **y** (*list*, *pd.Series*, *np.ndarray*, *optional*) – The target training data of length [n\_samples]

**Returns** self

**static load** (*file\_path*)

Loads component at file path

**Parameters** **file\_path** (*str*) – Location to load file

**Returns** ComponentBase object

**needs\_fitting** (*self*)

Returns boolean determining if component needs fitting before calling predict, predict\_proba, transform, or feature\_importances. This can be overridden to False for components that do not need to be fit or whose fit methods do nothing.

**property parameters** (*self*)

Returns the parameters which were used to initialize the component

**predict** (*self*, *X*)

Make predictions using selected features.

**Parameters** *X* (*pd.DataFrame*, *np.ndarray*) – Data of shape [n\_samples, n\_features]

**Returns** Predicted values

**Return type** *pd.Series*

**predict\_proba** (*self*, *X*)

Make probability estimates for labels.

**Parameters** *X* (*pd.DataFrame*, or *np.ndarray*) – Features

**Returns** Probability estimates

**Return type** *pd.Series*

**save** (*self*, *file\_path*, *pickle\_protocol=cloudpickle.DEFAULT\_PROTOCOL*)

Saves component at file path

**Parameters**

- **file\_path** (*str*) – Location to save file
- **pickle\_protocol** (*int*) – The pickle data stream format.

**Returns** None

```
class evalml.pipelines.components.estimators.LogisticRegressionClassifier (penalty='l2',
                                                                    C=1.0,
                                                                    multi_class='auto',
                                                                    solver='lbfgs',
                                                                    n_jobs=-
                                                                    1,
                                                                    random_seed=0,
                                                                    **kwargs)
```

Logistic Regression Classifier.

**Parameters**

- **penalty** (*{ "l1", "l2", "elasticnet", "none" }*) – The norm used in penalization. Defaults to “l2”.
- **C** (*float*) – Inverse of regularization strength. Must be a positive float. Defaults to 1.0.
- **multi\_class** (*{ "auto", "ovr", "multinomial" }*) – If the option chosen is “ovr”, then a binary problem is fit for each label. For “multinomial” the loss minimised is the multinomial loss fit across the entire probability distribution, even when the data is binary. “multinomial” is unavailable when solver=“liblinear”. “auto” selects “ovr” if the data is binary, or if solver=“liblinear”, and otherwise selects “multinomial”. Defaults to “auto”.
- **solver** (*{ "newton-cg", "lbfgs", "liblinear", "sag", "saga" }*) – Algorithm to use in the optimization problem. For small datasets, “liblinear” is a good choice, whereas “sag” and “saga” are faster for large ones. For multiclass problems, only “newton-cg”, “sag”, “saga” and “lbfgs” handle multinomial loss; “liblinear” is limited to one-versus-rest schemes.
  - “newton-cg”, “lbfgs”, “sag” and “saga” handle L2 or no penalty
  - “liblinear” and “saga” also handle L1 penalty
  - “saga” also supports “elasticnet” penalty

- “liblinear” does not support setting `penalty='none'`

Defaults to “lbfgs”.

- **n\_jobs** (*int*) – Number of parallel threads used to run xgboost. Note that creating thread contention will significantly slow down the algorithm. Defaults to -1.
- **random\_seed** (*int*) – Seed for the random number generator. Defaults to 0.

#### Attributes

<b>hyper-parameter_ranges</b>	{ “penalty”: [“l2”], “C”: Real(0.01, 10), }
<b>model_family</b>	ModelFamily.LINEAR_MODEL
<b>modifies_features</b>	True
<b>modifies_target</b>	False
<b>name</b>	Logistic Regression Classifier
<b>predict_uses_y</b>	False
<b>supported_problem_types</b>	[ ProblemTypes.BINARY, ProblemTypes.MULTICLASS, ProblemTypes.TIME_SERIES_BINARY, ProblemTypes.TIME_SERIES_MULTICLASS, ]

#### Methods

<i>clone</i>	Constructs a new component with the same parameters and random state.
<i>default_parameters</i>	Returns the default parameters for this component.
<i>describe</i>	Describe a component and its parameters
<i>feature_importance</i>	Returns importance associated with each feature.
<i>fit</i>	Fits component to data
<i>load</i>	Loads component at file path
<i>needs_fitting</i>	Returns boolean determining if component needs fitting before
<i>parameters</i>	Returns the parameters which were used to initialize the component
<i>predict</i>	Make predictions using selected features.
<i>predict_proba</i>	Make probability estimates for labels.
<i>save</i>	Saves component at file path

#### **clone** (*self*)

Constructs a new component with the same parameters and random state.

**Returns** A new instance of this component with identical parameters and random state.

#### **default\_parameters** (*cls*)

Returns the default parameters for this component.

Our convention is that `Component.default_parameters == Component().parameters`.

**Returns** default parameters for this component.

**Return type** dict

#### **describe** (*self*, *print\_name=False*, *return\_dict=False*)

Describe a component and its parameters

#### Parameters

- **print\_name** (*bool, optional*) – whether to print name of component
- **return\_dict** (*bool, optional*) – whether to return description as dictionary in the format {"name": name, "parameters": parameters}

**Returns** prints and returns dictionary

**Return type** None or dict

**property feature\_importance** (*self*)

Returns importance associated with each feature.

**Returns** Importance associated with each feature

**Return type** np.ndarray

**fit** (*self, X, y=None*)

Fits component to data

#### Parameters

- **X** (*list, pd.DataFrame or np.ndarray*) – The input training data of shape [n\_samples, n\_features]
- **y** (*list, pd.Series, np.ndarray, optional*) – The target training data of length [n\_samples]

**Returns** self

**static load** (*file\_path*)

Loads component at file path

**Parameters** **file\_path** (*str*) – Location to load file

**Returns** ComponentBase object

**needs\_fitting** (*self*)

Returns boolean determining if component needs fitting before calling predict, predict\_proba, transform, or feature\_importances. This can be overridden to False for components that do not need to be fit or whose fit methods do nothing.

**property parameters** (*self*)

Returns the parameters which were used to initialize the component

**predict** (*self, X*)

Make predictions using selected features.

**Parameters** **X** (*pd.DataFrame, np.ndarray*) – Data of shape [n\_samples, n\_features]

**Returns** Predicted values

**Return type** pd.Series

**predict\_proba** (*self, X*)

Make probability estimates for labels.

**Parameters** **X** (*pd.DataFrame, or np.ndarray*) – Features

**Returns** Probability estimates

**Return type** pd.Series

**save** (*self*, *file\_path*, *pickle\_protocol*=*cloudpickle.DEFAULT\_PROTOCOL*)  
Saves component at file path

#### Parameters

- **file\_path** (*str*) – Location to save file
- **pickle\_protocol** (*int*) – The pickle data stream format.

**Returns** None

```
class evalml.pipelines.components.estimators.ProphetRegressor (date_column='ds',
                                                             change-
                                                             point_prior_scale=0.05,
                                                             seasonal-
                                                             ity_prior_scale=10,
                                                             holi-
                                                             days_prior_scale=10,
                                                             seasonal-
                                                             ity_mode='additive',
                                                             random_seed=0,
                                                             stan_backend='CMDSTANPY',
                                                             **kwargs)
```

Prophet is a procedure for forecasting time series data based on an additive model where non-linear trends are fit with yearly, weekly, and daily seasonality, plus holiday effects. It works best with time series that have strong seasonal effects and several seasons of historical data. Prophet is robust to missing data and shifts in the trend, and typically handles outliers well.

More information here: <https://facebook.github.io/prophet/>

#### Attributes

<b>hyper-parameter_ranges</b>	{ "changepoint_prior_scale": Real(0.001, 0.5), "seasonality_prior_scale": Real(0.01, 10), "holidays_prior_scale": Real(0.01, 10), "seasonality_mode": ["additive", "multiplicative"], }
<b>model_family</b>	ModelFamily.PROPHET
<b>modifies_features</b>	True
<b>modifies_target</b>	False
<b>name</b>	Prophet Regressor
<b>pre-dict_uses_y</b>	False
<b>supported_problem_types</b>	[ProblemTypes.TIME_SERIES_REGRESSION]

#### Methods

<i>build_prophet_df</i>	
<i>clone</i>	Constructs a new component with the same parameters and random state.
<i>default_parameters</i>	Returns the default parameters for this component.
<i>describe</i>	Describe a component and its parameters
<i>feature_importance</i>	Returns array of 0's with len(1) as feature_importance is not defined for Prophet regressor.

continues on next page

Table 394 – continued from previous page

<i>fit</i>	Fits component to data
<i>get_params</i>	
<i>load</i>	Loads component at file path
<i>needs_fitting</i>	Returns boolean determining if component needs fitting before
<i>parameters</i>	Returns the parameters which were used to initialize the component
<i>predict</i>	Make predictions using selected features.
<i>predict_proba</i>	Make probability estimates for labels.
<i>save</i>	Saves component at file path

**static build\_prophet\_df** (*X*, *y=None*, *date\_column='ds'*)

**clone** (*self*)

Constructs a new component with the same parameters and random state.

**Returns** A new instance of this component with identical parameters and random state.

**default\_parameters** (*cls*)

Returns the default parameters for this component.

Our convention is that `Component.default_parameters == Component().parameters`.

**Returns** default parameters for this component.

**Return type** dict

**describe** (*self*, *print\_name=False*, *return\_dict=False*)

Describe a component and its parameters

**Parameters**

- **print\_name** (*bool*, *optional*) – whether to print name of component
- **return\_dict** (*bool*, *optional*) – whether to return description as dictionary in the format {"name": name, "parameters": parameters}

**Returns** prints and returns dictionary

**Return type** None or dict

**property feature\_importance** (*self*)

Returns array of 0's with len(1) as feature\_importance is not defined for Prophet regressor.

**fit** (*self*, *X*, *y=None*)

Fits component to data

**Parameters**

- **X** (*list*, *pd.DataFrame* or *np.ndarray*) – The input training data of shape [n\_samples, n\_features]
- **y** (*list*, *pd.Series*, *np.ndarray*, *optional*) – The target training data of length [n\_samples]

**Returns** self

**get\_params** (*self*)

**static load** (*file\_path*)

Loads component at file path

**Parameters** `file_path` (*str*) – Location to load file

**Returns** ComponentBase object

**needs\_fitting** (*self*)

Returns boolean determining if component needs fitting before calling predict, predict\_proba, transform, or feature\_importances. This can be overridden to False for components that do not need to be fit or whose fit methods do nothing.

**property parameters** (*self*)

Returns the parameters which were used to initialize the component

**predict** (*self*, *X*, *y=None*)

Make predictions using selected features.

**Parameters** *X* (*pd.DataFrame*, *np.ndarray*) – Data of shape [n\_samples, n\_features]

**Returns** Predicted values

**Return type** *pd.Series*

**predict\_proba** (*self*, *X*)

Make probability estimates for labels.

**Parameters** *X* (*pd.DataFrame*, or *np.ndarray*) – Features

**Returns** Probability estimates

**Return type** *pd.Series*

**save** (*self*, *file\_path*, *pickle\_protocol=cloudpickle.DEFAULT\_PROTOCOL*)

Saves component at file path

**Parameters**

- **file\_path** (*str*) – Location to save file
- **pickle\_protocol** (*int*) – The pickle data stream format.

**Returns** None

```
class evalml.pipelines.components.estimators.RandomForestClassifier (n_estimators=100,  
                                                                max_depth=6,  
                                                                n_jobs=-1, ran-  
                                                                dom_seed=0,  
                                                                **kwargs)
```

Random Forest Classifier.

**Parameters**

- **n\_estimators** (*float*) – The number of trees in the forest. Defaults to 100.
- **max\_depth** (*int*) – Maximum tree depth for base learners. Defaults to 6.
- **n\_jobs** (*int* or *None*) – Number of jobs to run in parallel. -1 uses all processes. Defaults to -1.
- **random\_seed** (*int*) – Seed for the random number generator. Defaults to 0.

**Attributes**



<b>hyper-parameter_ranges</b>	{ "n_estimators": Integer(10, 1000), "max_depth": Integer(1, 10), }
<b>model_family</b>	ModelFamily.RANDOM_FOREST
<b>modifies_features</b>	True
<b>modifies_target</b>	False
<b>name</b>	Random Forest Classifier
<b>pre-dict_uses_y</b>	False
<b>supported_problem_types</b>	[ ProblemTypes.BINARY, ProblemTypes.MULTICLASS, ProblemTypes.TIME_SERIES_BINARY, ProblemTypes.TIME_SERIES_MULTICLASS, ]

## Methods

<i>clone</i>	Constructs a new component with the same parameters and random state.
<i>default_parameters</i>	Returns the default parameters for this component.
<i>describe</i>	Describe a component and its parameters
<i>feature_importance</i>	Returns importance associated with each feature.
<i>fit</i>	Fits component to data
<i>load</i>	Loads component at file path
<i>needs_fitting</i>	Returns boolean determining if component needs fitting before
<i>parameters</i>	Returns the parameters which were used to initialize the component
<i>predict</i>	Make predictions using selected features.
<i>predict_proba</i>	Make probability estimates for labels.
<i>save</i>	Saves component at file path

### **clone** (*self*)

Constructs a new component with the same parameters and random state.

**Returns** A new instance of this component with identical parameters and random state.

### **default\_parameters** (*cls*)

Returns the default parameters for this component.

Our convention is that `Component.default_parameters == Component().parameters`.

**Returns** default parameters for this component.

**Return type** dict

### **describe** (*self*, *print\_name=False*, *return\_dict=False*)

Describe a component and its parameters

#### Parameters

- **print\_name** (*bool*, *optional*) – whether to print name of component
- **return\_dict** (*bool*, *optional*) – whether to return description as dictionary in the format {"name": name, "parameters": parameters}

**Returns** prints and returns dictionary

**Return type** None or dict

**property feature\_importance** (*self*)

Returns importance associated with each feature.

**Returns** Importance associated with each feature

**Return type** np.ndarray

**fit** (*self*, *X*, *y=None*)

Fits component to data

**Parameters**

- **X** (*list*, *pd.DataFrame* or *np.ndarray*) – The input training data of shape [n\_samples, n\_features]
- **y** (*list*, *pd.Series*, *np.ndarray*, *optional*) – The target training data of length [n\_samples]

**Returns** self

**static load** (*file\_path*)

Loads component at file path

**Parameters** **file\_path** (*str*) – Location to load file

**Returns** ComponentBase object

**needs\_fitting** (*self*)

Returns boolean determining if component needs fitting before calling predict, predict\_proba, transform, or feature\_importances. This can be overridden to False for components that do not need to be fit or whose fit methods do nothing.

**property parameters** (*self*)

Returns the parameters which were used to initialize the component

**predict** (*self*, *X*)

Make predictions using selected features.

**Parameters** **X** (*pd.DataFrame*, *np.ndarray*) – Data of shape [n\_samples, n\_features]

**Returns** Predicted values

**Return type** pd.Series

**predict\_proba** (*self*, *X*)

Make probability estimates for labels.

**Parameters** **X** (*pd.DataFrame*, or *np.ndarray*) – Features

**Returns** Probability estimates

**Return type** pd.Series

**save** (*self*, *file\_path*, *pickle\_protocol=cloudpickle.DEFAULT\_PROTOCOL*)

Saves component at file path

**Parameters**

- **file\_path** (*str*) – Location to save file
- **pickle\_protocol** (*int*) – The pickle data stream format.

**Returns** None

```
class evalml.pipelines.components.estimators.RandomForestRegressor (n_estimators=100,
                                                                    max_depth=6,
                                                                    n_jobs=-1,
                                                                    random_seed=0,
                                                                    **kwargs)
```

Random Forest Regressor.

#### Parameters

- **n\_estimators** (*float*) – The number of trees in the forest. Defaults to 100.
- **max\_depth** (*int*) – Maximum tree depth for base learners. Defaults to 6.
- **n\_jobs** (*int or None*) – Number of jobs to run in parallel. -1 uses all processes. Defaults to -1.
- **random\_seed** (*int*) – Seed for the random number generator. Defaults to 0.

#### Attributes

<b>hyper-parameter_ranges</b>	{ "n_estimators": Integer(10, 1000), "max_depth": Integer(1, 32), }
<b>model_family</b>	ModelFamily.RANDOM_FOREST
<b>modifies_features</b>	True
<b>modifies_target</b>	False
<b>name</b>	Random Forest Regressor
<b>predict_uses_y</b>	False
<b>supported_problem_types</b>	[ ProblemTypes.REGRESSION, ProblemTypes.TIME_SERIES_REGRESSION, ]

#### Methods

<i>clone</i>	Constructs a new component with the same parameters and random state.
<i>default_parameters</i>	Returns the default parameters for this component.
<i>describe</i>	Describe a component and its parameters
<i>feature_importance</i>	Returns importance associated with each feature.
<i>fit</i>	Fits component to data
<i>load</i>	Loads component at file path
<i>needs_fitting</i>	Returns boolean determining if component needs fitting before
<i>parameters</i>	Returns the parameters which were used to initialize the component
<i>predict</i>	Make predictions using selected features.
<i>predict_proba</i>	Make probability estimates for labels.
<i>save</i>	Saves component at file path

#### **clone** (*self*)

Constructs a new component with the same parameters and random state.

**Returns** A new instance of this component with identical parameters and random state.

**default\_parameters** (*cls*)

Returns the default parameters for this component.

Our convention is that `Component.default_parameters == Component().parameters`.

**Returns** default parameters for this component.

**Return type** dict

**describe** (*self*, *print\_name=False*, *return\_dict=False*)

Describe a component and its parameters

**Parameters**

- **print\_name** (*bool*, *optional*) – whether to print name of component
- **return\_dict** (*bool*, *optional*) – whether to return description as dictionary in the format {"name": name, "parameters": parameters}

**Returns** prints and returns dictionary

**Return type** None or dict

**property feature\_importance** (*self*)

Returns importance associated with each feature.

**Returns** Importance associated with each feature

**Return type** np.ndarray

**fit** (*self*, *X*, *y=None*)

Fits component to data

**Parameters**

- **X** (*list*, *pd.DataFrame* or *np.ndarray*) – The input training data of shape [n\_samples, n\_features]
- **y** (*list*, *pd.Series*, *np.ndarray*, *optional*) – The target training data of length [n\_samples]

**Returns** self

**static load** (*file\_path*)

Loads component at file path

**Parameters** **file\_path** (*str*) – Location to load file

**Returns** ComponentBase object

**needs\_fitting** (*self*)

Returns boolean determining if component needs fitting before calling predict, predict\_proba, transform, or feature\_importances. This can be overridden to False for components that do not need to be fit or whose fit methods do nothing.

**property parameters** (*self*)

Returns the parameters which were used to initialize the component

**predict** (*self*, *X*)

Make predictions using selected features.

**Parameters** **X** (*pd.DataFrame*, *np.ndarray*) – Data of shape [n\_samples, n\_features]

**Returns** Predicted values

**Return type** pd.Series

**predict\_proba** (*self*, *X*)

Make probability estimates for labels.

**Parameters** **X** (*pd.DataFrame*, or *np.ndarray*) – Features

**Returns** Probability estimates

**Return type** *pd.Series*

**save** (*self*, *file\_path*, *pickle\_protocol=cloudpickle.DEFAULT\_PROTOCOL*)

Saves component at file path

**Parameters**

- **file\_path** (*str*) – Location to save file
- **pickle\_protocol** (*int*) – The pickle data stream format.

**Returns** None

```
class evalml.pipelines.components.estimators.SVMClassifier (C=1.0, kernel='rbf',
                                                         gamma='scale',
                                                         probability=True,
                                                         random_seed=0,
                                                         **kwargs)
```

Support Vector Machine Classifier.

**Parameters**

- **C** (*float*) – The regularization parameter. The strength of the regularization is inversely proportional to C. Must be strictly positive. The penalty is a squared l2 penalty. Defaults to 1.0.
- **kernel** ({*"linear"*, *"poly"*, *"rbf"*, *"sigmoid"*, *"precomputed"*}) – Specifies the kernel type to be used in the algorithm. Defaults to “rbf”.
- **gamma** ({*"scale"*, *"auto"*} or *float*) – Kernel coefficient for “rbf”, “poly” and “sigmoid”. Defaults to “scale”. - If gamma=’scale’ (default) is passed then it uses 1 / (n\_features \* X.var()) as value of gamma - if “auto”, uses 1 / n\_features
- **probability** (*boolean*) – Whether to enable probability estimates. Defaults to True.
- **random\_seed** (*int*) – Seed for the random number generator. Defaults to 0.

**Attributes**

<b>hyper-parameter_ranges</b>	{ “C”: Real(0, 10), “kernel”: [“linear”, “poly”, “rbf”, “sigmoid”, “precomputed”], “gamma”: [“scale”, “auto”], }
<b>model_family</b>	ModelFamily.SVM
<b>modifies_features</b>	True
<b>modifies_target</b>	False
<b>name</b>	SVM Classifier
<b>predict_uses_y</b>	False
<b>supported_problem_types</b>	[ ProblemTypes.BINARY, ProblemTypes.MULTICLASS, ProblemTypes.TIME_SERIES_BINARY, ProblemTypes.TIME_SERIES_MULTICLASS, ]

**Methods**

<code>clone</code>	Constructs a new component with the same parameters and random state.
<code>default_parameters</code>	Returns the default parameters for this component.
<code>describe</code>	Describe a component and its parameters
<code>feature_importance</code>	Feature importance only works with linear kernels.
<code>fit</code>	Fits component to data
<code>load</code>	Loads component at file path
<code>needs_fitting</code>	Returns boolean determining if component needs fitting before
<code>parameters</code>	Returns the parameters which were used to initialize the component
<code>predict</code>	Make predictions using selected features.
<code>predict_proba</code>	Make probability estimates for labels.
<code>save</code>	Saves component at file path

**clone** (*self*)

Constructs a new component with the same parameters and random state.

**Returns** A new instance of this component with identical parameters and random state.

**default\_parameters** (*cls*)

Returns the default parameters for this component.

Our convention is that `Component.default_parameters == Component().parameters`.

**Returns** default parameters for this component.

**Return type** dict

**describe** (*self*, *print\_name=False*, *return\_dict=False*)

Describe a component and its parameters

**Parameters**

- **print\_name** (*bool*, *optional*) – whether to print name of component
- **return\_dict** (*bool*, *optional*) – whether to return description as dictionary in the format {"name": name, "parameters": parameters}

**Returns** prints and returns dictionary

**Return type** None or dict

**property feature\_importance** (*self*)

Feature importance only works with linear kernels. If the kernel isn't linear, we return a numpy array of zeros

**fit** (*self*, *X*, *y=None*)

Fits component to data

**Parameters**

- **X** (*list*, *pd.DataFrame* or *np.ndarray*) – The input training data of shape [n\_samples, n\_features]
- **y** (*list*, *pd.Series*, *np.ndarray*, *optional*) – The target training data of length [n\_samples]

**Returns** self

**static load** (*file\_path*)

Loads component at file path

**Parameters** *file\_path* (*str*) – Location to load file

**Returns** ComponentBase object

**needs\_fitting** (*self*)

Returns boolean determining if component needs fitting before calling predict, predict\_proba, transform, or feature\_importances. This can be overridden to False for components that do not need to be fit or whose fit methods do nothing.

**property parameters** (*self*)

Returns the parameters which were used to initialize the component

**predict** (*self*, *X*)

Make predictions using selected features.

**Parameters** *X* (*pd.DataFrame*, *np.ndarray*) – Data of shape [n\_samples, n\_features]

**Returns** Predicted values

**Return type** *pd.Series*

**predict\_proba** (*self*, *X*)

Make probability estimates for labels.

**Parameters** *X* (*pd.DataFrame*, or *np.ndarray*) – Features

**Returns** Probability estimates

**Return type** *pd.Series*

**save** (*self*, *file\_path*, *pickle\_protocol=cloudpickle.DEFAULT\_PROTOCOL*)

Saves component at file path

**Parameters**

- **file\_path** (*str*) – Location to save file
- **pickle\_protocol** (*int*) – The pickle data stream format.

**Returns** None

```
class evalml.pipelines.components.estimators.SVMRegressor (C=1.0, kernel='rbf',
                                                         gamma='scale',
                                                         random_seed=0,
                                                         **kwargs)
```

Support Vector Machine Regressor.

**Parameters**

- **C** (*float*) – The regularization parameter. The strength of the regularization is inversely proportional to C. Must be strictly positive. The penalty is a squared l2 penalty. Defaults to 1.0.
- **kernel** (*{ "linear", "poly", "rbf", "sigmoid", "precomputed" }*) – Specifies the kernel type to be used in the algorithm. Defaults to “rbf”.
- **gamma** (*{ "scale", "auto" } or float*) – Kernel coefficient for “rbf”, “poly” and “sigmoid”. Defaults to “scale”. - If gamma=’scale’ (default) is passed then it uses  $1 / (n\_features * X.var())$  as value of gamma - if “auto”, uses  $1 / n\_features$
- **random\_seed** (*int*) – Seed for the random number generator. Defaults to 0.

### Attributes

<b>hyper-parameter_ranges</b>	{ "C": Real(0, 10), "kernel": ["linear", "poly", "rbf", "sigmoid", "precomputed"], "gamma": ["scale", "auto"], }
<b>model_family</b>	ModelFamily.SVM
<b>modifies_features</b>	True
<b>modifies_target</b>	False
<b>name</b>	SVM Regressor
<b>pre_dict_uses_y</b>	False
<b>supported_problem_types</b>	[ ProblemTypes.REGRESSION, ProblemTypes.TIME_SERIES_REGRESSION, ]

### Methods

<i>clone</i>	Constructs a new component with the same parameters and random state.
<i>default_parameters</i>	Returns the default parameters for this component.
<i>describe</i>	Describe a component and its parameters
<i>feature_importance</i>	Feature importance only works with linear kernels.
<i>fit</i>	Fits component to data
<i>load</i>	Loads component at file path
<i>needs_fitting</i>	Returns boolean determining if component needs fitting before
<i>parameters</i>	Returns the parameters which were used to initialize the component
<i>predict</i>	Make predictions using selected features.
<i>predict_proba</i>	Make probability estimates for labels.
<i>save</i>	Saves component at file path

#### **clone** (*self*)

Constructs a new component with the same parameters and random state.

**Returns** A new instance of this component with identical parameters and random state.

#### **default\_parameters** (*cls*)

Returns the default parameters for this component.

Our convention is that `Component.default_parameters == Component().parameters`.

**Returns** default parameters for this component.

**Return type** dict

#### **describe** (*self*, *print\_name=False*, *return\_dict=False*)

Describe a component and its parameters

#### Parameters

- **print\_name** (*bool*, *optional*) – whether to print name of component
- **return\_dict** (*bool*, *optional*) – whether to return description as dictionary in the format {"name": name, "parameters": parameters}



**Returns** prints and returns dictionary

**Return type** None or dict

**property feature\_importance** (*self*)

Feature importance only works with linear kernels. If the kernel isn't linear, we return a numpy array of zeros

**fit** (*self*, *X*, *y=None*)

Fits component to data

**Parameters**

- **X** (*list*, *pd.DataFrame* or *np.ndarray*) – The input training data of shape [n\_samples, n\_features]
- **y** (*list*, *pd.Series*, *np.ndarray*, *optional*) – The target training data of length [n\_samples]

**Returns** self

**static load** (*file\_path*)

Loads component at file path

**Parameters** **file\_path** (*str*) – Location to load file

**Returns** ComponentBase object

**needs\_fitting** (*self*)

Returns boolean determining if component needs fitting before calling predict, predict\_proba, transform, or feature\_importances. This can be overridden to False for components that do not need to be fit or whose fit methods do nothing.

**property parameters** (*self*)

Returns the parameters which were used to initialize the component

**predict** (*self*, *X*)

Make predictions using selected features.

**Parameters** **X** (*pd.DataFrame*, *np.ndarray*) – Data of shape [n\_samples, n\_features]

**Returns** Predicted values

**Return type** *pd.Series*

**predict\_proba** (*self*, *X*)

Make probability estimates for labels.

**Parameters** **X** (*pd.DataFrame*, or *np.ndarray*) – Features

**Returns** Probability estimates

**Return type** *pd.Series*

**save** (*self*, *file\_path*, *pickle\_protocol=cloudpickle.DEFAULT\_PROTOCOL*)

Saves component at file path

**Parameters**

- **file\_path** (*str*) – Location to save file
- **pickle\_protocol** (*int*) – The pickle data stream format.

**Returns** None

```
class evalml.pipelines.components.estimators.TimeSeriesBaselineEstimator (gap=1,  
                                                                    ran-  
                                                                    dom_seed=0,  
                                                                    **kwargs)
```

Time series estimator that predicts using the naive forecasting approach.

This is useful as a simple baseline estimator for time series problems.

#### Parameters

- **gap** (*int*) – Gap between prediction date and target date and must be a positive integer. If gap is 0, target date will be shifted ahead by 1 time period. Defaults to 1.
- **random\_seed** (*int*) – Seed for the random number generator. Defaults to 0.

#### Attributes

<b>hyper-parameter_ranges</b>	{}
<b>model_family</b>	ModelFamily.BASELINE
<b>modifies_features</b>	True
<b>modifies_target</b>	False
<b>name</b>	Time Series Baseline Estimator
<b>predict_uses_y</b>	True
<b>supported_problem_types</b>	[ProblemTypes.TIME_SERIES_REGRESSION, ProblemTypes.TIME_SERIES_BINARY, ProblemTypes.TIME_SERIES_MULTICLASS,]

#### Methods

<i>clone</i>	Constructs a new component with the same parameters and random state.
<i>default_parameters</i>	Returns the default parameters for this component.
<i>describe</i>	Describe a component and its parameters
<i>feature_importance</i>	Returns importance associated with each feature.
<i>fit</i>	Fits component to data
<i>load</i>	Loads component at file path
<i>needs_fitting</i>	Returns boolean determining if component needs fitting before
<i>parameters</i>	Returns the parameters which were used to initialize the component
<i>predict</i>	Make predictions using selected features.
<i>predict_proba</i>	Make probability estimates for labels.
<i>save</i>	Saves component at file path

**clone** (*self*)

Constructs a new component with the same parameters and random state.

**Returns** A new instance of this component with identical parameters and random state.

**default\_parameters** (*cls*)

Returns the default parameters for this component.

Our convention is that `Component.default_parameters == Component().parameters`.

**Returns** default parameters for this component.

**Return type** dict

**describe** (*self*, *print\_name=False*, *return\_dict=False*)

Describe a component and its parameters

**Parameters**

- **print\_name** (*bool*, *optional*) – whether to print name of component
- **return\_dict** (*bool*, *optional*) – whether to return description as dictionary in the format {"name": name, "parameters": parameters}

**Returns** prints and returns dictionary

**Return type** None or dict

**property feature\_importance** (*self*)

Returns importance associated with each feature.

Since baseline estimators do not use input features to calculate predictions, returns an array of zeroes.

**Returns** an array of zeroes

**Return type** np.ndarray (float)

**fit** (*self*, *X*, *y=None*)

Fits component to data

**Parameters**

- **X** (*list*, *pd.DataFrame* or *np.ndarray*) – The input training data of shape [n\_samples, n\_features]
- **y** (*list*, *pd.Series*, *np.ndarray*, *optional*) – The target training data of length [n\_samples]

**Returns** self

**static load** (*file\_path*)

Loads component at file path

**Parameters** **file\_path** (*str*) – Location to load file

**Returns** ComponentBase object

**needs\_fitting** (*self*)

Returns boolean determining if component needs fitting before calling predict, predict\_proba, transform, or feature\_importances. This can be overridden to False for components that do not need to be fit or whose fit methods do nothing.

**property parameters** (*self*)

Returns the parameters which were used to initialize the component

**predict** (*self*, *X*, *y=None*)

Make predictions using selected features.

**Parameters** **X** (*pd.DataFrame*, *np.ndarray*) – Data of shape [n\_samples, n\_features]

**Returns** Predicted values

**Return type** pd.Series

**predict\_proba** (*self*, *X*, *y=None*)

Make probability estimates for labels.

**Parameters** **X** (*pd.DataFrame*, or *np.ndarray*) – Features

**Returns** Probability estimates

**Return type** *pd.Series*

**save** (*self*, *file\_path*, *pickle\_protocol=cloudpickle.DEFAULT\_PROTOCOL*)

Saves component at file path

**Parameters**

- **file\_path** (*str*) – Location to save file
- **pickle\_protocol** (*int*) – The pickle data stream format.

**Returns** None

```
class evalml.pipelines.components.estimators.XGBoostClassifier(eta=0.1,  
                                                             max_depth=6,  
                                                             min_child_weight=1,  
                                                             n_estimators=100,  
                                                             ran-  
                                                             dom_seed=0,  
                                                             n_jobs=-1,  
                                                             **kwargs)
```

XGBoost Classifier.

**Parameters**

- **eta** (*float*) – Boosting learning rate. Defaults to 0.1.
- **max\_depth** (*int*) – Maximum tree depth for base learners. Defaults to 6.
- **min\_child\_weight** (*float*) – Minimum sum of instance weight (hessian) needed in a child. Defaults to 1.0
- **n\_estimators** (*int*) – Number of gradient boosted trees. Equivalent to number of boosting rounds. Defaults to 100.
- **random\_seed** (*int*) – Seed for the random number generator. Defaults to 0.
- **n\_jobs** (*int*) – Number of parallel threads used to run xgboost. Note that creating thread contention will significantly slow down the algorithm. Defaults to -1.

**Attributes**

<b>hyper-parameter_ranges</b>	{ “eta”: Real(0.000001, 1), “max_depth”: Integer(1, 10), “min_child_weight”: Real(1, 10), “n_estimators”: Integer(1, 1000), }
<b>model_family</b>	ModelFamily.XGBOOST
<b>modifies_features</b>	True
<b>modifies_target</b>	False
<b>name</b>	XGBoost Classifier
<b>pre-dict_uses_y</b>	False
<b>SEED_MAX</b>	None
<b>SEED_MIN</b>	None
<b>supported_problem_types</b>	[ ProblemTypes.BINARY, ProblemTypes.MULTICLASS, ProblemTypes.TIME_SERIES_BINARY, ProblemTypes.TIME_SERIES_MULTICLASS, ]

## Methods

<i>clone</i>	Constructs a new component with the same parameters and random state.
<i>default_parameters</i>	Returns the default parameters for this component.
<i>describe</i>	Describe a component and its parameters
<i>feature_importance</i>	Returns importance associated with each feature.
<i>fit</i>	Fits component to data
<i>load</i>	Loads component at file path
<i>needs_fitting</i>	Returns boolean determining if component needs fitting before
<i>parameters</i>	Returns the parameters which were used to initialize the component
<i>predict</i>	Make predictions using selected features.
<i>predict_proba</i>	Make probability estimates for labels.
<i>save</i>	Saves component at file path

### **clone** (*self*)

Constructs a new component with the same parameters and random state.

**Returns** A new instance of this component with identical parameters and random state.

### **default\_parameters** (*cls*)

Returns the default parameters for this component.

Our convention is that `Component.default_parameters == Component().parameters`.

**Returns** default parameters for this component.

**Return type** dict

### **describe** (*self*, *print\_name=False*, *return\_dict=False*)

Describe a component and its parameters

#### Parameters

- **print\_name** (*bool*, *optional*) – whether to print name of component
- **return\_dict** (*bool*, *optional*) – whether to return description as dictionary in the format {“name”: name, “parameters”: parameters}

**Returns** prints and returns dictionary

**Return type** None or dict

**property** `feature_importance` (*self*)

Returns importance associated with each feature.

**Returns** Importance associated with each feature

**Return type** np.ndarray

**fit** (*self*, *X*, *y=None*)

Fits component to data

**Parameters**

- **X** (*list*, *pd.DataFrame* or *np.ndarray*) – The input training data of shape [n\_samples, n\_features]
- **y** (*list*, *pd.Series*, *np.ndarray*, *optional*) – The target training data of length [n\_samples]

**Returns** self

**static** `load` (*file\_path*)

Loads component at file path

**Parameters** `file_path` (*str*) – Location to load file

**Returns** ComponentBase object

**needs\_fitting** (*self*)

Returns boolean determining if component needs fitting before calling predict, predict\_proba, transform, or feature\_importances. This can be overridden to False for components that do not need to be fit or whose fit methods do nothing.

**property** `parameters` (*self*)

Returns the parameters which were used to initialize the component

**predict** (*self*, *X*)

Make predictions using selected features.

**Parameters** **X** (*pd.DataFrame*, *np.ndarray*) – Data of shape [n\_samples, n\_features]

**Returns** Predicted values

**Return type** pd.Series

**predict\_proba** (*self*, *X*)

Make probability estimates for labels.

**Parameters** **X** (*pd.DataFrame*, or *np.ndarray*) – Features

**Returns** Probability estimates

**Return type** pd.Series

**save** (*self*, *file\_path*, *pickle\_protocol=cloudpickle.DEFAULT\_PROTOCOL*)

Saves component at file path

**Parameters**

- **file\_path** (*str*) – Location to save file
- **pickle\_protocol** (*int*) – The pickle data stream format.

**Returns** None

```
class evalml.pipelines.components.estimators.XGBoostRegressor(eta=0.1,
                                                             max_depth=6,
                                                             min_child_weight=1,
                                                             n_estimators=100,
                                                             random_seed=0,
                                                             n_jobs=-1,
                                                             **kwargs)
```

XGBoost Regressor.

#### Parameters

- **eta** (*float*) – Boosting learning rate. Defaults to 0.1.
- **max\_depth** (*int*) – Maximum tree depth for base learners. Defaults to 6.
- **min\_child\_weight** (*float*) – Minimum sum of instance weight (hessian) needed in a child. Defaults to 1.0
- **n\_estimators** (*int*) – Number of gradient boosted trees. Equivalent to number of boosting rounds. Defaults to 100.
- **random\_seed** (*int*) – Seed for the random number generator. Defaults to 0.
- **n\_jobs** (*int*) – Number of parallel threads used to run xgboost. Note that creating thread contention will significantly slow down the algorithm. Defaults to -1.

#### Attributes

<b>hyper-parameter_ranges</b>	{ "eta": Real(0.000001, 1), "max_depth": Integer(1, 20), "min_child_weight": Real(1, 10), "n_estimators": Integer(1, 1000), }
<b>model_family</b>	ModelFamily.XGBOOST
<b>modifies_features</b>	True
<b>modifies_target</b>	False
<b>name</b>	XGBoost Regressor
<b>predict_uses_y</b>	False
<b>SEED_MAX</b>	None
<b>SEED_MIN</b>	None
<b>supported_problem_types</b>	[ ProblemTypes.REGRESSION, ProblemTypes.TIME_SERIES_REGRESSION, ]

#### Methods

<i>clone</i>	Constructs a new component with the same parameters and random state.
<i>default_parameters</i>	Returns the default parameters for this component.
<i>describe</i>	Describe a component and its parameters
<i>feature_importance</i>	Returns importance associated with each feature.
<i>fit</i>	Fits component to data
<i>load</i>	Loads component at file path
<i>needs_fitting</i>	Returns boolean determining if component needs fitting before

continues on next page

Table 401 – continued from previous page

<i>parameters</i>	Returns the parameters which were used to initialize the component
<i>predict</i>	Make predictions using selected features.
<i>predict_proba</i>	Make probability estimates for labels.
<i>save</i>	Saves component at file path

**clone** (*self*)

Constructs a new component with the same parameters and random state.

**Returns** A new instance of this component with identical parameters and random state.

**default\_parameters** (*cls*)

Returns the default parameters for this component.

Our convention is that `Component.default_parameters == Component().parameters`.

**Returns** default parameters for this component.

**Return type** dict

**describe** (*self*, *print\_name=False*, *return\_dict=False*)

Describe a component and its parameters

**Parameters**

- **print\_name** (*bool*, *optional*) – whether to print name of component
- **return\_dict** (*bool*, *optional*) – whether to return description as dictionary in the format {"name": name, "parameters": parameters}

**Returns** prints and returns dictionary

**Return type** None or dict

**property feature\_importance** (*self*)

Returns importance associated with each feature.

**Returns** Importance associated with each feature

**Return type** np.ndarray

**fit** (*self*, *X*, *y=None*)

Fits component to data

**Parameters**

- **X** (*list*, *pd.DataFrame* or *np.ndarray*) – The input training data of shape [n\_samples, n\_features]
- **y** (*list*, *pd.Series*, *np.ndarray*, *optional*) – The target training data of length [n\_samples]

**Returns** self

**static load** (*file\_path*)

Loads component at file path

**Parameters** **file\_path** (*str*) – Location to load file

**Returns** ComponentBase object

**needs\_fitting** (*self*)

Returns boolean determining if component needs fitting before calling predict, predict\_proba, transform,



or `feature_importances`. This can be overridden to `False` for components that do not need to be fit or whose fit methods do nothing.

**property** `parameters` (*self*)

Returns the parameters which were used to initialize the component

**predict** (*self*, *X*)

Make predictions using selected features.

**Parameters** *X* (*pd.DataFrame*, *np.ndarray*) – Data of shape [n\_samples, n\_features]

**Returns** Predicted values

**Return type** *pd.Series*

**predict\_proba** (*self*, *X*)

Make probability estimates for labels.

**Parameters** *X* (*pd.DataFrame*, *or np.ndarray*) – Features

**Returns** Probability estimates

**Return type** *pd.Series*

**save** (*self*, *file\_path*, *pickle\_protocol=cloudpickle.DEFAULT\_PROTOCOL*)

Saves component at file path

**Parameters**

- **file\_path** (*str*) – Location to save file
- **pickle\_protocol** (*int*) – The pickle data stream format.

**Returns** `None`

## transformers

## Subpackages

## dimensionality\_reduction

## Submodules

## Ida

## Module Contents

## Classes Summary

---

*LinearDiscriminantAnalysis*

Reduces the number of features by using Linear Discriminant Analysis.

---

## Contents

**class** evalml.pipelines.components.transformers.dimensionality\_reduction.lda.**LinearDiscrimin**

Reduces the number of features by using Linear Discriminant Analysis.

### Parameters

- **n\_components** (*int*) – The number of features to maintain after computation. Defaults to None.
- **random\_seed** (*int*) – Seed for the random number generator. Defaults to 0.

### Attributes

<b>hyper-parameter_ranges</b>	{}
<b>model_family</b>	ModelFamily.NONE
<b>modifies_features</b>	True
<b>modifies_target</b>	False
<b>name</b>	Linear Discriminant Analysis Transformer

### Methods

<i>clone</i>	Constructs a new component with the same parameters and random state.
<i>default_parameters</i>	Returns the default parameters for this component.
<i>describe</i>	Describe a component and its parameters
<i>fit</i>	Fits component to data
<i>fit_transform</i>	Fits on X and transforms X
<i>load</i>	Loads component at file path
<i>needs_fitting</i>	Returns boolean determining if component needs fitting before
<i>parameters</i>	Returns the parameters which were used to initialize the component
<i>save</i>	Saves component at file path
<i>transform</i>	Transforms data X.

**clone** (*self*)

Constructs a new component with the same parameters and random state.

**Returns** A new instance of this component with identical parameters and random state.

**default\_parameters** (*cls*)

Returns the default parameters for this component.

Our convention is that `Component.default_parameters == Component().parameters`.

**Returns** default parameters for this component.

**Return type** dict

**describe** (*self*, *print\_name=False*, *return\_dict=False*)

Describe a component and its parameters

**Parameters**

- **print\_name** (*bool*, *optional*) – whether to print name of component
- **return\_dict** (*bool*, *optional*) – whether to return description as dictionary in the format {"name": name, "parameters": parameters}

**Returns** prints and returns dictionary

**Return type** None or dict

**fit** (*self*, *X*, *y*)

Fits component to data

**Parameters**

- **X** (*list*, *pd.DataFrame* or *np.ndarray*) – The input training data of shape [n\_samples, n\_features]
- **y** (*list*, *pd.Series*, *np.ndarray*, *optional*) – The target training data of length [n\_samples]

**Returns** self

**fit\_transform** (*self*, *X*, *y=None*)

Fits on X and transforms X

**Parameters**

- **X** (*pd.DataFrame*) – Data to fit and transform
- **y** (*pd.Series*) – Target data

**Returns** Transformed X

**Return type** pd.DataFrame

**static load** (*file\_path*)

Loads component at file path

**Parameters** **file\_path** (*str*) – Location to load file

**Returns** ComponentBase object

**needs\_fitting** (*self*)

Returns boolean determining if component needs fitting before calling predict, predict\_proba, transform, or feature\_importances. This can be overridden to False for components that do not need to be fit or whose fit methods do nothing.

**property parameters** (*self*)

Returns the parameters which were used to initialize the component

**save** (*self*, *file\_path*, *pickle\_protocol=cloudpickle.DEFAULT\_PROTOCOL*)

Saves component at file path

**Parameters**

- **file\_path** (*str*) – Location to save file
- **pickle\_protocol** (*int*) – The pickle data stream format.

**Returns** None

**transform** (*self*, *X*, *y=None*)  
Transforms data *X*.

**Parameters**

- **x** (*pd.DataFrame*) – Data to transform.
- **y** (*pd.Series*, *optional*) – Target data.

**Returns** Transformed *X*

**Return type** *pd.DataFrame*

pca

Module Contents

Classes Summary

<i>PCA</i>	Reduces the number of features by using Principal Component Analysis (PCA).
------------	---

Contents

**class** evalml.pipelines.components.transformers.dimensionality\_reduction.pca.**PCA** (*variance=0.95*, *n\_components=None*, *random\_seed=0*, *\*\*kwargs*)

Reduces the number of features by using Principal Component Analysis (PCA).

**Parameters**

- **variance** (*float*) – The percentage of the original data variance that should be preserved when reducing the number of features. Defaults to 0.95.
- **n\_components** (*int*) – The number of features to maintain after computing SVD. Defaults to *None*, but will override variance variable if set.
- **random\_seed** (*int*) – Seed for the random number generator. Defaults to 0.

**Attributes**

<b>hyper-parameter_ranges</b>	Real(0.25, 1)}:type: {"variance"}
<b>model_family</b>	ModelFamily.NONE
<b>modifies_features</b>	True
<b>modifies_target</b>	False
<b>name</b>	PCA Transformer

**Methods**

<code>clone</code>	Constructs a new component with the same parameters and random state.
<code>default_parameters</code>	Returns the default parameters for this component.
<code>describe</code>	Describe a component and its parameters
<code>fit</code>	Fits component to data
<code>fit_transform</code>	Fits on X and transforms X
<code>load</code>	Loads component at file path
<code>needs_fitting</code>	Returns boolean determining if component needs fitting before
<code>parameters</code>	Returns the parameters which were used to initialize the component
<code>save</code>	Saves component at file path
<code>transform</code>	Transforms data X.

**clone** (*self*)

Constructs a new component with the same parameters and random state.

**Returns** A new instance of this component with identical parameters and random state.

**default\_parameters** (*cls*)

Returns the default parameters for this component.

Our convention is that `Component.default_parameters == Component().parameters`.

**Returns** default parameters for this component.

**Return type** dict

**describe** (*self*, *print\_name=False*, *return\_dict=False*)

Describe a component and its parameters

**Parameters**

- **print\_name** (*bool*, *optional*) – whether to print name of component
- **return\_dict** (*bool*, *optional*) – whether to return description as dictionary in the format {"name": name, "parameters": parameters}

**Returns** prints and returns dictionary

**Return type** None or dict

**fit** (*self*, *X*, *y=None*)

Fits component to data

**Parameters**

- **X** (*list*, *pd.DataFrame* or *np.ndarray*) – The input training data of shape [n\_samples, n\_features]
- **y** (*list*, *pd.Series*, *np.ndarray*, *optional*) – The target training data of length [n\_samples]

**Returns** self

**fit\_transform** (*self*, *X*, *y=None*)

Fits on X and transforms X

**Parameters**

- **X** (*pd.DataFrame*) – Data to fit and transform

- **y** (*pd.Series*) – Target data

**Returns** Transformed X

**Return type** *pd.DataFrame*

**static load** (*file\_path*)

Loads component at file path

**Parameters** **file\_path** (*str*) – Location to load file

**Returns** *ComponentBase* object

**needs\_fitting** (*self*)

Returns boolean determining if component needs fitting before calling *predict*, *predict\_proba*, *transform*, or *feature\_importances*. This can be overridden to *False* for components that do not need to be fit or whose fit methods do nothing.

**property parameters** (*self*)

Returns the parameters which were used to initialize the component

**save** (*self*, *file\_path*, *pickle\_protocol=cloudpickle.DEFAULT\_PROTOCOL*)

Saves component at file path

**Parameters**

- **file\_path** (*str*) – Location to save file
- **pickle\_protocol** (*int*) – The pickle data stream format.

**Returns** *None*

**transform** (*self*, *X*, *y=None*)

Transforms data *X*.

**Parameters**

- **x** (*pd.DataFrame*) – Data to transform.
- **y** (*pd.Series*, *optional*) – Target data.

**Returns** Transformed X

**Return type** *pd.DataFrame*

## Package Contents

## Classes Summary

---

<i>LinearDiscriminantAnalysis</i>	Reduces the number of features by using Linear Discriminant Analysis.
<i>PCA</i>	Reduces the number of features by using Principal Component Analysis (PCA).

---

## Contents

**class** evalml.pipelines.components.transformers.dimensionality\_reduction.**LinearDiscriminantAnalysis**

Reduces the number of features by using Linear Discriminant Analysis.

### Parameters

- **n\_components** (*int*) – The number of features to maintain after computation. Defaults to None.
- **random\_seed** (*int*) – Seed for the random number generator. Defaults to 0.

### Attributes

<b>hyper-parameter_ranges</b>	{}
<b>model_family</b>	ModelFamily.NONE
<b>modifies_features</b>	True
<b>modifies_target</b>	False
<b>name</b>	Linear Discriminant Analysis Transformer

### Methods

<i>clone</i>	Constructs a new component with the same parameters and random state.
<i>default_parameters</i>	Returns the default parameters for this component.
<i>describe</i>	Describe a component and its parameters
<i>fit</i>	Fits component to data
<i>fit_transform</i>	Fits on X and transforms X
<i>load</i>	Loads component at file path
<i>needs_fitting</i>	Returns boolean determining if component needs fitting before
<i>parameters</i>	Returns the parameters which were used to initialize the component
<i>save</i>	Saves component at file path
<i>transform</i>	Transforms data X.

**clone** (*self*)

Constructs a new component with the same parameters and random state.

**Returns** A new instance of this component with identical parameters and random state.

**default\_parameters** (*cls*)

Returns the default parameters for this component.

Our convention is that `Component.default_parameters == Component().parameters`.

**Returns** default parameters for this component.

**Return type** dict

**describe** (*self*, *print\_name=False*, *return\_dict=False*)

Describe a component and its parameters

**Parameters**

- **print\_name** (*bool*, *optional*) – whether to print name of component
- **return\_dict** (*bool*, *optional*) – whether to return description as dictionary in the format {"name": name, "parameters": parameters}

**Returns** prints and returns dictionary

**Return type** None or dict

**fit** (*self*, *X*, *y*)

Fits component to data

**Parameters**

- **X** (*list*, *pd.DataFrame* or *np.ndarray*) – The input training data of shape [n\_samples, n\_features]
- **y** (*list*, *pd.Series*, *np.ndarray*, *optional*) – The target training data of length [n\_samples]

**Returns** self

**fit\_transform** (*self*, *X*, *y=None*)

Fits on X and transforms X

**Parameters**

- **X** (*pd.DataFrame*) – Data to fit and transform
- **y** (*pd.Series*) – Target data

**Returns** Transformed X

**Return type** pd.DataFrame

**static load** (*file\_path*)

Loads component at file path

**Parameters** **file\_path** (*str*) – Location to load file

**Returns** ComponentBase object

**needs\_fitting** (*self*)

Returns boolean determining if component needs fitting before calling predict, predict\_proba, transform, or feature\_importances. This can be overridden to False for components that do not need to be fit or whose fit methods do nothing.

**property parameters** (*self*)

Returns the parameters which were used to initialize the component

**save** (*self*, *file\_path*, *pickle\_protocol=cloudpickle.DEFAULT\_PROTOCOL*)

Saves component at file path

**Parameters**

- **file\_path** (*str*) – Location to save file
- **pickle\_protocol** (*int*) – The pickle data stream format.

**Returns** None



**transform** (*self*, *X*, *y=None*)  
Transforms data *X*.

#### Parameters

- **x** (*pd.DataFrame*) – Data to transform.
- **y** (*pd.Series*, *optional*) – Target data.

**Returns** Transformed *X*

**Return type** *pd.DataFrame*

**class** evalml.pipelines.components.transformers.dimensionality\_reduction.**PCA** (*variance=0.95*,  
*n\_components=None*,  
*random\_seed=0*,  
*\*\*kwargs*)

Reduces the number of features by using Principal Component Analysis (PCA).

#### Parameters

- **variance** (*float*) – The percentage of the original data variance that should be preserved when reducing the number of features. Defaults to 0.95.
- **n\_components** (*int*) – The number of features to maintain after computing SVD. Defaults to *None*, but will override *variance* variable if set.
- **random\_seed** (*int*) – Seed for the random number generator. Defaults to 0.

#### Attributes

<b>hyper-parameter_ranges</b>	Real(0.25, 1)}:type: {"variance"}
<b>model_family</b>	ModelFamily.NONE
<b>modifies_features</b>	True
<b>modifies_target</b>	False
<b>name</b>	PCA Transformer

#### Methods

<i>clone</i>	Constructs a new component with the same parameters and random state.
<i>default_parameters</i>	Returns the default parameters for this component.
<i>describe</i>	Describe a component and its parameters
<i>fit</i>	Fits component to data
<i>fit_transform</i>	Fits on <i>X</i> and transforms <i>X</i>
<i>load</i>	Loads component at file path
<i>needs_fitting</i>	Returns boolean determining if component needs fitting before
<i>parameters</i>	Returns the parameters which were used to initialize the component
<i>save</i>	Saves component at file path
<i>transform</i>	Transforms data <i>X</i> .

**clone** (*self*)

Constructs a new component with the same parameters and random state.

**Returns** A new instance of this component with identical parameters and random state.

**default\_parameters** (*cls*)

Returns the default parameters for this component.

Our convention is that `Component.default_parameters == Component().parameters`.

**Returns** default parameters for this component.

**Return type** dict

**describe** (*self*, *print\_name=False*, *return\_dict=False*)

Describe a component and its parameters

**Parameters**

- **print\_name** (*bool*, *optional*) – whether to print name of component
- **return\_dict** (*bool*, *optional*) – whether to return description as dictionary in the format {"name": name, "parameters": parameters}

**Returns** prints and returns dictionary

**Return type** None or dict

**fit** (*self*, *X*, *y=None*)

Fits component to data

**Parameters**

- **X** (*list*, *pd.DataFrame* or *np.ndarray*) – The input training data of shape [n\_samples, n\_features]
- **y** (*list*, *pd.Series*, *np.ndarray*, *optional*) – The target training data of length [n\_samples]

**Returns** self

**fit\_transform** (*self*, *X*, *y=None*)

Fits on X and transforms X

**Parameters**

- **X** (*pd.DataFrame*) – Data to fit and transform
- **y** (*pd.Series*) – Target data

**Returns** Transformed X

**Return type** pd.DataFrame

**static load** (*file\_path*)

Loads component at file path

**Parameters** **file\_path** (*str*) – Location to load file

**Returns** ComponentBase object

**needs\_fitting** (*self*)

Returns boolean determining if component needs fitting before calling predict, predict\_proba, transform, or feature\_importances. This can be overridden to False for components that do not need to be fit or whose fit methods do nothing.

**property parameters** (*self*)

Returns the parameters which were used to initialize the component

**save** (*self*, *file\_path*, *pickle\_protocol*=*cloudpickle.DEFAULT\_PROTOCOL*)

Saves component at file path

**Parameters**

- **file\_path** (*str*) – Location to save file
- **pickle\_protocol** (*int*) – The pickle data stream format.

**Returns** None

**transform** (*self*, *X*, *y*=None)

Transforms data X.

**Parameters**

- **x** (*pd.DataFrame*) – Data to transform.
- **y** (*pd.Series*, *optional*) – Target data.

**Returns** Transformed X

**Return type** *pd.DataFrame*

## encoders

### Submodules

### onehot\_encoder

### Module Contents

### Classes Summary

<i>OneHotEncoder</i>	A transformer that encodes categorical features in a one-hot numeric array.
<i>OneHotEncoderMeta</i>	A version of the ComponentBaseMeta class which includes validation on an additional one-hot-encoder-specific method <i>categories</i> .

Contents

`class evalml.pipelines.components.transformers.encoders.onehot_encoder.OneHotEncoder` (*top\_n=1, features\_to\_encode=None, categories=None, drop='if\_binary', handle\_unknown='ignore', handle\_missing='error', random\_seed=None, \*\*kwargs*)

A transformer that encodes categorical features in a one-hot numeric array.

Parameters

- **top\_n** (*int*) – Number of categories per column to encode. If *None*, all categories will be encoded. Otherwise, the *n* most frequent will be encoded and all others will be dropped. Defaults to 10.
- **features\_to\_encode** (*list[str]*) – List of columns to encode. All other columns will remain untouched. If *None*, all appropriate columns will be encoded. Defaults to *None*.
- **categories** (*list*) – A two dimensional list of categories, where *categories[i]* is a list of the categories for the column at index *i*. This can also be *None*, or “*auto*” if *top\_n* is not *None*. Defaults to *None*.
- **drop** (*string, list*) – Method (“*first*” or “*if\_binary*”) to use to drop one category per feature. Can also be a list specifying which categories to drop for each feature. Defaults to “*if\_binary*”.
- **handle\_unknown** (*string*) – Whether to ignore or error for unknown categories for a feature encountered during *fit* or *transform*. If either *top\_n* or *categories* is used to limit the number of categories per column, this must be “*ignore*”. Defaults to “*ignore*”.
- **handle\_missing** (*string*) – Options for how to handle missing (NaN) values encountered during *fit* or *transform*. If this is set to “*as\_category*” and NaN values are within the *n* most frequent, “*nan*” values will be encoded as their own column. If this is set to “*error*”, any missing values encountered will raise an error. Defaults to “*error*”.
- **random\_seed** (*int*) – Seed for the random number generator. Defaults to 0.

Attributes

hyper-parameter_ranges	{}
model_family	ModelFamily.NONE
modifies_features	True
modifies_target	False
name	One Hot Encoder

**Methods**

<code>categories</code>	Returns a list of the unique categories to be encoded for the particular feature, in order.
<code>clone</code>	Constructs a new component with the same parameters and random state.
<code>default_parameters</code>	Returns the default parameters for this component.
<code>describe</code>	Describe a component and its parameters
<code>fit</code>	Fits component to data
<code>fit_transform</code>	Fits on $X$ and transforms $X$
<code>get_feature_names</code>	Return feature names for the categorical features after fitting.
<code>load</code>	Loads component at file path
<code>needs_fitting</code>	Returns boolean determining if component needs fitting before
<code>parameters</code>	Returns the parameters which were used to initialize the component
<code>save</code>	Saves component at file path
<code>transform</code>	One-hot encode the input data.

**categories** (*self*, *feature\_name*)

Returns a list of the unique categories to be encoded for the particular feature, in order.

**Parameters** **feature\_name** (*str*) – the name of any feature provided to one-hot encoder during fit

**Returns** the unique categories, in the same dtype as they were provided during fit

**Return type** `np.ndarray`

**clone** (*self*)

Constructs a new component with the same parameters and random state.

**Returns** A new instance of this component with identical parameters and random state.

**default\_parameters** (*cls*)

Returns the default parameters for this component.

Our convention is that `Component.default_parameters == Component().parameters`.

**Returns** default parameters for this component.

**Return type** `dict`

**describe** (*self*, *print\_name=False*, *return\_dict=False*)

Describe a component and its parameters

**Parameters**

- **print\_name** (*bool*, *optional*) – whether to print name of component
- **return\_dict** (*bool*, *optional*) – whether to return description as dictionary in the format {"name": name, "parameters": parameters}

**Returns** prints and returns dictionary

**Return type** `None` or `dict`

**fit** (*self*,  $X$ , *y=None*)

Fits component to data

**Parameters**

- **X** (*list*, *pd.DataFrame* or *np.ndarray*) – The input training data of shape [n\_samples, n\_features]
- **y** (*list*, *pd.Series*, *np.ndarray*, *optional*) – The target training data of length [n\_samples]

**Returns** self**fit\_transform** (*self*, *X*, *y=None*)

Fits on X and transforms X

**Parameters**

- **X** (*pd.DataFrame*) – Data to fit and transform
- **y** (*pd.Series*) – Target data

**Returns** Transformed X**Return type** *pd.DataFrame***get\_feature\_names** (*self*)

Return feature names for the categorical features after fitting.

Feature names are formatted as {column name}\_{category name}. In the event of a duplicate name, an integer will be added at the end of the feature name to distinguish it.

For example, consider a dataframe with a column called “A” and category “x\_y” and another column called “A\_x” with “y”. In this example, the feature names would be “A\_x\_y” and “A\_x\_y\_1”.

**Returns** The feature names after encoding, provided in the same order as input\_features.**Return type** *np.ndarray***static load** (*file\_path*)

Loads component at file path

**Parameters** **file\_path** (*str*) – Location to load file**Returns** ComponentBase object**needs\_fitting** (*self*)

Returns boolean determining if component needs fitting before calling predict, predict\_proba, transform, or feature\_importances. This can be overridden to False for components that do not need to be fit or whose fit methods do nothing.

**property parameters** (*self*)

Returns the parameters which were used to initialize the component

**save** (*self*, *file\_path*, *pickle\_protocol=cloudpickle.DEFAULT\_PROTOCOL*)

Saves component at file path

**Parameters**

- **file\_path** (*str*) – Location to save file
- **pickle\_protocol** (*int*) – The pickle data stream format.

**Returns** None**transform** (*self*, *X*, *y=None*)

One-hot encode the input data.

**Parameters**

- **X** (*pd.DataFrame*) – Features to one-hot encode.
- **y** (*pd.Series*) – Ignored.

**Returns** Transformed data, where each categorical feature has been encoded into numerical columns using one-hot encoding.

**Return type** *pd.DataFrame*

**class** `evalml.pipelines.components.transformers.encoders.onehot_encoder.OneHotEncoderMeta`  
A version of the `ComponentBaseMeta` class which includes validation on an additional one-hot-encoder-specific method *categories*.

#### Attributes

<b>FIT_METHODS</b>	['fit', 'fit_transform']
<b>METHODS_TO_CHECK</b>	None
<b>PROPERTIES_TO_CHECK</b>	['feature_importance']

#### Methods

<i>check_for_fit</i>	<i>check_for_fit</i> wraps a method that validates if <i>self._is_fitted</i> is <i>True</i> .
<i>register</i>	Register a virtual subclass of an ABC.
<i>set_fit</i>	

**classmethod** `check_for_fit (cls, method)`

*check\_for\_fit* wraps a method that validates if *self.\_is\_fitted* is *True*. It raises an exception if *False* and calls and returns the wrapped method if *True*.

**register** (*cls, subclass*)

Register a virtual subclass of an ABC.

Returns the subclass, to allow usage as a class decorator.

**classmethod** `set_fit (cls, method)`

## target\_encoder

### Module Contents

### Classes Summary

<i>TargetEncoder</i>	A transformer that encodes categorical features into target encodings.
----------------------	--

Contents

```
class evalml.pipelines.components.transformers.encoders.target_encoder.TargetEncoder (cols=None, smoothing=1.0, handle_unknown='value', handle_missing='value', random_seed=None, **kwargs)
```

A transformer that encodes categorical features into target encodings.

Parameters

- **cols** (*list*) – Columns to encode. If None, all string columns will be encoded, otherwise only the columns provided will be encoded. Defaults to None
- **smoothing** (*float*) – The smoothing factor to apply. The larger this value is, the more influence the expected target value has on the resulting target encodings. Must be strictly larger than 0. Defaults to 1.0
- **handle\_unknown** (*string*) – Determines how to handle unknown categories for a feature encountered. Options are ‘value’, ‘error’, and ‘return\_nan’. Defaults to ‘value’, which replaces with the target mean
- **handle\_missing** (*string*) – Determines how to handle missing values encountered during *fit* or *transform*. Options are ‘value’, ‘error’, and ‘return\_nan’. Defaults to ‘value’, which replaces with the target mean
- **random\_seed** (*int*) – Seed for the random number generator. Defaults to 0.

Attributes

hyper-parameter_ranges	{}
model_family	ModelFamily.NONE
modifies_features	True
modifies_target	False
name	Target Encoder

Methods

<i>clone</i>	Constructs a new component with the same parameters and random state.
<i>default_parameters</i>	Returns the default parameters for this component.
<i>describe</i>	Describe a component and its parameters
<i>fit</i>	Fits component to data
<i>fit_transform</i>	Fits on X and transforms X
<i>get_feature_names</i>	Return feature names for the input features after fitting.

continues on next page



Table 413 – continued from previous page

<i>load</i>	Loads component at file path
<i>needs_fitting</i>	Returns boolean determining if component needs fitting before
<i>parameters</i>	Returns the parameters which were used to initialize the component
<i>save</i>	Saves component at file path
<i>transform</i>	Transforms data X.

**clone** (*self*)

Constructs a new component with the same parameters and random state.

**Returns** A new instance of this component with identical parameters and random state.

**default\_parameters** (*cls*)

Returns the default parameters for this component.

Our convention is that `Component.default_parameters == Component().parameters`.

**Returns** default parameters for this component.

**Return type** dict

**describe** (*self*, *print\_name=False*, *return\_dict=False*)

Describe a component and its parameters

**Parameters**

- **print\_name** (*bool*, *optional*) – whether to print name of component
- **return\_dict** (*bool*, *optional*) – whether to return description as dictionary in the format {"name": name, "parameters": parameters}

**Returns** prints and returns dictionary

**Return type** None or dict

**fit** (*self*, *X*, *y*)

Fits component to data

**Parameters**

- **X** (*list*, *pd.DataFrame* or *np.ndarray*) – The input training data of shape [n\_samples, n\_features]
- **y** (*list*, *pd.Series*, *np.ndarray*, *optional*) – The target training data of length [n\_samples]

**Returns** self

**fit\_transform** (*self*, *X*, *y*)

Fits on X and transforms X

**Parameters**

- **X** (*pd.DataFrame*) – Data to fit and transform
- **y** (*pd.Series*) – Target data

**Returns** Transformed X

**Return type** pd.DataFrame

**get\_feature\_names** (*self*)

Return feature names for the input features after fitting.

**Returns** The feature names after encoding

**Return type** np.array

**static load** (*file\_path*)

Loads component at file path

**Parameters** **file\_path** (*str*) – Location to load file

**Returns** ComponentBase object

**needs\_fitting** (*self*)

Returns boolean determining if component needs fitting before calling predict, predict\_proba, transform, or feature\_importances. This can be overridden to False for components that do not need to be fit or whose fit methods do nothing.

**property parameters** (*self*)

Returns the parameters which were used to initialize the component

**save** (*self*, *file\_path*, *pickle\_protocol*=cloudpickle.DEFAULT\_PROTOCOL)

Saves component at file path

**Parameters**

- **file\_path** (*str*) – Location to save file
- **pickle\_protocol** (*int*) – The pickle data stream format.

**Returns** None

**transform** (*self*, *X*, *y=None*)

Transforms data X.

**Parameters**

- **X** (*pd.DataFrame*) – Data to transform.
- **y** (*pd.Series*, *optional*) – Target data.

**Returns** Transformed X

**Return type** pd.DataFrame

## Package Contents

### Classes Summary

<i>OneHotEncoder</i>	A transformer that encodes categorical features in a one-hot numeric array.
<i>TargetEncoder</i>	A transformer that encodes categorical features into target encodings.

## Contents

```
class evalml.pipelines.components.transformers.encoders.OneHotEncoder (top_n=10,
                                                                    fea-
                                                                    tures_to_encode=None,
                                                                    cate-
                                                                    gories=None,
                                                                    drop='if_binary',
                                                                    han-
                                                                    dle_unknown='ignore',
                                                                    han-
                                                                    dle_missing='error',
                                                                    ran-
                                                                    dom_seed=0,
                                                                    **kwargs)
```

A transformer that encodes categorical features in a one-hot numeric array.

### Parameters

- **top\_n** (*int*) – Number of categories per column to encode. If *None*, all categories will be encoded. Otherwise, the *n* most frequent will be encoded and all others will be dropped. Defaults to 10.
- **features\_to\_encode** (*list[str]*) – List of columns to encode. All other columns will remain untouched. If *None*, all appropriate columns will be encoded. Defaults to *None*.
- **categories** (*list*) – A two dimensional list of categories, where *categories[i]* is a list of the categories for the column at index *i*. This can also be *None*, or “auto” if *top\_n* is not *None*. Defaults to *None*.
- **drop** (*string, list*) – Method (“first” or “if\_binary”) to use to drop one category per feature. Can also be a list specifying which categories to drop for each feature. Defaults to “if\_binary”.
- **handle\_unknown** (*string*) – Whether to ignore or error for unknown categories for a feature encountered during *fit* or *transform*. If either *top\_n* or *categories* is used to limit the number of categories per column, this must be “ignore”. Defaults to “ignore”.
- **handle\_missing** (*string*) – Options for how to handle missing (NaN) values encountered during *fit* or *transform*. If this is set to “as\_category” and NaN values are within the *n* most frequent, “nan” values will be encoded as their own column. If this is set to “error”, any missing values encountered will raise an error. Defaults to “error”.
- **random\_seed** (*int*) – Seed for the random number generator. Defaults to 0.

### Attributes

hyper-parameter_ranges	{}
model_family	ModelFamily.NONE
modifies_features	True
modifies_target	False
name	One Hot Encoder

### Methods

<code>categories</code>	Returns a list of the unique categories to be encoded for the particular feature, in order.
<code>clone</code>	Constructs a new component with the same parameters and random state.
<code>default_parameters</code>	Returns the default parameters for this component.
<code>describe</code>	Describe a component and its parameters
<code>fit</code>	Fits component to data
<code>fit_transform</code>	Fits on X and transforms X
<code>get_feature_names</code>	Return feature names for the categorical features after fitting.
<code>load</code>	Loads component at file path
<code>needs_fitting</code>	Returns boolean determining if component needs fitting before
<code>parameters</code>	Returns the parameters which were used to initialize the component
<code>save</code>	Saves component at file path
<code>transform</code>	One-hot encode the input data.

**categories** (*self*, *feature\_name*)

Returns a list of the unique categories to be encoded for the particular feature, in order.

**Parameters** **feature\_name** (*str*) – the name of any feature provided to one-hot encoder during fit

**Returns** the unique categories, in the same dtype as they were provided during fit

**Return type** np.ndarray

**clone** (*self*)

Constructs a new component with the same parameters and random state.

**Returns** A new instance of this component with identical parameters and random state.

**default\_parameters** (*cls*)

Returns the default parameters for this component.

Our convention is that `Component.default_parameters == Component().parameters`.

**Returns** default parameters for this component.

**Return type** dict

**describe** (*self*, *print\_name=False*, *return\_dict=False*)

Describe a component and its parameters

**Parameters**

- **print\_name** (*bool*, *optional*) – whether to print name of component
- **return\_dict** (*bool*, *optional*) – whether to return description as dictionary in the format {"name": name, "parameters": parameters}

**Returns** prints and returns dictionary

**Return type** None or dict

**fit** (*self*, *X*, *y=None*)

Fits component to data

**Parameters**

- **X** (*list, pd.DataFrame or np.ndarray*) – The input training data of shape [n\_samples, n\_features]
- **y** (*list, pd.Series, np.ndarray, optional*) – The target training data of length [n\_samples]

**Returns** self

**fit\_transform** (*self, X, y=None*)

Fits on X and transforms X

**Parameters**

- **X** (*pd.DataFrame*) – Data to fit and transform
- **y** (*pd.Series*) – Target data

**Returns** Transformed X

**Return type** pd.DataFrame

**get\_feature\_names** (*self*)

Return feature names for the categorical features after fitting.

Feature names are formatted as {column name}\_{category name}. In the event of a duplicate name, an integer will be added at the end of the feature name to distinguish it.

For example, consider a dataframe with a column called “A” and category “x\_y” and another column called “A\_x” with “y”. In this example, the feature names would be “A\_x\_y” and “A\_x\_y\_1”.

**Returns** The feature names after encoding, provided in the same order as input\_features.

**Return type** np.ndarray

**static load** (*file\_path*)

Loads component at file path

**Parameters** **file\_path** (*str*) – Location to load file

**Returns** ComponentBase object

**needs\_fitting** (*self*)

Returns boolean determining if component needs fitting before calling predict, predict\_proba, transform, or feature\_importances. This can be overridden to False for components that do not need to be fit or whose fit methods do nothing.

**property parameters** (*self*)

Returns the parameters which were used to initialize the component

**save** (*self, file\_path, pickle\_protocol=cloudpickle.DEFAULT\_PROTOCOL*)

Saves component at file path

**Parameters**

- **file\_path** (*str*) – Location to save file
- **pickle\_protocol** (*int*) – The pickle data stream format.

**Returns** None

**transform** (*self, X, y=None*)

One-hot encode the input data.

**Parameters**

- **X** (*pd.DataFrame*) – Features to one-hot encode.

- **y** (*pd.Series*) – Ignored.

**Returns** Transformed data, where each categorical feature has been encoded into numerical columns using one-hot encoding.

**Return type** `pd.DataFrame`

```
class evalml.pipelines.components.transformers.encoders.TargetEncoder (cols=None,
                                                                    smoothing=1.0,
                                                                    handle_unknown='value',
                                                                    handle_missing='value',
                                                                    random_seed=0,
                                                                    **kwargs)
```

A transformer that encodes categorical features into target encodings.

#### Parameters

- **cols** (*list*) – Columns to encode. If `None`, all string columns will be encoded, otherwise only the columns provided will be encoded. Defaults to `None`
- **smoothing** (*float*) – The smoothing factor to apply. The larger this value is, the more influence the expected target value has on the resulting target encodings. Must be strictly larger than 0. Defaults to 1.0
- **handle\_unknown** (*string*) – Determines how to handle unknown categories for a feature encountered. Options are ‘value’, ‘error’, and ‘return\_nan’. Defaults to ‘value’, which replaces with the target mean
- **handle\_missing** (*string*) – Determines how to handle missing values encountered during *fit* or *transform*. Options are ‘value’, ‘error’, and ‘return\_nan’. Defaults to ‘value’, which replaces with the target mean
- **random\_seed** (*int*) – Seed for the random number generator. Defaults to 0.

#### Attributes

<b>hyper-parameter_ranges</b>	<code>{}</code>
<b>model_family</b>	<code>ModelFamily.NONE</code>
<b>modifies_features</b>	<code>True</code>
<b>modifies_target</b>	<code>False</code>
<b>name</b>	Target Encoder

#### Methods

<code>clone</code>	Constructs a new component with the same parameters and random state.
<code>default_parameters</code>	Returns the default parameters for this component.
<code>describe</code>	Describe a component and its parameters
<code>fit</code>	Fits component to data

continues on next page

Table 416 – continued from previous page

<i>fit_transform</i>	Fits on X and transforms X
<i>get_feature_names</i>	Return feature names for the input features after fitting.
<i>load</i>	Loads component at file path
<i>needs_fitting</i>	Returns boolean determining if component needs fitting before
<i>parameters</i>	Returns the parameters which were used to initialize the component
<i>save</i>	Saves component at file path
<i>transform</i>	Transforms data X.

**clone** (*self*)

Constructs a new component with the same parameters and random state.

**Returns** A new instance of this component with identical parameters and random state.

**default\_parameters** (*cls*)

Returns the default parameters for this component.

Our convention is that `Component.default_parameters == Component().parameters`.

**Returns** default parameters for this component.

**Return type** dict

**describe** (*self*, *print\_name=False*, *return\_dict=False*)

Describe a component and its parameters

**Parameters**

- **print\_name** (*bool*, *optional*) – whether to print name of component
- **return\_dict** (*bool*, *optional*) – whether to return description as dictionary in the format {"name": name, "parameters": parameters}

**Returns** prints and returns dictionary

**Return type** None or dict

**fit** (*self*, *X*, *y*)

Fits component to data

**Parameters**

- **X** (*list*, *pd.DataFrame* or *np.ndarray*) – The input training data of shape [n\_samples, n\_features]
- **y** (*list*, *pd.Series*, *np.ndarray*, *optional*) – The target training data of length [n\_samples]

**Returns** self

**fit\_transform** (*self*, *X*, *y*)

Fits on X and transforms X

**Parameters**

- **X** (*pd.DataFrame*) – Data to fit and transform
- **y** (*pd.Series*) – Target data

**Returns** Transformed X

**Return type** `pd.DataFrame`

**get\_feature\_names** (*self*)

Return feature names for the input features after fitting.

**Returns** The feature names after encoding

**Return type** `np.array`

**static load** (*file\_path*)

Loads component at file path

**Parameters** **file\_path** (*str*) – Location to load file

**Returns** `ComponentBase` object

**needs\_fitting** (*self*)

Returns boolean determining if component needs fitting before calling `predict`, `predict_proba`, `transform`, or `feature_importances`. This can be overridden to `False` for components that do not need to be fit or whose fit methods do nothing.

**property parameters** (*self*)

Returns the parameters which were used to initialize the component

**save** (*self*, *file\_path*, *pickle\_protocol=cloudpickle.DEFAULT\_PROTOCOL*)

Saves component at file path

**Parameters**

- **file\_path** (*str*) – Location to save file
- **pickle\_protocol** (*int*) – The pickle data stream format.

**Returns** `None`

**transform** (*self*, *X*, *y=None*)

Transforms data *X*.

**Parameters**

- **X** (*pd.DataFrame*) – Data to transform.
- **y** (*pd.Series*, *optional*) – Target data.

**Returns** Transformed *X*

**Return type** `pd.DataFrame`

## feature\_selection

### Submodules

### feature\_selector

### Module Contents

### Classes Summary

---

<i>FeatureSelector</i>
------------------------

Selects top features based on importance weights.
---

---



## Contents

**class** evalml.pipelines.components.transformers.feature\_selection.feature\_selector.**FeatureS**

Selects top features based on importance weights.

### Parameters

- **parameters** (*dict*) – Dictionary of parameters for the component. Defaults to None.
- **component\_obj** (*obj*) – Third-party objects useful in component implementation. Defaults to None.
- **random\_seed** (*int*) – Seed for the random number generator. Defaults to 0.

### Attributes

<b>model_family</b>	ModelFamily.NONE
<b>modifies_features</b>	True
<b>modifies_target</b>	False

### Methods

<i>clone</i>	Constructs a new component with the same parameters and random state.
<i>default_parameters</i>	Returns the default parameters for this component.
<i>describe</i>	Describe a component and its parameters
<i>fit</i>	Fits component to data
<i>fit_transform</i>	Fits on X and transforms X
<i>get_names</i>	Get names of selected features.
<i>load</i>	Loads component at file path
<i>name</i>	Returns string name of this component
<i>needs_fitting</i>	Returns boolean determining if component needs fitting before
<i>parameters</i>	Returns the parameters which were used to initialize the component
<i>save</i>	Saves component at file path
<i>transform</i>	Transforms input data by selecting features. If the component_obj does not have a transform method, will raise an MethodPropertyNotFoundError exception.

**clone** (*self*)

Constructs a new component with the same parameters and random state.

**Returns** A new instance of this component with identical parameters and random state.

**default\_parameters** (*cls*)

Returns the default parameters for this component.

Our convention is that `Component.default_parameters == Component().parameters`.

**Returns** default parameters for this component.

**Return type** dict

**describe** (*self*, *print\_name=False*, *return\_dict=False*)

Describe a component and its parameters

**Parameters**

- **print\_name** (*bool*, *optional*) – whether to print name of component
- **return\_dict** (*bool*, *optional*) – whether to return description as dictionary in the format {"name": name, "parameters": parameters}

**Returns** prints and returns dictionary

**Return type** None or dict

**fit** (*self*, *X*, *y=None*)

Fits component to data

**Parameters**

- **X** (*list*, *pd.DataFrame* or *np.ndarray*) – The input training data of shape [n\_samples, n\_features]
- **y** (*list*, *pd.Series*, *np.ndarray*, *optional*) – The target training data of length [n\_samples]

**Returns** self

**fit\_transform** (*self*, *X*, *y=None*)

Fits on X and transforms X

**Parameters**

- **X** (*pd.DataFrame*) – Data to fit and transform
- **y** (*pd.Series*) – Target data

**Returns** Transformed X

**Return type** pd.DataFrame

**get\_names** (*self*)

Get names of selected features.

**Returns** List of the names of features selected

**Return type** list[str]

**static load** (*file\_path*)

Loads component at file path

**Parameters** **file\_path** (*str*) – Location to load file

**Returns** ComponentBase object

**property name** (*cls*)

Returns string name of this component

**needs\_fitting** (*self*)

Returns boolean determining if component needs fitting before calling predict, predict\_proba, transform, or feature\_importances. This can be overridden to False for components that do not need to be fit or whose fit methods do nothing.

**property parameters** (*self*)

Returns the parameters which were used to initialize the component

**save** (*self*, *file\_path*, *pickle\_protocol*=*cloudpickle.DEFAULT\_PROTOCOL*)

Saves component at file path

#### Parameters

- **file\_path** (*str*) – Location to save file
- **pickle\_protocol** (*int*) – The pickle data stream format.

**Returns** None

**transform** (*self*, *X*, *y*=*None*)

Transforms input data by selecting features. If the component\_obj does not have a transform method, will raise an `MethodPropertyNotFoundError` exception.

#### Parameters

- **X** (*pd.DataFrame*) – Data to transform.
- **y** (*pd.Series*, *optional*) – Target data. Ignored.

**Returns** Transformed X

**Return type** `pd.DataFrame`

## rf\_classifier\_feature\_selector

### Module Contents

### Classes Summary

---

*RFClassifierSelectFromModel*

Selects top features based on importance weights using a Random Forest classifier.

---

### Contents

**class** `evalml.pipelines.components.transformers.feature_selection.rf_classifier_feature_sel`

Selects top features based on importance weights using a Random Forest classifier.

#### Parameters

- **number\_features** (*int*) – The maximum number of features to select. If both `percent_features` and `number_features` are specified, take the greater number of features. Defaults to 0.5. Defaults to None.
- **n\_estimators** (*float*) – The number of trees in the forest. Defaults to 100.
- **max\_depth** (*int*) – Maximum tree depth for base learners. Defaults to 6.
- **percent\_features** (*float*) – Percentage of features to use. If both `percent_features` and `number_features` are specified, take the greater number of features. Defaults to 0.5.
- **threshold** (*string or float*) – The threshold value to use for feature selection. Features whose importance is greater or equal are kept while the others are discarded. If “median”, then the threshold value is the median of the feature importances. A scaling factor (e.g., “1.25\*mean”) may also be used. Defaults to `-np.inf`.
- **n\_jobs** (*int or None*) – Number of jobs to run in parallel. -1 uses all processes. Defaults to -1.
- **random\_seed** (*int*) – Seed for the random number generator. Defaults to 0.

### Attributes

<b>hyper-parameter_ranges</b>	{ “percent_features”: Real(0.01, 1), “threshold”: [“mean”, -np.inf], }
<b>model_family</b>	ModelFamily.NONE
<b>modifies_features</b>	True
<b>modifies_target</b>	False
<b>name</b>	RF Classifier Select From Model

### Methods

<i>clone</i>	Constructs a new component with the same parameters and random state.
<i>default_parameters</i>	Returns the default parameters for this component.
<i>describe</i>	Describe a component and its parameters
<i>fit</i>	Fits component to data
<i>fit_transform</i>	Fits on X and transforms X
<i>get_names</i>	Get names of selected features.
<i>load</i>	Loads component at file path
<i>needs_fitting</i>	Returns boolean determining if component needs fitting before
<i>parameters</i>	Returns the parameters which were used to initialize the component
<i>save</i>	Saves component at file path
<i>transform</i>	Transforms input data by selecting features. If the <code>component_obj</code> does not have a <code>transform</code> method, will raise an <code>MethodPropertyNotFoundError</code> exception.

### **clone** (*self*)

Constructs a new component with the same parameters and random state.

**Returns** A new instance of this component with identical parameters and random state.

**default\_parameters** (*cls*)

Returns the default parameters for this component.

Our convention is that `Component.default_parameters == Component().parameters`.

**Returns** default parameters for this component.

**Return type** dict

**describe** (*self*, *print\_name=False*, *return\_dict=False*)

Describe a component and its parameters

**Parameters**

- **print\_name** (*bool*, *optional*) – whether to print name of component
- **return\_dict** (*bool*, *optional*) – whether to return description as dictionary in the format {"name": name, "parameters": parameters}

**Returns** prints and returns dictionary

**Return type** None or dict

**fit** (*self*, *X*, *y=None*)

Fits component to data

**Parameters**

- **X** (*list*, *pd.DataFrame* or *np.ndarray*) – The input training data of shape [n\_samples, n\_features]
- **y** (*list*, *pd.Series*, *np.ndarray*, *optional*) – The target training data of length [n\_samples]

**Returns** self

**fit\_transform** (*self*, *X*, *y=None*)

Fits on X and transforms X

**Parameters**

- **X** (*pd.DataFrame*) – Data to fit and transform
- **y** (*pd.Series*) – Target data

**Returns** Transformed X

**Return type** pd.DataFrame

**get\_names** (*self*)

Get names of selected features.

**Returns** List of the names of features selected

**Return type** list[str]

**static load** (*file\_path*)

Loads component at file path

**Parameters** **file\_path** (*str*) – Location to load file

**Returns** ComponentBase object

**needs\_fitting** (*self*)

Returns boolean determining if component needs fitting before calling predict, predict\_proba, transform,

or `feature_importances`. This can be overridden to `False` for components that do not need to be fit or whose fit methods do nothing.

**property** `parameters` (*self*)

Returns the parameters which were used to initialize the component

**save** (*self*, *file\_path*, *pickle\_protocol=cloudpickle.DEFAULT\_PROTOCOL*)

Saves component at file path

**Parameters**

- **file\_path** (*str*) – Location to save file
- **pickle\_protocol** (*int*) – The pickle data stream format.

**Returns** `None`

**transform** (*self*, *X*, *y=None*)

Transforms input data by selecting features. If the `component_obj` does not have a transform method, will raise an `MethodPropertyNotFoundError` exception.

**Parameters**

- **X** (*pd.DataFrame*) – Data to transform.
- **y** (*pd.Series*, *optional*) – Target data. Ignored.

**Returns** Transformed X

**Return type** `pd.DataFrame`

## rf\_regressor\_feature\_selector

### Module Contents

### Classes Summary

---

*RFRegressorSelectFromModel*

Selects top features based on importance weights using a Random Forest regressor.

---

## Contents

**class** evalml.pipelines.components.transformers.feature\_selection.rf\_regressor\_feature\_select

Selects top features based on importance weights using a Random Forest regressor.

### Parameters

- **number\_features** (*int*) – The maximum number of features to select. If both percent\_features and number\_features are specified, take the greater number of features. Defaults to 0.5. Defaults to None.
- **n\_estimators** (*float*) – The number of trees in the forest. Defaults to 100.
- **max\_depth** (*int*) – Maximum tree depth for base learners. Defaults to 6.
- **percent\_features** (*float*) – Percentage of features to use. If both percent\_features and number\_features are specified, take the greater number of features. Defaults to 0.5.
- **threshold** (*string or float*) – The threshold value to use for feature selection. Features whose importance is greater or equal are kept while the others are discarded. If “median”, then the threshold value is the median of the feature importances. A scaling factor (e.g., “1.25\*mean”) may also be used. Defaults to -np.inf.
- **n\_jobs** (*int or None*) – Number of jobs to run in parallel. -1 uses all processes. Defaults to -1.
- **random\_seed** (*int*) – Seed for the random number generator. Defaults to 0.

### Attributes

<b>hyper-parameter_ranges</b>	{ “percent_features”: Real(0.01, 1), “threshold”: [“mean”, -np.inf], }
<b>model_family</b>	ModelFamily.NONE
<b>modifies_features</b>	True
<b>modifies_target</b>	False
<b>name</b>	RF Regressor Select From Model

### Methods

<code>clone</code>	Constructs a new component with the same parameters and random state.
<code>default_parameters</code>	Returns the default parameters for this component.
<code>describe</code>	Describe a component and its parameters
<code>fit</code>	Fits component to data
<code>fit_transform</code>	Fits on X and transforms X
<code>get_names</code>	Get names of selected features.
<code>load</code>	Loads component at file path
<code>needs_fitting</code>	Returns boolean determining if component needs fitting before
<code>parameters</code>	Returns the parameters which were used to initialize the component
<code>save</code>	Saves component at file path
<code>transform</code>	Transforms input data by selecting features. If the component_obj does not have a transform method, will raise an <code>MethodPropertyNotFoundError</code> exception.

**clone** (*self*)

Constructs a new component with the same parameters and random state.

**Returns** A new instance of this component with identical parameters and random state.

**default\_parameters** (*cls*)

Returns the default parameters for this component.

Our convention is that `Component.default_parameters == Component().parameters`.

**Returns** default parameters for this component.

**Return type** dict

**describe** (*self*, *print\_name=False*, *return\_dict=False*)

Describe a component and its parameters

**Parameters**

- **print\_name** (*bool*, *optional*) – whether to print name of component
- **return\_dict** (*bool*, *optional*) – whether to return description as dictionary in the format {"name": name, "parameters": parameters}

**Returns** prints and returns dictionary

**Return type** None or dict

**fit** (*self*, *X*, *y=None*)

Fits component to data

**Parameters**

- **X** (*list*, *pd.DataFrame* or *np.ndarray*) – The input training data of shape [n\_samples, n\_features]
- **y** (*list*, *pd.Series*, *np.ndarray*, *optional*) – The target training data of length [n\_samples]

**Returns** self

**fit\_transform** (*self*, *X*, *y=None*)

Fits on X and transforms X



**Parameters**

- **x** (*pd.DataFrame*) – Data to fit and transform
- **y** (*pd.Series*) – Target data

**Returns** Transformed X**Return type** *pd.DataFrame***get\_names** (*self*)

Get names of selected features.

**Returns** List of the names of features selected**Return type** *list[str]***static load** (*file\_path*)

Loads component at file path

**Parameters** **file\_path** (*str*) – Location to load file**Returns** *ComponentBase* object**needs\_fitting** (*self*)Returns boolean determining if component needs fitting before calling *predict*, *predict\_proba*, *transform*, or *feature\_importances*. This can be overridden to *False* for components that do not need to be fit or whose fit methods do nothing.**property parameters** (*self*)

Returns the parameters which were used to initialize the component

**save** (*self, file\_path, pickle\_protocol=cloudpickle.DEFAULT\_PROTOCOL*)

Saves component at file path

**Parameters**

- **file\_path** (*str*) – Location to save file
- **pickle\_protocol** (*int*) – The pickle data stream format.

**Returns** *None***transform** (*self, X, y=None*)Transforms input data by selecting features. If the *component\_obj* does not have a *transform* method, will raise an *MethodPropertyNotFoundError* exception.**Parameters**

- **x** (*pd.DataFrame*) – Data to transform.
- **y** (*pd.Series, optional*) – Target data. Ignored.

**Returns** Transformed X**Return type** *pd.DataFrame*

## Package Contents

### Classes Summary

<i>FeatureSelector</i>	Selects top features based on importance weights.
<i>RFClassifierSelectFromModel</i>	Selects top features based on importance weights using a Random Forest classifier.
<i>RFRegressorSelectFromModel</i>	Selects top features based on importance weights using a Random Forest regressor.

### Contents

**class** evalml.pipelines.components.transformers.feature\_selection.**FeatureSelector** (*parameters=None, component\_obj=None, random\_seed=0, \*\*kwargs*)

Selects top features based on importance weights.

#### Parameters

- **parameters** (*dict*) – Dictionary of parameters for the component. Defaults to None.
- **component\_obj** (*obj*) – Third-party objects useful in component implementation. Defaults to None.
- **random\_seed** (*int*) – Seed for the random number generator. Defaults to 0.

#### Attributes

<b>model_family</b>	ModelFamily.NONE
<b>modifies_features</b>	True
<b>modifies_target</b>	False

#### Methods

<i>clone</i>	Constructs a new component with the same parameters and random state.
<i>default_parameters</i>	Returns the default parameters for this component.
<i>describe</i>	Describe a component and its parameters
<i>fit</i>	Fits component to data
<i>fit_transform</i>	Fits on X and transforms X
<i>get_names</i>	Get names of selected features.
<i>load</i>	Loads component at file path
<i>name</i>	Returns string name of this component
<i>needs_fitting</i>	Returns boolean determining if component needs fitting before

continues on next page

Table 424 – continued from previous page

<i>parameters</i>	Returns the parameters which were used to initialize the component
<i>save</i>	Saves component at file path
<i>transform</i>	Transforms input data by selecting features. If the component_obj does not have a transform method, will raise an <code>MethodPropertyNotFoundError</code> exception.

**clone** (*self*)

Constructs a new component with the same parameters and random state.

**Returns** A new instance of this component with identical parameters and random state.

**default\_parameters** (*cls*)

Returns the default parameters for this component.

Our convention is that `Component.default_parameters == Component().parameters`.

**Returns** default parameters for this component.

**Return type** dict

**describe** (*self*, *print\_name=False*, *return\_dict=False*)

Describe a component and its parameters

**Parameters**

- **print\_name** (*bool*, *optional*) – whether to print name of component
- **return\_dict** (*bool*, *optional*) – whether to return description as dictionary in the format {"name": name, "parameters": parameters}

**Returns** prints and returns dictionary

**Return type** None or dict

**fit** (*self*, *X*, *y=None*)

Fits component to data

**Parameters**

- **X** (*list*, *pd.DataFrame* or *np.ndarray*) – The input training data of shape [n\_samples, n\_features]
- **y** (*list*, *pd.Series*, *np.ndarray*, *optional*) – The target training data of length [n\_samples]

**Returns** self

**fit\_transform** (*self*, *X*, *y=None*)

Fits on X and transforms X

**Parameters**

- **X** (*pd.DataFrame*) – Data to fit and transform
- **y** (*pd.Series*) – Target data

**Returns** Transformed X

**Return type** pd.DataFrame

**get\_names** (*self*)

Get names of selected features.

**Returns** List of the names of features selected

**Return type** list[str]

**static load** (*file\_path*)

Loads component at file path

**Parameters** **file\_path** (*str*) – Location to load file

**Returns** ComponentBase object

**property name** (*cls*)

Returns string name of this component

**needs\_fitting** (*self*)

Returns boolean determining if component needs fitting before calling predict, predict\_proba, transform, or feature\_importances. This can be overridden to False for components that do not need to be fit or whose fit methods do nothing.

**property parameters** (*self*)

Returns the parameters which were used to initialize the component

**save** (*self*, *file\_path*, *pickle\_protocol=cloudpickle.DEFAULT\_PROTOCOL*)

Saves component at file path

**Parameters**

- **file\_path** (*str*) – Location to save file
- **pickle\_protocol** (*int*) – The pickle data stream format.

**Returns** None

**transform** (*self*, *X*, *y=None*)

Transforms input data by selecting features. If the component\_obj does not have a transform method, will raise an MethodPropertyNotFoundError exception.

**Parameters**

- **X** (*pd.DataFrame*) – Data to transform.
- **y** (*pd.Series*, *optional*) – Target data. Ignored.

**Returns** Transformed X

**Return type** pd.DataFrame

**class** evalml.pipelines.components.transformers.feature\_selection.RFClassifierSelectFromModel

Selects top features based on importance weights using a Random Forest classifier.

**Parameters**

- **number\_features** (*int*) – The maximum number of features to select. If both `percent_features` and `number_features` are specified, take the greater number of features. Defaults to 0.5. Defaults to None.
- **n\_estimators** (*float*) – The number of trees in the forest. Defaults to 100.
- **max\_depth** (*int*) – Maximum tree depth for base learners. Defaults to 6.
- **percent\_features** (*float*) – Percentage of features to use. If both `percent_features` and `number_features` are specified, take the greater number of features. Defaults to 0.5.
- **threshold** (*string or float*) – The threshold value to use for feature selection. Features whose importance is greater or equal are kept while the others are discarded. If “median”, then the threshold value is the median of the feature importances. A scaling factor (e.g., “1.25\*mean”) may also be used. Defaults to `-np.inf`.
- **n\_jobs** (*int or None*) – Number of jobs to run in parallel. -1 uses all processes. Defaults to -1.
- **random\_seed** (*int*) – Seed for the random number generator. Defaults to 0.

#### Attributes

<b>hyper-parameter_ranges</b>	{ “percent_features”: Real(0.01, 1), “threshold”: [“mean”, -np.inf], }
<b>model_family</b>	ModelFamily.NONE
<b>modifies_features</b>	True
<b>modifies_target</b>	False
<b>name</b>	RF Classifier Select From Model

#### Methods

<i>clone</i>	Constructs a new component with the same parameters and random state.
<i>default_parameters</i>	Returns the default parameters for this component.
<i>describe</i>	Describe a component and its parameters
<i>fit</i>	Fits component to data
<i>fit_transform</i>	Fits on X and transforms X
<i>get_names</i>	Get names of selected features.
<i>load</i>	Loads component at file path
<i>needs_fitting</i>	Returns boolean determining if component needs fitting before
<i>parameters</i>	Returns the parameters which were used to initialize the component
<i>save</i>	Saves component at file path
<i>transform</i>	Transforms input data by selecting features. If the <code>component_obj</code> does not have a <code>transform</code> method, will raise an <code>MethodPropertyNotFoundError</code> exception.

#### **clone** (*self*)

Constructs a new component with the same parameters and random state.

**Returns** A new instance of this component with identical parameters and random state.

**default\_parameters** (*cls*)

Returns the default parameters for this component.

Our convention is that `Component.default_parameters == Component().parameters`.

**Returns** default parameters for this component.

**Return type** dict

**describe** (*self*, *print\_name=False*, *return\_dict=False*)

Describe a component and its parameters

**Parameters**

- **print\_name** (*bool*, *optional*) – whether to print name of component
- **return\_dict** (*bool*, *optional*) – whether to return description as dictionary in the format {"name": name, "parameters": parameters}

**Returns** prints and returns dictionary

**Return type** None or dict

**fit** (*self*, *X*, *y=None*)

Fits component to data

**Parameters**

- **X** (*list*, *pd.DataFrame* or *np.ndarray*) – The input training data of shape [n\_samples, n\_features]
- **y** (*list*, *pd.Series*, *np.ndarray*, *optional*) – The target training data of length [n\_samples]

**Returns** self

**fit\_transform** (*self*, *X*, *y=None*)

Fits on X and transforms X

**Parameters**

- **X** (*pd.DataFrame*) – Data to fit and transform
- **y** (*pd.Series*) – Target data

**Returns** Transformed X

**Return type** pd.DataFrame

**get\_names** (*self*)

Get names of selected features.

**Returns** List of the names of features selected

**Return type** list[str]

**static load** (*file\_path*)

Loads component at file path

**Parameters** **file\_path** (*str*) – Location to load file

**Returns** ComponentBase object

**needs\_fitting** (*self*)

Returns boolean determining if component needs fitting before calling predict, predict\_proba, transform,

or `feature_importances`. This can be overridden to `False` for components that do not need to be fit or whose fit methods do nothing.

**property** `parameters` (*self*)

Returns the parameters which were used to initialize the component

**save** (*self*, *file\_path*, *pickle\_protocol*=`cloudpickle.DEFAULT_PROTOCOL`)

Saves component at file path

#### Parameters

- **file\_path** (*str*) – Location to save file
- **pickle\_protocol** (*int*) – The pickle data stream format.

**Returns** `None`

**transform** (*self*, *X*, *y*=`None`)

Transforms input data by selecting features. If the component\_obj does not have a transform method, will raise an `MethodPropertyNotFoundError` exception.

#### Parameters

- **X** (*pd.DataFrame*) – Data to transform.
- **y** (*pd.Series*, *optional*) – Target data. Ignored.

**Returns** Transformed X

**Return type** `pd.DataFrame`

**class** `evalml.pipelines.components.transformers.feature_selection.RFRegressorSelectFromModel`

Selects top features based on importance weights using a Random Forest regressor.

#### Parameters

- **number\_features** (*int*) – The maximum number of features to select. If both `percent_features` and `number_features` are specified, take the greater number of features. Defaults to 0.5. Defaults to `None`.
- **n\_estimators** (*float*) – The number of trees in the forest. Defaults to 100.
- **max\_depth** (*int*) – Maximum tree depth for base learners. Defaults to 6.
- **percent\_features** (*float*) – Percentage of features to use. If both `percent_features` and `number_features` are specified, take the greater number of features. Defaults to 0.5.
- **threshold** (*string or float*) – The threshold value to use for feature selection. Features whose importance is greater or equal are kept while the others are discarded. If “median”, then the threshold value is the median of the feature importances. A scaling factor (e.g., “1.25\*mean”) may also be used. Defaults to `-np.inf`.

- **n\_jobs** (*int* or *None*) – Number of jobs to run in parallel. -1 uses all processes. Defaults to -1.
- **random\_seed** (*int*) – Seed for the random number generator. Defaults to 0.

#### Attributes

<b>hyper-parameter_ranges</b>	{ “percent_features”: Real(0.01, 1), “threshold”: [“mean”, -np.inf], }
<b>model_family</b>	ModelFamily.NONE
<b>modifies_features</b>	True
<b>modifies_target</b>	False
<b>name</b>	RF Regressor Select From Model

#### Methods

<i>clone</i>	Constructs a new component with the same parameters and random state.
<i>default_parameters</i>	Returns the default parameters for this component.
<i>describe</i>	Describe a component and its parameters
<i>fit</i>	Fits component to data
<i>fit_transform</i>	Fits on X and transforms X
<i>get_names</i>	Get names of selected features.
<i>load</i>	Loads component at file path
<i>needs_fitting</i>	Returns boolean determining if component needs fitting before
<i>parameters</i>	Returns the parameters which were used to initialize the component
<i>save</i>	Saves component at file path
<i>transform</i>	Transforms input data by selecting features. If the component_obj does not have a transform method, will raise an MethodPropertyNotFoundError exception.

#### **clone** (*self*)

Constructs a new component with the same parameters and random state.

**Returns** A new instance of this component with identical parameters and random state.

#### **default\_parameters** (*cls*)

Returns the default parameters for this component.

Our convention is that `Component.default_parameters == Component().parameters`.

**Returns** default parameters for this component.

**Return type** dict

#### **describe** (*self*, *print\_name=False*, *return\_dict=False*)

Describe a component and its parameters

#### Parameters

- **print\_name** (*bool*, *optional*) – whether to print name of component



- **return\_dict** (*bool, optional*) – whether to return description as dictionary in the format {"name": name, "parameters": parameters}

**Returns** prints and returns dictionary

**Return type** None or dict

**fit** (*self, X, y=None*)

Fits component to data

**Parameters**

- **X** (*list, pd.DataFrame or np.ndarray*) – The input training data of shape [n\_samples, n\_features]
- **y** (*list, pd.Series, np.ndarray, optional*) – The target training data of length [n\_samples]

**Returns** self

**fit\_transform** (*self, X, y=None*)

Fits on X and transforms X

**Parameters**

- **X** (*pd.DataFrame*) – Data to fit and transform
- **y** (*pd.Series*) – Target data

**Returns** Transformed X

**Return type** pd.DataFrame

**get\_names** (*self*)

Get names of selected features.

**Returns** List of the names of features selected

**Return type** list[str]

**static load** (*file\_path*)

Loads component at file path

**Parameters** **file\_path** (*str*) – Location to load file

**Returns** ComponentBase object

**needs\_fitting** (*self*)

Returns boolean determining if component needs fitting before calling predict, predict\_proba, transform, or feature\_importances. This can be overridden to False for components that do not need to be fit or whose fit methods do nothing.

**property parameters** (*self*)

Returns the parameters which were used to initialize the component

**save** (*self, file\_path, pickle\_protocol=cloudpickle.DEFAULT\_PROTOCOL*)

Saves component at file path

**Parameters**

- **file\_path** (*str*) – Location to save file
- **pickle\_protocol** (*int*) – The pickle data stream format.

**Returns** None

**transform** (*self*, *X*, *y=None*)

Transforms input data by selecting features. If the component\_obj does not have a transform method, will raise an `MethodPropertyNotFoundError` exception.

#### Parameters

- **x** (*pd.DataFrame*) – Data to transform.
- **y** (*pd.Series*, *optional*) – Target data. Ignored.

**Returns** Transformed X

**Return type** `pd.DataFrame`

## imputers

### Submodules

#### imputer

### Module Contents

### Classes Summary

<i>Imputer</i>	Imputes missing data according to a specified imputation strategy.
----------------	--

### Contents

**class** `evalml.pipelines.components.transformers.imputers.imputer.Imputer` (*categorical\_impute\_strategy*, *categorical\_fill\_value=None*, *numeric\_impute\_strategy='most\_frequent'*, *numeric\_fill\_value=None*, *random\_seed=0*, *\*\*kwargs*)

Imputes missing data according to a specified imputation strategy.

#### Parameters

- **categorical\_impute\_strategy** (*string*) – Impute strategy to use for string, object, boolean, categorical dtypes. Valid values include “most\_frequent” and “constant”.
- **numeric\_impute\_strategy** (*string*) – Impute strategy to use for numeric columns. Valid values include “mean”, “median”, “most\_frequent”, and “constant”.
- **categorical\_fill\_value** (*string*) – When `categorical_impute_strategy == “constant”`, `fill_value` is used to replace missing data. The default value of `None` will fill with the

string “missing\_value”.

- **numeric\_fill\_value** (*int*, *float*) – When `numeric_impute_strategy == “constant”`, `fill_value` is used to replace missing data. The default value of `None` will fill with 0.
- **random\_seed** (*int*) – Seed for the random number generator. Defaults to 0.

#### Attributes

<b>hyper-parameter_ranges</b>	{ “categorical_impute_strategy”: [“most_frequent”], “numeric_impute_strategy”: [“mean”, “median”, “most_frequent”], }
<b>model_family</b>	ModelFamily.NONE
<b>modifies_features</b>	True
<b>modifies_target</b>	False
<b>name</b>	Imputer

#### Methods

<i>clone</i>	Constructs a new component with the same parameters and random state.
<i>default_parameters</i>	Returns the default parameters for this component.
<i>describe</i>	Describe a component and its parameters
<i>fit</i>	Fits imputer to data. ‘None’ values are converted to <code>np.nan</code> before imputation and are
<i>fit_transform</i>	Fits on X and transforms X
<i>load</i>	Loads component at file path
<i>needs_fitting</i>	Returns boolean determining if component needs fitting before
<i>parameters</i>	Returns the parameters which were used to initialize the component
<i>save</i>	Saves component at file path
<i>transform</i>	Transforms data X by imputing missing values. ‘None’ values are converted to <code>np.nan</code> before imputation and are

**clone** (*self*)

Constructs a new component with the same parameters and random state.

**Returns** A new instance of this component with identical parameters and random state.

**default\_parameters** (*cls*)

Returns the default parameters for this component.

Our convention is that `Component.default_parameters == Component().parameters`.

**Returns** default parameters for this component.

**Return type** dict

**describe** (*self*, *print\_name=False*, *return\_dict=False*)

Describe a component and its parameters

**Parameters**

- **print\_name** (*bool, optional*) – whether to print name of component
- **return\_dict** (*bool, optional*) – whether to return description as dictionary in the format {"name": name, "parameters": parameters}

**Returns** prints and returns dictionary

**Return type** None or dict

**fit** (*self, X, y=None*)

**Fits imputer to data. 'None' values are converted to np.nan before imputation and are** treated as the same.

**Parameters**

- **X** (*pd.DataFrame, np.ndarray*) – The input training data of shape [n\_samples, n\_features]
- **y** (*pd.Series, optional*) – The target training data of length [n\_samples]

**Returns** self

**fit\_transform** (*self, X, y=None*)

Fits on X and transforms X

**Parameters**

- **X** (*pd.DataFrame*) – Data to fit and transform
- **y** (*pd.Series*) – Target data

**Returns** Transformed X

**Return type** pd.DataFrame

**static load** (*file\_path*)

Loads component at file path

**Parameters** **file\_path** (*str*) – Location to load file

**Returns** ComponentBase object

**needs\_fitting** (*self*)

Returns boolean determining if component needs fitting before calling predict, predict\_proba, transform, or feature\_importances. This can be overridden to False for components that do not need to be fit or whose fit methods do nothing.

**property parameters** (*self*)

Returns the parameters which were used to initialize the component

**save** (*self, file\_path, pickle\_protocol=cloudpickle.DEFAULT\_PROTOCOL*)

Saves component at file path

**Parameters**

- **file\_path** (*str*) – Location to save file
- **pickle\_protocol** (*int*) – The pickle data stream format.

**Returns** None

**transform** (*self, X, y=None*)

**Transforms data X by imputing missing values. 'None' values are converted to np.nan before imputation and are** treated as the same.

**Parameters**

- **x** (*pd.DataFrame*) – Data to transform
- **y** (*pd.Series*, *optional*) – Ignored.

**Returns** Transformed X**Return type** *pd.DataFrame***per\_column\_imputer****Module Contents****Classes Summary***PerColumnImputer*

Imputes missing data according to a specified imputation strategy per column.

**Contents****class** evalml.pipelines.components.transformers.imputers.per\_column\_imputer.**PerColumnImputer**

Imputes missing data according to a specified imputation strategy per column.

**Parameters**

- **impute\_strategies** (*dict*) – Column and {"impute\_strategy": strategy, "fill\_value":value} pairings. Valid values for impute strategy include "mean", "median", "most\_frequent", "constant" for numerical data, and "most\_frequent", "constant" for object data types. Defaults to None, which uses "most\_frequent" for all columns. When impute\_strategy == "constant", fill\_value is used to replace missing data. When None, uses 0 when imputing numerical data and "missing\_value" for strings or object data types.
- **default\_impute\_strategy** (*str*) – Impute strategy to fall back on when none is provided for a certain column. Valid values include "mean", "median", "most\_frequent", "constant" for numerical data, and "most\_frequent", "constant" for object data types. Defaults to "most\_frequent".
- **random\_seed** (*int*) – Seed for the random number generator. Defaults to 0.

**Attributes**

<b>hyper-parameter_ranges</b>	{}
<b>model_family</b>	ModelFamily.NONE
<b>modifies_features</b>	True
<b>modifies_target</b>	False
<b>name</b>	Per Column Imputer

## Methods

<i>clone</i>	Constructs a new component with the same parameters and random state.
<i>default_parameters</i>	Returns the default parameters for this component.
<i>describe</i>	Describe a component and its parameters
<i>fit</i>	Fits imputers on input data
<i>fit_transform</i>	Fits on X and transforms X
<i>load</i>	Loads component at file path
<i>needs_fitting</i>	Returns boolean determining if component needs fitting before
<i>parameters</i>	Returns the parameters which were used to initialize the component
<i>save</i>	Saves component at file path
<i>transform</i>	Transforms input data by imputing missing values.

### **clone** (*self*)

Constructs a new component with the same parameters and random state.

**Returns** A new instance of this component with identical parameters and random state.

### **default\_parameters** (*cls*)

Returns the default parameters for this component.

Our convention is that `Component.default_parameters == Component().parameters`.

**Returns** default parameters for this component.

**Return type** dict

### **describe** (*self*, *print\_name=False*, *return\_dict=False*)

Describe a component and its parameters

#### Parameters

- **print\_name** (*bool*, *optional*) – whether to print name of component
- **return\_dict** (*bool*, *optional*) – whether to return description as dictionary in the format {"name": name, "parameters": parameters}

**Returns** prints and returns dictionary

**Return type** None or dict

### **fit** (*self*, *X*, *y=None*)

Fits imputers on input data

#### Parameters

- **X** (*pd.DataFrame* or *np.ndarray*) – The input training data of shape [n\_samples, n\_features] to fit.
- **y** (*pd.Series*, *optional*) – The target training data of length [n\_samples]. Ignored.

**Returns** *self*

**fit\_transform** (*self*, *X*, *y=None*)

Fits on X and transforms X

**Parameters**

- **X** (*pd.DataFrame*) – Data to fit and transform
- **y** (*pd.Series*) – Target data

**Returns** Transformed X

**Return type** *pd.DataFrame*

**static load** (*file\_path*)

Loads component at file path

**Parameters** **file\_path** (*str*) – Location to load file

**Returns** *ComponentBase* object

**needs\_fitting** (*self*)

Returns boolean determining if component needs fitting before calling predict, predict\_proba, transform, or feature\_importances. This can be overridden to False for components that do not need to be fit or whose fit methods do nothing.

**property parameters** (*self*)

Returns the parameters which were used to initialize the component

**save** (*self*, *file\_path*, *pickle\_protocol=cloudpickle.DEFAULT\_PROTOCOL*)

Saves component at file path

**Parameters**

- **file\_path** (*str*) – Location to save file
- **pickle\_protocol** (*int*) – The pickle data stream format.

**Returns** *None*

**transform** (*self*, *X*, *y=None*)

Transforms input data by imputing missing values.

**Parameters**

- **X** (*pd.DataFrame* or *np.ndarray*) – The input training data of shape [n\_samples, n\_features] to transform.
- **y** (*pd.Series*, *optional*) – The target training data of length [n\_samples]. Ignored.

**Returns** Transformed X

**Return type** *pd.DataFrame*

## simple\_imputer

## Module Contents

## Classes Summary

<i>SimpleImputer</i>	Imputes missing data according to a specified imputation strategy.
----------------------	--

## Contents

```
class evalml.pipelines.components.transformers.imputers.simple_imputer.SimpleImputer(impute_value=fill_value,
                                           random_state=random_state,
                                           domain=domain,
                                           seed=seed,
                                           **kwargs)
```

Imputes missing data according to a specified imputation strategy.

## Parameters

- **impute\_strategy** (*string*) – Impute strategy to use. Valid values include “mean”, “median”, “most\_frequent”, “constant” for numerical data, and “most\_frequent”, “constant” for object data types.
- **fill\_value** (*string*) – When `impute_strategy == “constant”`, `fill_value` is used to replace missing data. Defaults to 0 when imputing numerical data and “missing\_value” for strings or object data types.
- **random\_seed** (*int*) – Seed for the random number generator. Defaults to 0.

## Attributes

<b>hyper-parameter_ranges</b>	{ "impute_strategy": ["mean", "median", "most_frequent"] }
<b>model_family</b>	ModelFamily.NONE
<b>modifies_features</b>	True
<b>modifies_target</b>	False
<b>name</b>	Simple Imputer

## Methods

<i>clone</i>	Constructs a new component with the same parameters and random state.
<i>default_parameters</i>	Returns the default parameters for this component.
<i>describe</i>	Describe a component and its parameters
<i>fit</i>	Fits imputer to data. ‘None’ values are converted to np.nan before imputation and are
<i>fit_transform</i>	Fits on X and transforms X

continues on next page



Table 432 – continued from previous page

<i>load</i>	Loads component at file path
<i>needs_fitting</i>	Returns boolean determining if component needs fitting before
<i>parameters</i>	Returns the parameters which were used to initialize the component
<i>save</i>	Saves component at file path
<i>transform</i>	Transforms input by imputing missing values. ‘None’ and np.nan values are treated as the same.

**clone** (*self*)

Constructs a new component with the same parameters and random state.

**Returns** A new instance of this component with identical parameters and random state.

**default\_parameters** (*cls*)

Returns the default parameters for this component.

Our convention is that `Component.default_parameters == Component().parameters`.

**Returns** default parameters for this component.

**Return type** dict

**describe** (*self*, *print\_name=False*, *return\_dict=False*)

Describe a component and its parameters

**Parameters**

- **print\_name** (*bool*, *optional*) – whether to print name of component
- **return\_dict** (*bool*, *optional*) – whether to return description as dictionary in the format {“name”: name, “parameters”: parameters}

**Returns** prints and returns dictionary

**Return type** None or dict

**fit** (*self*, *X*, *y=None*)

**Fits imputer to data. ‘None’ values are converted to np.nan before imputation and are** treated as the same.

**Parameters**

- **X** (*pd.DataFrame* or *np.ndarray*) – the input training data of shape [n\_samples, n\_features]
- **y** (*pd.Series*, *optional*) – the target training data of length [n\_samples]

**Returns** self

**fit\_transform** (*self*, *X*, *y=None*)

Fits on X and transforms X

**Parameters**

- **X** (*pd.DataFrame*) – Data to fit and transform
- **y** (*pd.Series*, *optional*) – Target data.

**Returns** Transformed X

**Return type** pd.DataFrame

**static load** (*file\_path*)

Loads component at file path

**Parameters** **file\_path** (*str*) – Location to load file

**Returns** ComponentBase object

**needs\_fitting** (*self*)

Returns boolean determining if component needs fitting before calling predict, predict\_proba, transform, or feature\_importances. This can be overridden to False for components that do not need to be fit or whose fit methods do nothing.

**property parameters** (*self*)

Returns the parameters which were used to initialize the component

**save** (*self*, *file\_path*, *pickle\_protocol*=*cloudpickle.DEFAULT\_PROTOCOL*)

Saves component at file path

**Parameters**

- **file\_path** (*str*) – Location to save file
- **pickle\_protocol** (*int*) – The pickle data stream format.

**Returns** None

**transform** (*self*, *X*, *y*=*None*)

Transforms input by imputing missing values. ‘None’ and np.nan values are treated as the same.

**Parameters**

- **X** (*pd.DataFrame*) – Data to transform
- **y** (*pd.Series*, *optional*) – Ignored.

**Returns** Transformed X

**Return type** pd.DataFrame

**target\_imputer**

**Module Contents**

**Classes Summary**

---

<i>TargetImputer</i>	Imputes missing target data according to a specified imputation strategy.
<i>TargetImputerMeta</i>	A version of the ComponentBaseMeta class which handles when input features is None

---

## Contents

**class** evalml.pipelines.components.transformers.imputers.target\_imputer.**TargetImputer** (*impute\_*  
*fill\_value*  
*ran-*  
*dom\_see*  
*\*\*kwargs*

Imputes missing target data according to a specified imputation strategy.

### Parameters

- **impute\_strategy** (*string*) – Impute strategy to use. Valid values include “mean”, “median”, “most\_frequent”, “constant” for numerical data, and “most\_frequent”, “constant” for object data types. Defaults to “most\_frequent”.
- **fill\_value** (*string*) – When impute\_strategy == “constant”, fill\_value is used to replace missing data. Defaults to None which uses 0 when imputing numerical data and “missing\_value” for strings or object data types.
- **random\_seed** (*int*) – Seed for the random number generator. Defaults to 0.

### Attributes

<b>hyper-parameter_ranges</b>	{ “impute_strategy”: [“mean”, “median”, “most_frequent”]}
<b>model_family</b>	ModelFamily.NONE
<b>modifies_features</b>	False
<b>modifies_target</b>	True
<b>name</b>	Target Imputer

### Methods

<i>clone</i>	Constructs a new component with the same parameters and random state.
<i>default_parameters</i>	Returns the default parameters for this component.
<i>describe</i>	Describe a component and its parameters
<i>fit</i>	Fits imputer to target data. ‘None’ values are converted to np.nan before imputation and are
<i>fit_transform</i>	Fits on and transforms the input target data.
<i>load</i>	Loads component at file path
<i>needs_fitting</i>	Returns boolean determining if component needs fitting before
<i>parameters</i>	Returns the parameters which were used to initialize the component
<i>save</i>	Saves component at file path
<i>transform</i>	Transforms input target data by imputing missing values. ‘None’ and np.nan values are treated as the same.

**clone** (*self*)

Constructs a new component with the same parameters and random state.

**Returns** A new instance of this component with identical parameters and random state.

**default\_parameters** (*cls*)

Returns the default parameters for this component.

Our convention is that `Component.default_parameters == Component().parameters`.

**Returns** default parameters for this component.

**Return type** dict

**describe** (*self*, *print\_name=False*, *return\_dict=False*)

Describe a component and its parameters

**Parameters**

- **print\_name** (*bool*, *optional*) – whether to print name of component
- **return\_dict** (*bool*, *optional*) – whether to return description as dictionary in the format {"name": name, "parameters": parameters}

**Returns** prints and returns dictionary

**Return type** None or dict

**fit** (*self*, *X*, *y*)

**Fits imputer to target data. ‘None’ values are converted to np.nan before imputation and are treated as the same.**

**Parameters**

- **X** (*pd.DataFrame* or *np.ndarray*) – The input training data of shape [n\_samples, n\_features]. Ignored.
- **y** (*pd.Series*, *optional*) – The target training data of length [n\_samples].

**Returns** self

**fit\_transform** (*self*, *X*, *y*)

Fits on and transforms the input target data.

**Parameters**

- **X** (*pd.DataFrame*) – Features. Ignored.
- **y** (*pd.Series*) – Target data to impute.

**Returns** The original X, transformed y

**Return type** (pd.DataFrame, pd.Series)

**static load** (*file\_path*)

Loads component at file path

**Parameters** **file\_path** (*str*) – Location to load file

**Returns** ComponentBase object

**needs\_fitting** (*self*)

Returns boolean determining if component needs fitting before calling predict, predict\_proba, transform, or feature\_importances. This can be overridden to False for components that do not need to be fit or whose fit methods do nothing.

**property parameters** (*self*)

Returns the parameters which were used to initialize the component

**save** (*self*, *file\_path*, *pickle\_protocol*=*cloudpickle.DEFAULT\_PROTOCOL*)

Saves component at file path

#### Parameters

- **file\_path** (*str*) – Location to save file
- **pickle\_protocol** (*int*) – The pickle data stream format.

**Returns** None

**transform** (*self*, *X*, *y*)

Transforms input target data by imputing missing values. ‘None’ and np.nan values are treated as the same.

#### Parameters

- **X** (*pd.DataFrame*) – Features. Ignored.
- **y** (*pd.Series*) – Target data to impute.

**Returns** The original X, transformed y

**Return type** (pd.DataFrame, pd.Series)

**class** evalml.pipelines.components.transformers.imputers.target\_imputer.**TargetImputerMeta**

A version of the ComponentBaseMeta class which handles when input features is None

#### Attributes

<b>FIT_METHODS</b>	['fit', 'fit_transform']
<b>METHODS_TO_CHECK</b>	['predict', 'predict_proba', 'transform', 'inverse_transform']
<b>PROPERTIES_TO_CHECK</b>	['feature_importance']

#### Methods

<i>check_for_fit</i>	<i>check_for_fit</i> wraps a method that validates if <i>self.is_fitted</i> is <i>True</i> .
<i>register</i>	Register a virtual subclass of an ABC.
<i>set_fit</i>	

**classmethod** **check\_for\_fit** (*cls*, *method*)

*check\_for\_fit* wraps a method that validates if *self.is\_fitted* is *True*. It raises an exception if *False* and calls and returns the wrapped method if *True*.

**register** (*cls*, *subclass*)

Register a virtual subclass of an ABC.

Returns the subclass, to allow usage as a class decorator.

**classmethod** **set\_fit** (*cls*, *method*)

## Package Contents

### Classes Summary

<i>Imputer</i>	Imputes missing data according to a specified imputation strategy.
<i>PerColumnImputer</i>	Imputes missing data according to a specified imputation strategy per column.
<i>SimpleImputer</i>	Imputes missing data according to a specified imputation strategy.
<i>TargetImputer</i>	Imputes missing target data according to a specified imputation strategy.

### Contents

**class** evalml.pipelines.components.transformers.imputers.**Imputer** (*categorical\_impute\_strategy='most\_frequent', categorical\_fill\_value=None, numeric\_impute\_strategy='mean', numeric\_fill\_value=None, random\_seed=0, \*\*kwargs*)

Imputes missing data according to a specified imputation strategy.

#### Parameters

- **categorical\_impute\_strategy** (*string*) – Impute strategy to use for string, object, boolean, categorical dtypes. Valid values include “most\_frequent” and “constant”.
- **numeric\_impute\_strategy** (*string*) – Impute strategy to use for numeric columns. Valid values include “mean”, “median”, “most\_frequent”, and “constant”.
- **categorical\_fill\_value** (*string*) – When categorical\_impute\_strategy == “constant”, fill\_value is used to replace missing data. The default value of None will fill with the string “missing\_value”.
- **numeric\_fill\_value** (*int, float*) – When numeric\_impute\_strategy == “constant”, fill\_value is used to replace missing data. The default value of None will fill with 0.
- **random\_seed** (*int*) – Seed for the random number generator. Defaults to 0.

#### Attributes

<b>hyper-parameter_ranges</b>	{ "categorical_impute_strategy": ["most_frequent"], "numeric_impute_strategy": ["mean", "median", "most_frequent"], }
<b>model_family</b>	ModelFamily.NONE
<b>modifies_features</b>	True
<b>modifies_target</b>	False
<b>name</b>	Imputer

## Methods

<i>clone</i>	Constructs a new component with the same parameters and random state.
<i>default_parameters</i>	Returns the default parameters for this component.
<i>describe</i>	Describe a component and its parameters
<i>fit</i>	Fits imputer to data. 'None' values are converted to np.nan before imputation and are
<i>fit_transform</i>	Fits on X and transforms X
<i>load</i>	Loads component at file path
<i>needs_fitting</i>	Returns boolean determining if component needs fitting before
<i>parameters</i>	Returns the parameters which were used to initialize the component
<i>save</i>	Saves component at file path
<i>transform</i>	Transforms data X by imputing missing values. 'None' values are converted to np.nan before imputation and are

### **clone** (*self*)

Constructs a new component with the same parameters and random state.

**Returns** A new instance of this component with identical parameters and random state.

### **default\_parameters** (*cls*)

Returns the default parameters for this component.

Our convention is that `Component.default_parameters == Component().parameters`.

**Returns** default parameters for this component.

**Return type** dict

### **describe** (*self*, *print\_name=False*, *return\_dict=False*)

Describe a component and its parameters

#### Parameters

- **print\_name** (*bool*, *optional*) – whether to print name of component
- **return\_dict** (*bool*, *optional*) – whether to return description as dictionary in the format {"name": name, "parameters": parameters}

**Returns** prints and returns dictionary

**Return type** None or dict

### **fit** (*self*, *X*, *y=None*)

**Fits imputer to data.** ‘None’ values are converted to np.nan before imputation and are treated as the same.

**Parameters**

- **X** (*pd.DataFrame*, *np.ndarray*) – The input training data of shape [n\_samples, n\_features]
- **y** (*pd.Series*, *optional*) – The target training data of length [n\_samples]

**Returns** self

**fit\_transform** (*self*, *X*, *y=None*)

Fits on X and transforms X

**Parameters**

- **X** (*pd.DataFrame*) – Data to fit and transform
- **y** (*pd.Series*) – Target data

**Returns** Transformed X

**Return type** pd.DataFrame

**static load** (*file\_path*)

Loads component at file path

**Parameters** **file\_path** (*str*) – Location to load file

**Returns** ComponentBase object

**needs\_fitting** (*self*)

Returns boolean determining if component needs fitting before calling predict, predict\_proba, transform, or feature\_importances. This can be overridden to False for components that do not need to be fit or whose fit methods do nothing.

**property parameters** (*self*)

Returns the parameters which were used to initialize the component

**save** (*self*, *file\_path*, *pickle\_protocol=cloudpickle.DEFAULT\_PROTOCOL*)

Saves component at file path

**Parameters**

- **file\_path** (*str*) – Location to save file
- **pickle\_protocol** (*int*) – The pickle data stream format.

**Returns** None

**transform** (*self*, *X*, *y=None*)

**Transforms data X by imputing missing values.** ‘None’ values are converted to np.nan before imputation and are treated as the same.

**Parameters**

- **X** (*pd.DataFrame*) – Data to transform
- **y** (*pd.Series*, *optional*) – Ignored.

**Returns** Transformed X

**Return type** pd.DataFrame



```
class evalml.pipelines.components.transformers.imputers.PerColumnImputer (impute_strategies=None,  

                                                                    de-  

                                                                    fault_impute_strategy='m  

                                                                    ran-  

                                                                    dom_seed=0,  

                                                                    **kwargs)
```

Imputes missing data according to a specified imputation strategy per column.

#### Parameters

- **impute\_strategies** (*dict*) – Column and {"impute\_strategy": strategy, "fill\_value":value} pairings. Valid values for impute strategy include "mean", "median", "most\_frequent", "constant" for numerical data, and "most\_frequent", "constant" for object data types. Defaults to None, which uses "most\_frequent" for all columns. When impute\_strategy == "constant", fill\_value is used to replace missing data. When None, uses 0 when imputing numerical data and "missing\_value" for strings or object data types.
- **default\_impute\_strategy** (*str*) – Impute strategy to fall back on when none is provided for a certain column. Valid values include "mean", "median", "most\_frequent", "constant" for numerical data, and "most\_frequent", "constant" for object data types. Defaults to "most\_frequent".
- **random\_seed** (*int*) – Seed for the random number generator. Defaults to 0.

#### Attributes

<b>hyper-parameter_ranges</b>	{}
<b>model_family</b>	ModelFamily.NONE
<b>modifies_features</b>	True
<b>modifies_target</b>	False
<b>name</b>	Per Column Imputer

#### Methods

<i>clone</i>	Constructs a new component with the same parameters and random state.
<i>default_parameters</i>	Returns the default parameters for this component.
<i>describe</i>	Describe a component and its parameters
<i>fit</i>	Fits imputers on input data
<i>fit_transform</i>	Fits on X and transforms X
<i>load</i>	Loads component at file path
<i>needs_fitting</i>	Returns boolean determining if component needs fitting before
<i>parameters</i>	Returns the parameters which were used to initialize the component
<i>save</i>	Saves component at file path
<i>transform</i>	Transforms input data by imputing missing values.

**clone** (*self*)

Constructs a new component with the same parameters and random state.

**Returns** A new instance of this component with identical parameters and random state.

**default\_parameters** (*cls*)

Returns the default parameters for this component.

Our convention is that `Component.default_parameters == Component().parameters`.

**Returns** default parameters for this component.

**Return type** dict

**describe** (*self*, *print\_name=False*, *return\_dict=False*)

Describe a component and its parameters

**Parameters**

- **print\_name** (*bool*, *optional*) – whether to print name of component
- **return\_dict** (*bool*, *optional*) – whether to return description as dictionary in the format {"name": name, "parameters": parameters}

**Returns** prints and returns dictionary

**Return type** None or dict

**fit** (*self*, *X*, *y=None*)

Fits imputers on input data

**Parameters**

- **X** (*pd.DataFrame* or *np.ndarray*) – The input training data of shape [n\_samples, n\_features] to fit.
- **y** (*pd.Series*, *optional*) – The target training data of length [n\_samples]. Ignored.

**Returns** self

**fit\_transform** (*self*, *X*, *y=None*)

Fits on X and transforms X

**Parameters**

- **X** (*pd.DataFrame*) – Data to fit and transform
- **y** (*pd.Series*) – Target data

**Returns** Transformed X

**Return type** pd.DataFrame

**static load** (*file\_path*)

Loads component at file path

**Parameters** **file\_path** (*str*) – Location to load file

**Returns** ComponentBase object

**needs\_fitting** (*self*)

Returns boolean determining if component needs fitting before calling predict, predict\_proba, transform, or feature\_importances. This can be overridden to False for components that do not need to be fit or whose fit methods do nothing.

**property parameters** (*self*)

Returns the parameters which were used to initialize the component

**save** (*self*, *file\_path*, *pickle\_protocol=cloudpickle.DEFAULT\_PROTOCOL*)

Saves component at file path

**Parameters**

- **file\_path** (*str*) – Location to save file
- **pickle\_protocol** (*int*) – The pickle data stream format.

**Returns** None

**transform** (*self*, *X*, *y=None*)

Transforms input data by imputing missing values.

**Parameters**

- **X** (*pd.DataFrame* or *np.ndarray*) – The input training data of shape [n\_samples, n\_features] to transform.
- **y** (*pd.Series*, *optional*) – The target training data of length [n\_samples]. Ignored.

**Returns** Transformed X

**Return type** *pd.DataFrame*

```
class evalml.pipelines.components.transformers.imputers.SimpleImputer (impute_strategy='most_frequent',  
                                                                    fill_value=None,  
                                                                    random_seed=0,  
                                                                    **kwargs)
```

Imputes missing data according to a specified imputation strategy.

**Parameters**

- **impute\_strategy** (*string*) – Impute strategy to use. Valid values include “mean”, “median”, “most\_frequent”, “constant” for numerical data, and “most\_frequent”, “constant” for object data types.
- **fill\_value** (*string*) – When *impute\_strategy* == “constant”, *fill\_value* is used to replace missing data. Defaults to 0 when imputing numerical data and “missing\_value” for strings or object data types.
- **random\_seed** (*int*) – Seed for the random number generator. Defaults to 0.

**Attributes**

<b>hyper-parameter_ranges</b>	{ “impute_strategy”: [“mean”, “median”, “most_frequent”]}
<b>model_family</b>	ModelFamily.NONE
<b>modifies_features</b>	True
<b>modifies_target</b>	False
<b>name</b>	Simple Imputer

**Methods**

<i>clone</i>	Constructs a new component with the same parameters and random state.
<i>default_parameters</i>	Returns the default parameters for this component.
<i>describe</i>	Describe a component and its parameters
<i>fit</i>	Fits imputer to data. ‘None’ values are converted to np.nan before imputation and are

continues on next page

Table 439 – continued from previous page

<code>fit_transform</code>	Fits on X and transforms X
<code>load</code>	Loads component at file path
<code>needs_fitting</code>	Returns boolean determining if component needs fitting before
<code>parameters</code>	Returns the parameters which were used to initialize the component
<code>save</code>	Saves component at file path
<code>transform</code>	Transforms input by imputing missing values. ‘None’ and np.nan values are treated as the same.

**clone** (*self*)

Constructs a new component with the same parameters and random state.

**Returns** A new instance of this component with identical parameters and random state.

**default\_parameters** (*cls*)

Returns the default parameters for this component.

Our convention is that `Component.default_parameters == Component().parameters`.

**Returns** default parameters for this component.

**Return type** dict

**describe** (*self*, *print\_name=False*, *return\_dict=False*)

Describe a component and its parameters

**Parameters**

- **print\_name** (*bool*, *optional*) – whether to print name of component
- **return\_dict** (*bool*, *optional*) – whether to return description as dictionary in the format {“name”: name, “parameters”: parameters}

**Returns** prints and returns dictionary

**Return type** None or dict

**fit** (*self*, *X*, *y=None*)

**Fits imputer to data. ‘None’ values are converted to np.nan before imputation and are** treated as the same.

**Parameters**

- **X** (*pd.DataFrame* or *np.ndarray*) – the input training data of shape [n\_samples, n\_features]
- **y** (*pd.Series*, *optional*) – the target training data of length [n\_samples]

**Returns** self

**fit\_transform** (*self*, *X*, *y=None*)

Fits on X and transforms X

**Parameters**

- **X** (*pd.DataFrame*) – Data to fit and transform
- **y** (*pd.Series*, *optional*) – Target data.

**Returns** Transformed X

**Return type** `pd.DataFrame`

**static load** (*file\_path*)

Loads component at file path

**Parameters** `file_path` (*str*) – Location to load file

**Returns** `ComponentBase` object

**needs\_fitting** (*self*)

Returns boolean determining if component needs fitting before calling `predict`, `predict_proba`, `transform`, or `feature_importances`. This can be overridden to `False` for components that do not need to be fit or whose fit methods do nothing.

**property parameters** (*self*)

Returns the parameters which were used to initialize the component

**save** (*self*, *file\_path*, *pickle\_protocol=cloudpickle.DEFAULT\_PROTOCOL*)

Saves component at file path

**Parameters**

- **file\_path** (*str*) – Location to save file
- **pickle\_protocol** (*int*) – The pickle data stream format.

**Returns** `None`

**transform** (*self*, *X*, *y=None*)

Transforms input by imputing missing values. ‘None’ and `np.nan` values are treated as the same.

**Parameters**

- **X** (*pd.DataFrame*) – Data to transform
- **y** (*pd.Series*, *optional*) – Ignored.

**Returns** Transformed `X`

**Return type** `pd.DataFrame`

```
class evalml.pipelines.components.transformers.imputers.TargetImputer (impute_strategy='most_frequent',
                                                                    fill_value=None,
                                                                    random_seed=0,
                                                                    **kwargs)
```

Imputes missing target data according to a specified imputation strategy.

**Parameters**

- **impute\_strategy** (*string*) – Impute strategy to use. Valid values include “mean”, “median”, “most\_frequent”, “constant” for numerical data, and “most\_frequent”, “constant” for object data types. Defaults to “most\_frequent”.
- **fill\_value** (*string*) – When `impute_strategy == “constant”`, `fill_value` is used to replace missing data. Defaults to `None` which uses 0 when imputing numerical data and “missing\_value” for strings or object data types.
- **random\_seed** (*int*) – Seed for the random number generator. Defaults to 0.

**Attributes**

<b>hyper-parameter_ranges</b>	{ "impute_strategy": ["mean", "median", "most_frequent"] }
<b>model_family</b>	ModelFamily.NONE
<b>modifies_features</b>	False
<b>modifies_target</b>	True
<b>name</b>	Target Imputer

## Methods

<i>clone</i>	Constructs a new component with the same parameters and random state.
<i>default_parameters</i>	Returns the default parameters for this component.
<i>describe</i>	Describe a component and its parameters
<i>fit</i>	Fits imputer to target data. 'None' values are converted to np.nan before imputation and are
<i>fit_transform</i>	Fits on and transforms the input target data.
<i>load</i>	Loads component at file path
<i>needs_fitting</i>	Returns boolean determining if component needs fitting before
<i>parameters</i>	Returns the parameters which were used to initialize the component
<i>save</i>	Saves component at file path
<i>transform</i>	Transforms input target data by imputing missing values. 'None' and np.nan values are treated as the same.

### **clone** (*self*)

Constructs a new component with the same parameters and random state.

**Returns** A new instance of this component with identical parameters and random state.

### **default\_parameters** (*cls*)

Returns the default parameters for this component.

Our convention is that `Component.default_parameters == Component().parameters`.

**Returns** default parameters for this component.

**Return type** dict

### **describe** (*self*, *print\_name=False*, *return\_dict=False*)

Describe a component and its parameters

#### Parameters

- **print\_name** (*bool*, *optional*) – whether to print name of component
- **return\_dict** (*bool*, *optional*) – whether to return description as dictionary in the format {"name": name, "parameters": parameters}

**Returns** prints and returns dictionary

**Return type** None or dict

### **fit** (*self*, *X*, *y*)

**Fits imputer to target data.** ‘None’ values are converted to `np.nan` before imputation and are treated as the same.

#### Parameters

- **X** (`pd.DataFrame` or `np.ndarray`) – The input training data of shape `[n_samples, n_features]`. Ignored.
- **y** (`pd.Series`, optional) – The target training data of length `[n_samples]`.

**Returns** `self`

**fit\_transform** (`self`, `X`, `y`)

Fits on and transforms the input target data.

#### Parameters

- **X** (`pd.DataFrame`) – Features. Ignored.
- **y** (`pd.Series`) – Target data to impute.

**Returns** The original X, transformed y

**Return type** (`pd.DataFrame`, `pd.Series`)

**static load** (`file_path`)

Loads component at file path

**Parameters** **file\_path** (`str`) – Location to load file

**Returns** `ComponentBase` object

**needs\_fitting** (`self`)

Returns boolean determining if component needs fitting before calling `predict`, `predict_proba`, `transform`, or `feature_importances`. This can be overridden to `False` for components that do not need to be fit or whose fit methods do nothing.

**property parameters** (`self`)

Returns the parameters which were used to initialize the component

**save** (`self`, `file_path`, `pickle_protocol=cloudpickle.DEFAULT_PROTOCOL`)

Saves component at file path

#### Parameters

- **file\_path** (`str`) – Location to save file
- **pickle\_protocol** (`int`) – The pickle data stream format.

**Returns** `None`

**transform** (`self`, `X`, `y`)

Transforms input target data by imputing missing values. ‘None’ and `np.nan` values are treated as the same.

#### Parameters

- **X** (`pd.DataFrame`) – Features. Ignored.
- **y** (`pd.Series`) – Target data to impute.

**Returns** The original X, transformed y

**Return type** (`pd.DataFrame`, `pd.Series`)

## preprocessing

### Submodules

#### datetime\_featurizer

### Module Contents

### Classes Summary

---

<code>DateTimeFeaturizer</code>	Transformer that can automatically extract features from datetime columns.
---------------------------------	--

---

### Contents

**class** evalml.pipelines.components.transformers.preprocessing.datetime\_featurizer.**DateTimeFeaturizer**

Transformer that can automatically extract features from datetime columns.

#### Parameters

- **features\_to\_extract** (*list*) – List of features to extract. Valid options include “year”, “month”, “day\_of\_week”, “hour”. Defaults to None.
- **encode\_as\_categories** (*bool*) – Whether day-of-week and month features should be encoded as pandas “category” dtype. This allows OneHotEncoders to encode these features. Defaults to False.
- **date\_index** (*str*) – Name of the column containing the datetime information used to order the data. Ignored.
- **random\_seed** (*int*) – Seed for the random number generator. Defaults to 0.

#### Attributes

<b>hyper-parameter_ranges</b>	{}
<b>model_family</b>	ModelFamily.NONE
<b>modifies_features</b>	True
<b>modifies_target</b>	False
<b>name</b>	DateTime Featurization Component

#### Methods



<code>clone</code>	Constructs a new component with the same parameters and random state.
<code>default_parameters</code>	Returns the default parameters for this component.
<code>describe</code>	Describe a component and its parameters
<code>fit</code>	Fits component to data
<code>fit_transform</code>	Fits on X and transforms X
<code>get_feature_names</code>	Gets the categories of each datetime feature.
<code>load</code>	Loads component at file path
<code>needs_fitting</code>	Returns boolean determining if component needs fitting before
<code>parameters</code>	Returns the parameters which were used to initialize the component
<code>save</code>	Saves component at file path
<code>transform</code>	Transforms data X by creating new features using existing DateTime columns, and then dropping those DateTime columns

**clone** (*self*)

Constructs a new component with the same parameters and random state.

**Returns** A new instance of this component with identical parameters and random state.

**default\_parameters** (*cls*)

Returns the default parameters for this component.

Our convention is that `Component.default_parameters == Component().parameters`.

**Returns** default parameters for this component.

**Return type** dict

**describe** (*self*, *print\_name=False*, *return\_dict=False*)

Describe a component and its parameters

**Parameters**

- **print\_name** (*bool*, *optional*) – whether to print name of component
- **return\_dict** (*bool*, *optional*) – whether to return description as dictionary in the format {"name": name, "parameters": parameters}

**Returns** prints and returns dictionary

**Return type** None or dict

**fit** (*self*, *X*, *y=None*)

Fits component to data

**Parameters**

- **X** (*list*, *pd.DataFrame* or *np.ndarray*) – The input training data of shape [n\_samples, n\_features]
- **y** (*list*, *pd.Series*, *np.ndarray*, *optional*) – The target training data of length [n\_samples]

**Returns** self

**fit\_transform** (*self*, *X*, *y=None*)

Fits on X and transforms X

**Parameters**

- **x** (*pd.DataFrame*) – Data to fit and transform
- **y** (*pd.Series*) – Target data

**Returns** Transformed X

**Return type** *pd.DataFrame*

**get\_feature\_names** (*self*)

Gets the categories of each datetime feature.

**Returns** Dictionary, where each key-value pair is a column name and a dictionary mapping the unique feature values to their integer encoding.

**static load** (*file\_path*)

Loads component at file path

**Parameters** **file\_path** (*str*) – Location to load file

**Returns** *ComponentBase* object

**needs\_fitting** (*self*)

Returns boolean determining if component needs fitting before calling `predict`, `predict_proba`, `transform`, or `feature_importances`. This can be overridden to `False` for components that do not need to be fit or whose fit methods do nothing.

**property parameters** (*self*)

Returns the parameters which were used to initialize the component

**save** (*self, file\_path, pickle\_protocol=cloudpickle.DEFAULT\_PROTOCOL*)

Saves component at file path

**Parameters**

- **file\_path** (*str*) – Location to save file
- **pickle\_protocol** (*int*) – The pickle data stream format.

**Returns** `None`

**transform** (*self, X, y=None*)

Transforms data X by creating new features using existing `DateTime` columns, and then dropping those `DateTime` columns

**Parameters**

- **x** (*pd.DataFrame*) – Data to transform
- **y** (*pd.Series, optional*) – Ignored.

**Returns** Transformed X

**Return type** *pd.DataFrame*

## delayed\_feature\_transformer

### Module Contents

### Classes Summary

<i>DelayedFeatureTransformer</i>	Transformer that delays input features and target variable for time series problems.
----------------------------------	--

### Contents

**class** evalml.pipelines.components.transformers.preprocessing.delayed\_feature\_transformer.D

Transformer that delays input features and target variable for time series problems.

#### Parameters

- **date\_index** (*str*) – Name of the column containing the datetime information used to order the data. Ignored.
- **max\_delay** (*int*) – Maximum number of time units to delay each feature. Defaults to 2.
- **delay\_features** (*bool*) – Whether to delay the input features. Defaults to True.
- **delay\_target** (*bool*) – Whether to delay the target. Defaults to True.
- **gap** (*int*) – The number of time units between when the features are collected and when the target is collected. For example, if you are predicting the next time step's target, gap=1. This is only needed because when gap=0, we need to be sure to start the lagging of the target variable at 1. Defaults to 1.
- **random\_seed** (*int*) – Seed for the random number generator. This transformer performs the same regardless of the random seed provided.

#### Attributes

<b>hyper-parameter_ranges</b>	{}
<b>model_family</b>	ModelFamily.NONE
<b>modifies_features</b>	True
<b>modifies_target</b>	False
<b>name</b>	Delayed Feature Transformer
<b>needs_fitting</b>	False

## Methods

<code>clone</code>	Constructs a new component with the same parameters and random state.
<code>default_parameters</code>	Returns the default parameters for this component.
<code>describe</code>	Describe a component and its parameters
<code>fit</code>	Fits the DelayFeatureTransformer.
<code>fit_transform</code>	Fits on X and transforms X
<code>load</code>	Loads component at file path
<code>parameters</code>	Returns the parameters which were used to initialize the component
<code>save</code>	Saves component at file path
<code>transform</code>	Computes the delayed features for all features in X and y.

**clone** (*self*)

Constructs a new component with the same parameters and random state.

**Returns** A new instance of this component with identical parameters and random state.

**default\_parameters** (*cls*)

Returns the default parameters for this component.

Our convention is that `Component.default_parameters == Component().parameters`.

**Returns** default parameters for this component.

**Return type** dict

**describe** (*self*, *print\_name=False*, *return\_dict=False*)

Describe a component and its parameters

**Parameters**

- **print\_name** (*bool*, *optional*) – whether to print name of component
- **return\_dict** (*bool*, *optional*) – whether to return description as dictionary in the format {"name": name, "parameters": parameters}

**Returns** prints and returns dictionary

**Return type** None or dict

**fit** (*self*, *X*, *y=None*)

Fits the DelayFeatureTransformer.

**Parameters**

- **X** (*pd.DataFrame* or *np.ndarray*) – The input training data of shape [n\_samples, n\_features]
- **y** (*pd.Series*, *optional*) – The target training data of length [n\_samples]

**Returns** self

**fit\_transform** (*self*, *X*, *y*)

Fits on X and transforms X

**Parameters**

- **X** (*pd.DataFrame*) – Data to fit and transform
- **y** (*pd.Series*) – Target data

**Returns** Transformed X

**Return type** `pd.DataFrame`

**static load** (*file\_path*)

Loads component at file path

**Parameters** `file_path` (*str*) – Location to load file

**Returns** ComponentBase object

**property parameters** (*self*)

Returns the parameters which were used to initialize the component

**save** (*self*, *file\_path*, *pickle\_protocol=cloudpickle.DEFAULT\_PROTOCOL*)

Saves component at file path

**Parameters**

- `file_path` (*str*) – Location to save file
- `pickle_protocol` (*int*) – The pickle data stream format.

**Returns** None

**transform** (*self*, *X*, *y=None*)

Computes the delayed features for all features in X and y.

For each feature in X, it will add a column to the output dataframe for each delay in the (inclusive) range [1, max\_delay]. The values of each delayed feature are simply the original feature shifted forward in time by the delay amount. For example, a delay of 3 units means that the feature value at row n will be taken from the n-3rd row of that feature

If y is not None, it will also compute the delayed values for the target variable.

**Parameters**

- `X` (*pd.DataFrame* or *None*) – Data to transform. None is expected when only the target variable is being used.
- `y` (*pd.Series*, or *None*) – Target.

**Returns** Transformed X.

**Return type** `pd.DataFrame`

## drop\_null\_columns

## Module Contents

## Classes Summary

---

*DropNullColumns*

Transformer to drop features whose percentage of NaN values exceeds a specified threshold.

---

## Contents

**class** evalml.pipelines.components.transformers.preprocessing.drop\_null\_columns.**DropNullColumns**

Transformer to drop features whose percentage of NaN values exceeds a specified threshold.

### Parameters

- **pct\_null\_threshold** (*float*) – The percentage of NaN values in an input feature to drop. Must be a value between [0, 1] inclusive. If equal to 0.0, will drop columns with any null values. If equal to 1.0, will drop columns with all null values. Defaults to 0.95.
- **random\_seed** (*int*) – Seed for the random number generator. Defaults to 0.

### Attributes

<b>hyper-parameter_ranges</b>	{}
<b>model_family</b>	ModelFamily.NONE
<b>modifies_features</b>	True
<b>modifies_target</b>	False
<b>name</b>	Drop Null Columns Transformer

### Methods

<i>clone</i>	Constructs a new component with the same parameters and random state.
<i>default_parameters</i>	Returns the default parameters for this component.
<i>describe</i>	Describe a component and its parameters
<i>fit</i>	Fits component to data
<i>fit_transform</i>	Fits on X and transforms X
<i>load</i>	Loads component at file path
<i>needs_fitting</i>	Returns boolean determining if component needs fitting before
<i>parameters</i>	Returns the parameters which were used to initialize the component
<i>save</i>	Saves component at file path
<i>transform</i>	Transforms data X by dropping columns that exceed the threshold of null values.

**clone** (*self*)

Constructs a new component with the same parameters and random state.

**Returns** A new instance of this component with identical parameters and random state.

**default\_parameters** (*cls*)

Returns the default parameters for this component.

Our convention is that `Component.default_parameters == Component().parameters`.

**Returns** default parameters for this component.

**Return type** dict

**describe** (*self*, *print\_name=False*, *return\_dict=False*)

Describe a component and its parameters

**Parameters**

- **print\_name** (*bool*, *optional*) – whether to print name of component
- **return\_dict** (*bool*, *optional*) – whether to return description as dictionary in the format {"name": name, "parameters": parameters}

**Returns** prints and returns dictionary

**Return type** None or dict

**fit** (*self*, *X*, *y=None*)

Fits component to data

**Parameters**

- **X** (*list*, *pd.DataFrame* or *np.ndarray*) – The input training data of shape [n\_samples, n\_features]
- **y** (*list*, *pd.Series*, *np.ndarray*, *optional*) – The target training data of length [n\_samples]

**Returns** self

**fit\_transform** (*self*, *X*, *y=None*)

Fits on X and transforms X

**Parameters**

- **X** (*pd.DataFrame*) – Data to fit and transform
- **y** (*pd.Series*) – Target data

**Returns** Transformed X

**Return type** pd.DataFrame

**static load** (*file\_path*)

Loads component at file path

**Parameters** **file\_path** (*str*) – Location to load file

**Returns** ComponentBase object

**needs\_fitting** (*self*)

Returns boolean determining if component needs fitting before calling predict, predict\_proba, transform, or feature\_importances. This can be overridden to False for components that do not need to be fit or whose fit methods do nothing.

**property parameters** (*self*)

Returns the parameters which were used to initialize the component

**save** (*self*, *file\_path*, *pickle\_protocol=cloudpickle.DEFAULT\_PROTOCOL*)

Saves component at file path

**Parameters**

- **file\_path** (*str*) – Location to save file
- **pickle\_protocol** (*int*) – The pickle data stream format.

**Returns** None

**transform** (*self*, *X*, *y=None*)  
Transforms data *X* by dropping columns that exceed the threshold of null values.

- Parameters**
- **x** (*pd.DataFrame*) – Data to transform
  - **y** (*pd.Series*, *optional*) – Ignored.

**Returns** Transformed *X*  
**Return type** *pd.DataFrame*

featuretools

Module Contents

Classes Summary

<i>DFSTransformer</i>	Featuretools DFS component that generates features for the input features.
-----------------------	--

Contents

**class** evalml.pipelines.components.transformers.preprocessing.featuretools.DFSTransformer (*in* *ra*  
*do*  
\*\*

Featuretools DFS component that generates features for the input features.

- Parameters**
- **index** (*string*) – The name of the column that contains the indices. If no column with this name exists, then featuretools.EntitySet() creates a column with this name to serve as the index column. Defaults to ‘index’.
  - **random\_seed** (*int*) – Seed for the random number generator. Defaults to 0.

Attributes

hyper-parameter_ranges	{}
model_family	ModelFamily.NONE
modifies_features	True
modifies_target	False
name	DFS Transformer

Methods



<code>clone</code>	Constructs a new component with the same parameters and random state.
<code>default_parameters</code>	Returns the default parameters for this component.
<code>describe</code>	Describe a component and its parameters
<code>fit</code>	Fits the DFSTransformer Transformer component.
<code>fit_transform</code>	Fits on X and transforms X
<code>load</code>	Loads component at file path
<code>needs_fitting</code>	Returns boolean determining if component needs fitting before
<code>parameters</code>	Returns the parameters which were used to initialize the component
<code>save</code>	Saves component at file path
<code>transform</code>	Computes the feature matrix for the input X using featuretools' dfs algorithm.

**clone** (*self*)

Constructs a new component with the same parameters and random state.

**Returns** A new instance of this component with identical parameters and random state.

**default\_parameters** (*cls*)

Returns the default parameters for this component.

Our convention is that `Component.default_parameters == Component().parameters`.

**Returns** default parameters for this component.

**Return type** dict

**describe** (*self*, *print\_name=False*, *return\_dict=False*)

Describe a component and its parameters

**Parameters**

- **print\_name** (*bool*, *optional*) – whether to print name of component
- **return\_dict** (*bool*, *optional*) – whether to return description as dictionary in the format {"name": name, "parameters": parameters}

**Returns** prints and returns dictionary

**Return type** None or dict

**fit** (*self*, *X*, *y=None*)

Fits the DFSTransformer Transformer component.

**Parameters**

- **X** (*pd.DataFrame*, *np.array*) – The input data to transform, of shape [n\_samples, n\_features]
- **y** (*pd.Series*, *np.ndarray*, *optional*) – The target training data of length [n\_samples]

**Returns** self

**fit\_transform** (*self*, *X*, *y=None*)

Fits on X and transforms X

**Parameters**

- **X** (*pd.DataFrame*) – Data to fit and transform

- **y** (*pd.Series*) – Target data

**Returns** Transformed X

**Return type** *pd.DataFrame*

**static load** (*file\_path*)

Loads component at file path

**Parameters** **file\_path** (*str*) – Location to load file

**Returns** *ComponentBase* object

**needs\_fitting** (*self*)

Returns boolean determining if component needs fitting before calling *predict*, *predict\_proba*, *transform*, or *feature\_importances*. This can be overridden to *False* for components that do not need to be fit or whose fit methods do nothing.

**property parameters** (*self*)

Returns the parameters which were used to initialize the component

**save** (*self*, *file\_path*, *pickle\_protocol=cloudpickle.DEFAULT\_PROTOCOL*)

Saves component at file path

**Parameters**

- **file\_path** (*str*) – Location to save file
- **pickle\_protocol** (*int*) – The pickle data stream format.

**Returns** *None*

**transform** (*self*, *X*, *y=None*)

Computes the feature matrix for the input X using featuretools' dfs algorithm.

**Parameters**

- **X** (*pd.DataFrame* or *np.ndarray*) – The input training data to transform. Has shape [n\_samples, n\_features]
- **y** (*pd.Series*, *optional*) – Ignored.

**Returns** Feature matrix

**Return type** *pd.DataFrame*

## log\_transformer

## Module Contents

## Classes Summary

---

*LogTransformer*

Applies a log transformation to the target data.

---

## Contents

**class** evalml.pipelines.components.transformers.preprocessing.log\_transformer.LogTransformer

Applies a log transformation to the target data.

### Attributes

<b>hyper-parameter_ranges</b>	{}
<b>model_family</b>	ModelFamily.NONE
<b>modifies_features</b>	False
<b>modifies_target</b>	True
<b>name</b>	Log Transformer

### Methods

<i>clone</i>	Constructs a new component with the same parameters and random state.
<i>default_parameters</i>	Returns the default parameters for this component.
<i>describe</i>	Describe a component and its parameters
<i>fit</i>	Fits the LogTransformer.
<i>fit_transform</i>	Log transforms the target variable.
<i>inverse_transform</i>	Inverts the transformation done by the transform method.
<i>load</i>	Loads component at file path
<i>needs_fitting</i>	Returns boolean determining if component needs fitting before
<i>parameters</i>	Returns the parameters which were used to initialize the component
<i>save</i>	Saves component at file path
<i>transform</i>	Log transforms the target variable.

**clone** (*self*)

Constructs a new component with the same parameters and random state.

**Returns** A new instance of this component with identical parameters and random state.

**default\_parameters** (*cls*)

Returns the default parameters for this component.

Our convention is that `Component.default_parameters == Component().parameters`.

**Returns** default parameters for this component.

**Return type** dict

**describe** (*self*, *print\_name=False*, *return\_dict=False*)

Describe a component and its parameters

### Parameters

- **print\_name** (*bool*, *optional*) – whether to print name of component

- **return\_dict** (*bool, optional*) – whether to return description as dictionary in the format {"name": name, "parameters": parameters}

**Returns** prints and returns dictionary

**Return type** None or dict

**fit** (*self, X, y=None*)

Fits the LogTransformer.

**Parameters**

- **x** (*pd.DataFrame or np.ndarray*) – Ignored.
- **y** (*pd.Series, optional*) – Ignored.

**Returns** self

**fit\_transform** (*self, X, y=None*)

Log transforms the target variable.

**Parameters**

- **x** (*pd.DataFrame, optional*) – Ignored.
- **y** (*pd.Series*) – Target variable to log transform.

**Returns**

**The input features are returned without modification. The target** variable y is log transformed.

**Return type** tuple of pd.DataFrame, pd.Series

**inverse\_transform** (*self, y*)

Inverts the transformation done by the transform method.

**Arguments:** y (pd.Series): Target transformed by this component.

**Returns** Target without the transformation.

**Return type** pd.Series

**static load** (*file\_path*)

Loads component at file path

**Parameters** **file\_path** (*str*) – Location to load file

**Returns** ComponentBase object

**needs\_fitting** (*self*)

Returns boolean determining if component needs fitting before calling predict, predict\_proba, transform, or feature\_importances. This can be overridden to False for components that do not need to be fit or whose fit methods do nothing.

**property parameters** (*self*)

Returns the parameters which were used to initialize the component

**save** (*self, file\_path, pickle\_protocol=cloudpickle.DEFAULT\_PROTOCOL*)

Saves component at file path

**Parameters**

- **file\_path** (*str*) – Location to save file
- **pickle\_protocol** (*int*) – The pickle data stream format.

**Returns** None

**transform** (*self*, *X*, *y=None*)

Log transforms the target variable.

**Parameters**

- **X** (*pd.DataFrame*, *optional*) – Ignored.
- **y** (*pd.Series*) – Target data to log transform.

**Returns**

**The input features are returned without modification. The target** variable *y* is log transformed.

**Return type** tuple of *pd.DataFrame*, *pd.Series*

lsa

## Module Contents

## Classes Summary

<i>LSA</i>	Transformer to calculate the Latent Semantic Analysis Values of text input.
------------	---

## Contents

**class** evalml.pipelines.components.transformers.preprocessing.lsa.**LSA** (*random\_seed=0*, *\*\*kwargs*)

Transformer to calculate the Latent Semantic Analysis Values of text input.

**Parameters** **random\_seed** (*int*) – Seed for the random number generator. Defaults to 0.

**Attributes**

<b>hyper-parameter_ranges</b>	{}
<b>model_family</b>	ModelFamily.NONE
<b>modifies_features</b>	True
<b>modifies_target</b>	False
<b>name</b>	LSA Transformer

**Methods**

<i>clone</i>	Constructs a new component with the same parameters and random state.
<i>default_parameters</i>	Returns the default parameters for this component.
<i>describe</i>	Describe a component and its parameters

continues on next page

Table 452 – continued from previous page

<i>fit</i>	Fits component to data
<i>fit_transform</i>	Fits on X and transforms X
<i>load</i>	Loads component at file path
<i>needs_fitting</i>	Returns boolean determining if component needs fitting before
<i>parameters</i>	Returns the parameters which were used to initialize the component
<i>save</i>	Saves component at file path
<i>transform</i>	Transforms data X by applying the LSA pipeline.

**clone** (*self*)

Constructs a new component with the same parameters and random state.

**Returns** A new instance of this component with identical parameters and random state.

**default\_parameters** (*cls*)

Returns the default parameters for this component.

Our convention is that `Component.default_parameters == Component().parameters`.

**Returns** default parameters for this component.

**Return type** dict

**describe** (*self*, *print\_name=False*, *return\_dict=False*)

Describe a component and its parameters

**Parameters**

- **print\_name** (*bool*, *optional*) – whether to print name of component
- **return\_dict** (*bool*, *optional*) – whether to return description as dictionary in the format {"name": name, "parameters": parameters}

**Returns** prints and returns dictionary

**Return type** None or dict

**fit** (*self*, *X*, *y=None*)

Fits component to data

**Parameters**

- **X** (*list*, *pd.DataFrame* or *np.ndarray*) – The input training data of shape [n\_samples, n\_features]
- **y** (*list*, *pd.Series*, *np.ndarray*, *optional*) – The target training data of length [n\_samples]

**Returns** self

**fit\_transform** (*self*, *X*, *y=None*)

Fits on X and transforms X

**Parameters**

- **X** (*pd.DataFrame*) – Data to fit and transform
- **y** (*pd.Series*) – Target data

**Returns** Transformed X

**Return type** pd.DataFrame

**static load** (*file\_path*)

Loads component at file path

**Parameters** *file\_path* (*str*) – Location to load file

**Returns** ComponentBase object

**needs\_fitting** (*self*)

Returns boolean determining if component needs fitting before calling predict, predict\_proba, transform, or feature\_importances. This can be overridden to False for components that do not need to be fit or whose fit methods do nothing.

**property parameters** (*self*)

Returns the parameters which were used to initialize the component

**save** (*self*, *file\_path*, *pickle\_protocol=cloudpickle.DEFAULT\_PROTOCOL*)

Saves component at file path

**Parameters**

- **file\_path** (*str*) – Location to save file
- **pickle\_protocol** (*int*) – The pickle data stream format.

**Returns** None

**transform** (*self*, *X*, *y=None*)

Transforms data X by applying the LSA pipeline.

**Parameters**

- **X** (*pd.DataFrame*) – The data to transform.
- **y** (*pd.Series*, *optional*) – Ignored.

**Returns**

**Transformed X. The original column is removed and replaced with two columns of the format *LSA(original\_column\_name)[feature\_number]*, where *feature\_number* is 0 or 1.**

**Return type** *pd.DataFrame*

## polynomial\_detrender

### Module Contents

### Classes Summary

---

*PolynomialDetrender*

Removes trends from time series by fitting a polynomial to the data.

---

## Contents

**class** evalml.pipelines.components.transformers.preprocessing.polynomial\_detrender.**PolynomialDetrender**

Removes trends from time series by fitting a polynomial to the data.

### Parameters

- **degree** (*int*) – Degree for the polynomial. If 1, linear model is fit to the data. If 2, quadratic model is fit, etc. Defaults to 1.
- **random\_seed** (*int*) – Seed for the random number generator. Defaults to 0.

### Attributes

<b>hyper-parameter_ranges</b>	{ “degree”: Integer(1, 3)}
<b>model_family</b>	ModelFamily.NONE
<b>modifies_features</b>	False
<b>modifies_target</b>	True
<b>name</b>	Polynomial Detrender

### Methods

<i>clone</i>	Constructs a new component with the same parameters and random state.
<i>default_parameters</i>	Returns the default parameters for this component.
<i>describe</i>	Describe a component and its parameters
<i>fit</i>	Fits the PolynomialDetrender.
<i>fit_transform</i>	Removes fitted trend from target variable.
<i>inverse_transform</i>	Adds back fitted trend to target variable.
<i>load</i>	Loads component at file path
<i>needs_fitting</i>	Returns boolean determining if component needs fitting before
<i>parameters</i>	Returns the parameters which were used to initialize the component
<i>save</i>	Saves component at file path
<i>transform</i>	Removes fitted trend from target variable.

#### **clone** (*self*)

Constructs a new component with the same parameters and random state.

**Returns** A new instance of this component with identical parameters and random state.

#### **default\_parameters** (*cls*)

Returns the default parameters for this component.

Our convention is that `Component.default_parameters == Component().parameters`.

**Returns** default parameters for this component.

**Return type** dict



**describe** (*self*, *print\_name=False*, *return\_dict=False*)

Describe a component and its parameters

**Parameters**

- **print\_name** (*bool*, *optional*) – whether to print name of component
- **return\_dict** (*bool*, *optional*) – whether to return description as dictionary in the format {"name": name, "parameters": parameters}

**Returns** prints and returns dictionary

**Return type** None or dict

**fit** (*self*, *X*, *y=None*)

Fits the PolynomialDetrender.

**Parameters**

- **X** (*pd.DataFrame*, *optional*) – Ignored.
- **y** (*pd.Series*) – Target variable to detrend.

**Returns** self

**fit\_transform** (*self*, *X*, *y=None*)

Removes fitted trend from target variable.

**Parameters**

- **X** (*pd.DataFrame*, *optional*) – Ignored.
- **y** (*pd.Series*) – Target variable to detrend.

**Returns**

**The first element are the input features returned without modification.** The second element is the target variable y with the fitted trend removed.

**Return type** tuple of pd.DataFrame, pd.Series

**inverse\_transform** (*self*, *y*)

Adds back fitted trend to target variable.

**Parameters**

- **X** (*pd.DataFrame*, *optional*) – Ignored.
- **y** (*pd.Series*) – Target variable.

**Returns**

**The first element are the input features returned without modification.** The second element is the target variable y with the trend added back.

**Return type** tuple of pd.DataFrame, pd.Series

**static load** (*file\_path*)

Loads component at file path

**Parameters** **file\_path** (*str*) – Location to load file

**Returns** ComponentBase object

**needs\_fitting** (*self*)

Returns boolean determining if component needs fitting before calling predict, predict\_proba, transform, or feature\_importances. This can be overridden to False for components that do not need to be fit or whose fit methods do nothing.

**property parameters** (*self*)

Returns the parameters which were used to initialize the component

**save** (*self*, *file\_path*, *pickle\_protocol*=*cloudpickle.DEFAULT\_PROTOCOL*)

Saves component at file path

**Parameters**

- **file\_path** (*str*) – Location to save file
- **pickle\_protocol** (*int*) – The pickle data stream format.

**Returns** None

**transform** (*self*, *X*, *y*=*None*)

Removes fitted trend from target variable.

**Parameters**

- **X** (*pd.DataFrame*, *optional*) – Ignored.
- **y** (*pd.Series*) – Target variable to detrend.

**Returns**

**The input features are returned without modification. The target** variable *y* is de-trended

**Return type** tuple of *pd.DataFrame*, *pd.Series*

## text\_featurizer

### Module Contents

### Classes Summary

---

<i>TextFeaturizer</i>	Transformer that can automatically featurize text columns using featuretools' nlp_primitives.
-----------------------	---

---

### Contents

**class** evalml.pipelines.components.transformers.preprocessing.text\_featurizer.**TextFeaturizer**

Transformer that can automatically featurize text columns using featuretools' nlp\_primitives.

Since models cannot handle non-numeric data, any text must be broken down into features that provide useful information about that text. This component splits each text column into several informative features: Diversity Score, Mean Characters per Word, Polarity Score, and LSA (Latent Semantic Analysis). Calling transform on this component will replace any text columns in the given dataset with these numeric columns.

**Parameters** **random\_seed** (*int*) – Seed for the random number generator. Defaults to 0.

**Attributes**

<b>hyper-parameter_ranges</b>	{}
<b>model_family</b>	ModelFamily.NONE
<b>modifies_features</b>	True
<b>modifies_target</b>	False
<b>name</b>	Text Featurization Component

## Methods

<i>clone</i>	Constructs a new component with the same parameters and random state.
<i>default_parameters</i>	Returns the default parameters for this component.
<i>describe</i>	Describe a component and its parameters
<i>fit</i>	Fits component to data
<i>fit_transform</i>	Fits on X and transforms X
<i>load</i>	Loads component at file path
<i>needs_fitting</i>	Returns boolean determining if component needs fitting before
<i>parameters</i>	Returns the parameters which were used to initialize the component
<i>save</i>	Saves component at file path
<i>transform</i>	Transforms data X by creating new features using existing text columns

### **clone** (*self*)

Constructs a new component with the same parameters and random state.

**Returns** A new instance of this component with identical parameters and random state.

### **default\_parameters** (*cls*)

Returns the default parameters for this component.

Our convention is that `Component.default_parameters == Component().parameters`.

**Returns** default parameters for this component.

**Return type** dict

### **describe** (*self*, *print\_name=False*, *return\_dict=False*)

Describe a component and its parameters

#### Parameters

- **print\_name** (*bool*, *optional*) – whether to print name of component
- **return\_dict** (*bool*, *optional*) – whether to return description as dictionary in the format {"name": name, "parameters": parameters}

**Returns** prints and returns dictionary

**Return type** None or dict

### **fit** (*self*, *X*, *y=None*)

Fits component to data

**Parameters**

- **X** (*pd.DataFrame* or *np.ndarray*) – The input training data of shape [n\_samples, n\_features]
- **y** (*pd.Series*, *np.ndarray*, *optional*) – The target training data of length [n\_samples]

**Returns** *self***fit\_transform** (*self*, *X*, *y=None*)

Fits on X and transforms X

**Parameters**

- **X** (*pd.DataFrame*) – Data to fit and transform
- **y** (*pd.Series*) – Target data

**Returns** Transformed X**Return type** *pd.DataFrame***static load** (*file\_path*)

Loads component at file path

**Parameters** **file\_path** (*str*) – Location to load file**Returns** *ComponentBase* object**needs\_fitting** (*self*)

Returns boolean determining if component needs fitting before calling predict, predict\_proba, transform, or feature\_importances. This can be overridden to False for components that do not need to be fit or whose fit methods do nothing.

**property parameters** (*self*)

Returns the parameters which were used to initialize the component

**save** (*self*, *file\_path*, *pickle\_protocol=cloudpickle.DEFAULT\_PROTOCOL*)

Saves component at file path

**Parameters**

- **file\_path** (*str*) – Location to save file
- **pickle\_protocol** (*int*) – The pickle data stream format.

**Returns** *None***transform** (*self*, *X*, *y=None*)

Transforms data X by creating new features using existing text columns

**Parameters**

- **X** (*pd.DataFrame*) – The data to transform.
- **y** (*pd.Series*, *optional*) – Ignored.

**Returns** Transformed X**Return type** *pd.DataFrame*

## text\_transformer

### Module Contents

### Classes Summary

<i>TextTransformer</i>	Base class for all transformers working with text features.
------------------------	---

### Attributes Summary

<i>logger</i>	
---------------	--

### Contents

`evalml.pipelines.components.transformers.preprocessing.text_transformer.logger`

**class** `evalml.pipelines.components.transformers.preprocessing.text_transformer.TextTransformer`

Base class for all transformers working with text features.

#### Parameters

- **component\_obj** (*obj*) – Third-party objects useful in component implementation. Defaults to None.
- **random\_seed** (*int*) – Seed for the random number generator. Defaults to 0.

#### Attributes

<b>model_family</b>	ModelFamily.NONE
<b>modifies_features</b>	True
<b>modifies_target</b>	False

#### Methods

<i>clone</i>	Constructs a new component with the same parameters and random state.
<i>default_parameters</i>	Returns the default parameters for this component.
<i>describe</i>	Describe a component and its parameters
<i>fit</i>	Fits component to data
<i>fit_transform</i>	Fits on X and transforms X
<i>load</i>	Loads component at file path
<i>name</i>	Returns string name of this component

continues on next page

Table 459 – continued from previous page

<code>needs_fitting</code>	Returns boolean determining if component needs fitting before
<code>parameters</code>	Returns the parameters which were used to initialize the component
<code>save</code>	Saves component at file path
<code>transform</code>	Transforms data X.

**clone** (*self*)

Constructs a new component with the same parameters and random state.

**Returns** A new instance of this component with identical parameters and random state.

**default\_parameters** (*cls*)

Returns the default parameters for this component.

Our convention is that `Component.default_parameters == Component().parameters`.

**Returns** default parameters for this component.

**Return type** dict

**describe** (*self*, *print\_name=False*, *return\_dict=False*)

Describe a component and its parameters

**Parameters**

- **print\_name** (*bool*, *optional*) – whether to print name of component
- **return\_dict** (*bool*, *optional*) – whether to return description as dictionary in the format {"name": name, "parameters": parameters}

**Returns** prints and returns dictionary

**Return type** None or dict

**fit** (*self*, *X*, *y=None*)

Fits component to data

**Parameters**

- **X** (*list*, *pd.DataFrame* or *np.ndarray*) – The input training data of shape [n\_samples, n\_features]
- **y** (*list*, *pd.Series*, *np.ndarray*, *optional*) – The target training data of length [n\_samples]

**Returns** self

**fit\_transform** (*self*, *X*, *y=None*)

Fits on X and transforms X

**Parameters**

- **X** (*pd.DataFrame*) – Data to fit and transform
- **y** (*pd.Series*) – Target data

**Returns** Transformed X

**Return type** pd.DataFrame

**static load** (*file\_path*)

Loads component at file path

**Parameters** **file\_path** (*str*) – Location to load file

**Returns** ComponentBase object

**property name** (*cls*)

Returns string name of this component

**needs\_fitting** (*self*)

Returns boolean determining if component needs fitting before calling predict, predict\_proba, transform, or feature\_importances. This can be overridden to False for components that do not need to be fit or whose fit methods do nothing.

**property parameters** (*self*)

Returns the parameters which were used to initialize the component

**save** (*self, file\_path, pickle\_protocol=cloudpickle.DEFAULT\_PROTOCOL*)

Saves component at file path

#### Parameters

- **file\_path** (*str*) – Location to save file
- **pickle\_protocol** (*int*) – The pickle data stream format.

**Returns** None

**transform** (*self, X, y=None*)

Transforms data X.

#### Parameters

- **X** (*pd.DataFrame*) – Data to transform.
- **y** (*pd.Series, optional*) – Target data.

**Returns** Transformed X

**Return type** pd.DataFrame

## transform\_primitive\_components

## Module Contents

## Classes Summary

<i>EmailFeaturizer</i>	Transformer that can automatically extract features from emails.
<i>URLFeaturizer</i>	Transformer that can automatically extract features from URL.

## Contents

**class** evalml.pipelines.components.transformers.preprocessing.transform\_primitive\_components

Transformer that can automatically extract features from emails.

**Parameters** `random_seed` (*int*) – Seed for the random number generator. Defaults to 0.

### Attributes

<b>hyper-parameter_ranges</b>	{}
<b>model_family</b>	ModelFamily.NONE
<b>modifies_features</b>	True
<b>modifies_target</b>	False
<b>name</b>	Email Featurizer

### Methods

<code>clone</code>	Constructs a new component with the same parameters and random state.
<code>default_parameters</code>	Returns the default parameters for this component.
<code>describe</code>	Describe a component and its parameters
<code>fit</code>	Fits component to data
<code>fit_transform</code>	Fits on X and transforms X
<code>load</code>	Loads component at file path
<code>needs_fitting</code>	Returns boolean determining if component needs fitting before
<code>parameters</code>	Returns the parameters which were used to initialize the component
<code>save</code>	Saves component at file path
<code>transform</code>	Transforms data X.

**clone** (*self*)

Constructs a new component with the same parameters and random state.

**Returns** A new instance of this component with identical parameters and random state.

**default\_parameters** (*cls*)

Returns the default parameters for this component.

Our convention is that `Component.default_parameters == Component().parameters`.

**Returns** default parameters for this component.

**Return type** dict

**describe** (*self*, *print\_name=False*, *return\_dict=False*)

Describe a component and its parameters

**Parameters**

- **print\_name** (*bool*, *optional*) – whether to print name of component



- **return\_dict** (*bool, optional*) – whether to return description as dictionary in the format {"name": name, "parameters": parameters}

**Returns** prints and returns dictionary

**Return type** None or dict

**fit** (*self, X, y=None*)

Fits component to data

**Parameters**

- **X** (*list, pd.DataFrame or np.ndarray*) – The input training data of shape [n\_samples, n\_features]
- **y** (*list, pd.Series, np.ndarray, optional*) – The target training data of length [n\_samples]

**Returns** self

**fit\_transform** (*self, X, y=None*)

Fits on X and transforms X

**Parameters**

- **X** (*pd.DataFrame*) – Data to fit and transform
- **y** (*pd.Series*) – Target data

**Returns** Transformed X

**Return type** pd.DataFrame

**static load** (*file\_path*)

Loads component at file path

**Parameters** **file\_path** (*str*) – Location to load file

**Returns** ComponentBase object

**needs\_fitting** (*self*)

Returns boolean determining if component needs fitting before calling predict, predict\_proba, transform, or feature\_importances. This can be overridden to False for components that do not need to be fit or whose fit methods do nothing.

**property parameters** (*self*)

Returns the parameters which were used to initialize the component

**save** (*self, file\_path, pickle\_protocol=cloudpickle.DEFAULT\_PROTOCOL*)

Saves component at file path

**Parameters**

- **file\_path** (*str*) – Location to save file
- **pickle\_protocol** (*int*) – The pickle data stream format.

**Returns** None

**transform** (*self, X, y=None*)

Transforms data X.

**Parameters**

- **X** (*pd.DataFrame*) – Data to transform.
- **y** (*pd.Series, optional*) – Target data.

**Returns** Transformed X

**Return type** `pd.DataFrame`

**class** `evalml.pipelines.components.transformers.preprocessing.transform_primitive_component`

Transformer that can automatically extract features from URL.

**Parameters** `random_seed` (*int*) – Seed for the random number generator. Defaults to 0.

**Attributes**

<b>hyper-parameter_ranges</b>	<code>{}</code>
<b>model_family</b>	<code>ModelFamily.NONE</code>
<b>modifies_features</b>	<code>True</code>
<b>modifies_target</b>	<code>False</code>
<b>name</b>	URL Featurizer

**Methods**

<code>clone</code>	Constructs a new component with the same parameters and random state.
<code>default_parameters</code>	Returns the default parameters for this component.
<code>describe</code>	Describe a component and its parameters
<code>fit</code>	Fits component to data
<code>fit_transform</code>	Fits on X and transforms X
<code>load</code>	Loads component at file path
<code>needs_fitting</code>	Returns boolean determining if component needs fitting before
<code>parameters</code>	Returns the parameters which were used to initialize the component
<code>save</code>	Saves component at file path
<code>transform</code>	Transforms data X.

**clone** (*self*)

Constructs a new component with the same parameters and random state.

**Returns** A new instance of this component with identical parameters and random state.

**default\_parameters** (*cls*)

Returns the default parameters for this component.

Our convention is that `Component.default_parameters == Component().parameters`.

**Returns** default parameters for this component.

**Return type** `dict`

**describe** (*self*, *print\_name=False*, *return\_dict=False*)

Describe a component and its parameters

**Parameters**

- **print\_name** (*bool*, *optional*) – whether to print name of component

- **return\_dict** (*bool, optional*) – whether to return description as dictionary in the format {"name": name, "parameters": parameters}

**Returns** prints and returns dictionary

**Return type** None or dict

**fit** (*self, X, y=None*)

Fits component to data

**Parameters**

- **X** (*list, pd.DataFrame or np.ndarray*) – The input training data of shape [n\_samples, n\_features]
- **y** (*list, pd.Series, np.ndarray, optional*) – The target training data of length [n\_samples]

**Returns** self

**fit\_transform** (*self, X, y=None*)

Fits on X and transforms X

**Parameters**

- **X** (*pd.DataFrame*) – Data to fit and transform
- **y** (*pd.Series*) – Target data

**Returns** Transformed X

**Return type** pd.DataFrame

**static load** (*file\_path*)

Loads component at file path

**Parameters** **file\_path** (*str*) – Location to load file

**Returns** ComponentBase object

**needs\_fitting** (*self*)

Returns boolean determining if component needs fitting before calling predict, predict\_proba, transform, or feature\_importances. This can be overridden to False for components that do not need to be fit or whose fit methods do nothing.

**property parameters** (*self*)

Returns the parameters which were used to initialize the component

**save** (*self, file\_path, pickle\_protocol=cloudpickle.DEFAULT\_PROTOCOL*)

Saves component at file path

**Parameters**

- **file\_path** (*str*) – Location to save file
- **pickle\_protocol** (*int*) – The pickle data stream format.

**Returns** None

**transform** (*self, X, y=None*)

Transforms data X.

**Parameters**

- **X** (*pd.DataFrame*) – Data to transform.
- **y** (*pd.Series, optional*) – Target data.

**Returns** Transformed X  
**Return type** pd.DataFrame

Package Contents

Classes Summary

<i>DateTimeFeaturizer</i>	Transformer that can automatically extract features from datetime columns.
<i>DelayedFeatureTransformer</i>	Transformer that delays input features and target variable for time series problems.
<i>DFSTransformer</i>	Featuretools DFS component that generates features for the input features.
<i>DropNullColumns</i>	Transformer to drop features whose percentage of NaN values exceeds a specified threshold.
<i>EmailFeaturizer</i>	Transformer that can automatically extract features from emails.
<i>LogTransformer</i>	Applies a log transformation to the target data.
<i>LSA</i>	Transformer to calculate the Latent Semantic Analysis Values of text input.
<i>PolynomialDetrender</i>	Removes trends from time series by fitting a polynomial to the data.
<i>TextFeaturizer</i>	Transformer that can automatically featurize text columns using featuretools' nlp_primitives.
<i>TextTransformer</i>	Base class for all transformers working with text features.
<i>URLFeaturizer</i>	Transformer that can automatically extract features from URL.

Contents

**class** evalml.pipelines.components.transformers.preprocessing.**DateTimeFeaturizer** (*features\_to\_extract*, *encode\_as\_categories*, *date\_index=None*, *random\_seed=0*, *\*\*kwargs*)

Transformer that can automatically extract features from datetime columns.

Parameters

- **features\_to\_extract** (*list*) – List of features to extract. Valid options include “year”, “month”, “day\_of\_week”, “hour”. Defaults to None.
- **encode\_as\_categories** (*bool*) – Whether day-of-week and month features should be encoded as pandas “category” dtype. This allows OneHotEncoders to encode these features. Defaults to False.
- **date\_index** (*str*) – Name of the column containing the datetime information used to order the data. Ignored.

- **random\_seed** (*int*) – Seed for the random number generator. Defaults to 0.

#### Attributes

<b>hyper-parameter_ranges</b>	{}
<b>model_family</b>	ModelFamily.NONE
<b>modifies_features</b>	True
<b>modifies_target</b>	False
<b>name</b>	DateTime Featurization Component

#### Methods

<i>clone</i>	Constructs a new component with the same parameters and random state.
<i>default_parameters</i>	Returns the default parameters for this component.
<i>describe</i>	Describe a component and its parameters
<i>fit</i>	Fits component to data
<i>fit_transform</i>	Fits on X and transforms X
<i>get_feature_names</i>	Gets the categories of each datetime feature.
<i>load</i>	Loads component at file path
<i>needs_fitting</i>	Returns boolean determining if component needs fitting before
<i>parameters</i>	Returns the parameters which were used to initialize the component
<i>save</i>	Saves component at file path
<i>transform</i>	Transforms data X by creating new features using existing DateTime columns, and then dropping those DateTime columns

#### **clone** (*self*)

Constructs a new component with the same parameters and random state.

**Returns** A new instance of this component with identical parameters and random state.

#### **default\_parameters** (*cls*)

Returns the default parameters for this component.

Our convention is that `Component.default_parameters == Component().parameters`.

**Returns** default parameters for this component.

**Return type** dict

#### **describe** (*self*, *print\_name=False*, *return\_dict=False*)

Describe a component and its parameters

#### Parameters

- **print\_name** (*bool*, *optional*) – whether to print name of component
- **return\_dict** (*bool*, *optional*) – whether to return description as dictionary in the format {"name": name, "parameters": parameters}

**Returns** prints and returns dictionary

**Return type** None or dict

**fit** (*self*, *X*, *y=None*)

Fits component to data

**Parameters**

- **X** (*list*, *pd.DataFrame* or *np.ndarray*) – The input training data of shape [n\_samples, n\_features]
- **y** (*list*, *pd.Series*, *np.ndarray*, *optional*) – The target training data of length [n\_samples]

**Returns** *self*

**fit\_transform** (*self*, *X*, *y=None*)

Fits on X and transforms X

**Parameters**

- **X** (*pd.DataFrame*) – Data to fit and transform
- **y** (*pd.Series*) – Target data

**Returns** Transformed X

**Return type** *pd.DataFrame*

**get\_feature\_names** (*self*)

Gets the categories of each datetime feature.

**Returns** Dictionary, where each key-value pair is a column name and a dictionary mapping the unique feature values to their integer encoding.

**static load** (*file\_path*)

Loads component at file path

**Parameters** **file\_path** (*str*) – Location to load file

**Returns** ComponentBase object

**needs\_fitting** (*self*)

Returns boolean determining if component needs fitting before calling predict, predict\_proba, transform, or feature\_importances. This can be overridden to False for components that do not need to be fit or whose fit methods do nothing.

**property parameters** (*self*)

Returns the parameters which were used to initialize the component

**save** (*self*, *file\_path*, *pickle\_protocol=cloudpickle.DEFAULT\_PROTOCOL*)

Saves component at file path

**Parameters**

- **file\_path** (*str*) – Location to save file
- **pickle\_protocol** (*int*) – The pickle data stream format.

**Returns** None

**transform** (*self*, *X*, *y=None*)

Transforms data X by creating new features using existing DateTime columns, and then dropping those DateTime columns

**Parameters**

- **X** (*pd.DataFrame*) – Data to transform

- **y** (*pd.Series, optional*) – Ignored.

**Returns** Transformed X

**Return type** `pd.DataFrame`

**class** `evalml.pipelines.components.transformers.preprocessing.DelayedFeatureTransformer` (*date\_index, max\_delay, delay\_features, delay\_target, gap, random\_seed*, \*\*kwargs)

Transformer that delays input features and target variable for time series problems.

#### Parameters

- **date\_index** (*str*) – Name of the column containing the datetime information used to order the data. Ignored.
- **max\_delay** (*int*) – Maximum number of time units to delay each feature. Defaults to 2.
- **delay\_features** (*bool*) – Whether to delay the input features. Defaults to True.
- **delay\_target** (*bool*) – Whether to delay the target. Defaults to True.
- **gap** (*int*) – The number of time units between when the features are collected and when the target is collected. For example, if you are predicting the next time step's target, `gap=1`. This is only needed because when `gap=0`, we need to be sure to start the lagging of the target variable at 1. Defaults to 1.
- **random\_seed** (*int*) – Seed for the random number generator. This transformer performs the same regardless of the random seed provided.

#### Attributes

<b>hyper-parameter_ranges</b>	{}
<b>model_family</b>	ModelFamily.NONE
<b>modifies_features</b>	True
<b>modifies_target</b>	False
<b>name</b>	Delayed Feature Transformer
<b>needs_fitting</b>	False

#### Methods

<code>clone</code>	Constructs a new component with the same parameters and random state.
<code>default_parameters</code>	Returns the default parameters for this component.
<code>describe</code>	Describe a component and its parameters
<code>fit</code>	Fits the DelayFeatureTransformer.

continues on next page

Table 465 – continued from previous page

<code>fit_transform</code>	Fits on X and transforms X
<code>load</code>	Loads component at file path
<code>parameters</code>	Returns the parameters which were used to initialize the component
<code>save</code>	Saves component at file path
<code>transform</code>	Computes the delayed features for all features in X and y.

**clone** (*self*)

Constructs a new component with the same parameters and random state.

**Returns** A new instance of this component with identical parameters and random state.

**default\_parameters** (*cls*)

Returns the default parameters for this component.

Our convention is that `Component.default_parameters == Component().parameters`.

**Returns** default parameters for this component.

**Return type** dict

**describe** (*self*, *print\_name=False*, *return\_dict=False*)

Describe a component and its parameters

**Parameters**

- **print\_name** (*bool*, *optional*) – whether to print name of component
- **return\_dict** (*bool*, *optional*) – whether to return description as dictionary in the format {"name": name, "parameters": parameters}

**Returns** prints and returns dictionary

**Return type** None or dict

**fit** (*self*, *X*, *y=None*)

Fits the DelayFeatureTransformer.

**Parameters**

- **X** (*pd.DataFrame* or *np.ndarray*) – The input training data of shape [n\_samples, n\_features]
- **y** (*pd.Series*, *optional*) – The target training data of length [n\_samples]

**Returns** self

**fit\_transform** (*self*, *X*, *y*)

Fits on X and transforms X

**Parameters**

- **X** (*pd.DataFrame*) – Data to fit and transform
- **y** (*pd.Series*) – Target data

**Returns** Transformed X

**Return type** pd.DataFrame

**static load** (*file\_path*)

Loads component at file path

**Parameters** **file\_path** (*str*) – Location to load file



**Returns** ComponentBase object

**property parameters** (*self*)

Returns the parameters which were used to initialize the component

**save** (*self*, *file\_path*, *pickle\_protocol=cloudpickle.DEFAULT\_PROTOCOL*)

Saves component at file path

**Parameters**

- **file\_path** (*str*) – Location to save file
- **pickle\_protocol** (*int*) – The pickle data stream format.

**Returns** None

**transform** (*self*, *X*, *y=None*)

Computes the delayed features for all features in X and y.

For each feature in X, it will add a column to the output dataframe for each delay in the (inclusive) range [1, max\_delay]. The values of each delayed feature are simply the original feature shifted forward in time by the delay amount. For example, a delay of 3 units means that the feature value at row n will be taken from the n-3rd row of that feature

If y is not None, it will also compute the delayed values for the target variable.

**Parameters**

- **X** (*pd.DataFrame* or *None*) – Data to transform. None is expected when only the target variable is being used.
- **y** (*pd.Series*, or *None*) – Target.

**Returns** Transformed X.

**Return type** pd.DataFrame

```
class evalml.pipelines.components.transformers.preprocessing.DFSTransformer (index='index',  
                                                                           ran-  
                                                                           dom_seed=0,  
                                                                           **kwargs)
```

Featuretools DFS component that generates features for the input features.

**Parameters**

- **index** (*string*) – The name of the column that contains the indices. If no column with this name exists, then featuretools.EntitySet() creates a column with this name to serve as the index column. Defaults to 'index'.
- **random\_seed** (*int*) – Seed for the random number generator. Defaults to 0.

**Attributes**

<b>hyper-parameter_ranges</b>	{}
<b>model_family</b>	ModelFamily.NONE
<b>modifies_features</b>	True
<b>modifies_target</b>	False
<b>name</b>	DFS Transformer

## Methods

<code>clone</code>	Constructs a new component with the same parameters and random state.
<code>default_parameters</code>	Returns the default parameters for this component.
<code>describe</code>	Describe a component and its parameters
<code>fit</code>	Fits the DFSTransformer Transformer component.
<code>fit_transform</code>	Fits on X and transforms X
<code>load</code>	Loads component at file path
<code>needs_fitting</code>	Returns boolean determining if component needs fitting before
<code>parameters</code>	Returns the parameters which were used to initialize the component
<code>save</code>	Saves component at file path
<code>transform</code>	Computes the feature matrix for the input X using featuretools' dfs algorithm.

### **clone** (*self*)

Constructs a new component with the same parameters and random state.

**Returns** A new instance of this component with identical parameters and random state.

### **default\_parameters** (*cls*)

Returns the default parameters for this component.

Our convention is that `Component.default_parameters == Component().parameters`.

**Returns** default parameters for this component.

**Return type** dict

### **describe** (*self*, *print\_name=False*, *return\_dict=False*)

Describe a component and its parameters

#### Parameters

- **print\_name** (*bool*, *optional*) – whether to print name of component
- **return\_dict** (*bool*, *optional*) – whether to return description as dictionary in the format {"name": name, "parameters": parameters}

**Returns** prints and returns dictionary

**Return type** None or dict

### **fit** (*self*, *X*, *y=None*)

Fits the DFSTransformer Transformer component.

#### Parameters

- **X** (*pd.DataFrame*, *np.array*) – The input data to transform, of shape [n\_samples, n\_features]
- **y** (*pd.Series*, *np.ndarray*, *optional*) – The target training data of length [n\_samples]

**Returns** self

### **fit\_transform** (*self*, *X*, *y=None*)

Fits on X and transforms X

#### Parameters

- **x** (*pd.DataFrame*) – Data to fit and transform
- **y** (*pd.Series*) – Target data

**Returns** Transformed X

**Return type** *pd.DataFrame*

**static load** (*file\_path*)

Loads component at file path

**Parameters** **file\_path** (*str*) – Location to load file

**Returns** *ComponentBase* object

**needs\_fitting** (*self*)

Returns boolean determining if component needs fitting before calling `predict`, `predict_proba`, `transform`, or `feature_importances`. This can be overridden to `False` for components that do not need to be fit or whose fit methods do nothing.

**property parameters** (*self*)

Returns the parameters which were used to initialize the component

**save** (*self, file\_path, pickle\_protocol=cloudpickle.DEFAULT\_PROTOCOL*)

Saves component at file path

**Parameters**

- **file\_path** (*str*) – Location to save file
- **pickle\_protocol** (*int*) – The pickle data stream format.

**Returns** `None`

**transform** (*self, X, y=None*)

Computes the feature matrix for the input X using featuretools' dfs algorithm.

**Parameters**

- **x** (*pd.DataFrame* or *np.ndarray*) – The input training data to transform. Has shape `[n_samples, n_features]`
- **y** (*pd.Series, optional*) – Ignored.

**Returns** Feature matrix

**Return type** *pd.DataFrame*

**class** `evalml.pipelines.components.transformers.preprocessing.DropNullColumns` (*pct\_null\_threshold=*  
*ran-*  
*dom\_seed=0,*  
*\*\*kwargs*)

Transformer to drop features whose percentage of NaN values exceeds a specified threshold.

**Parameters**

- **pct\_null\_threshold** (*float*) – The percentage of NaN values in an input feature to drop. Must be a value between `[0, 1]` inclusive. If equal to `0.0`, will drop columns with any null values. If equal to `1.0`, will drop columns with all null values. Defaults to `0.95`.
- **random\_seed** (*int*) – Seed for the random number generator. Defaults to `0`.

**Attributes**

<b>hyper-parameter_ranges</b>	{}
<b>model_family</b>	ModelFamily.NONE
<b>modifies_features</b>	True
<b>modifies_target</b>	False
<b>name</b>	Drop Null Columns Transformer

## Methods

<i>clone</i>	Constructs a new component with the same parameters and random state.
<i>default_parameters</i>	Returns the default parameters for this component.
<i>describe</i>	Describe a component and its parameters
<i>fit</i>	Fits component to data
<i>fit_transform</i>	Fits on X and transforms X
<i>load</i>	Loads component at file path
<i>needs_fitting</i>	Returns boolean determining if component needs fitting before
<i>parameters</i>	Returns the parameters which were used to initialize the component
<i>save</i>	Saves component at file path
<i>transform</i>	Transforms data X by dropping columns that exceed the threshold of null values.

### **clone** (*self*)

Constructs a new component with the same parameters and random state.

**Returns** A new instance of this component with identical parameters and random state.

### **default\_parameters** (*cls*)

Returns the default parameters for this component.

Our convention is that `Component.default_parameters == Component().parameters`.

**Returns** default parameters for this component.

**Return type** dict

### **describe** (*self*, *print\_name=False*, *return\_dict=False*)

Describe a component and its parameters

#### Parameters

- **print\_name** (*bool*, *optional*) – whether to print name of component
- **return\_dict** (*bool*, *optional*) – whether to return description as dictionary in the format {"name": name, "parameters": parameters}

**Returns** prints and returns dictionary

**Return type** None or dict

### **fit** (*self*, *X*, *y=None*)

Fits component to data

**Parameters**

- **X** (*list*, *pd.DataFrame* or *np.ndarray*) – The input training data of shape [n\_samples, n\_features]
- **y** (*list*, *pd.Series*, *np.ndarray*, *optional*) – The target training data of length [n\_samples]

**Returns** *self***fit\_transform** (*self*, *X*, *y=None*)

Fits on X and transforms X

**Parameters**

- **X** (*pd.DataFrame*) – Data to fit and transform
- **y** (*pd.Series*) – Target data

**Returns** Transformed X**Return type** *pd.DataFrame***static load** (*file\_path*)

Loads component at file path

**Parameters** **file\_path** (*str*) – Location to load file**Returns** ComponentBase object**needs\_fitting** (*self*)

Returns boolean determining if component needs fitting before calling predict, predict\_proba, transform, or feature\_importances. This can be overridden to False for components that do not need to be fit or whose fit methods do nothing.

**property parameters** (*self*)

Returns the parameters which were used to initialize the component

**save** (*self*, *file\_path*, *pickle\_protocol=cloudpickle.DEFAULT\_PROTOCOL*)

Saves component at file path

**Parameters**

- **file\_path** (*str*) – Location to save file
- **pickle\_protocol** (*int*) – The pickle data stream format.

**Returns** None**transform** (*self*, *X*, *y=None*)

Transforms data X by dropping columns that exceed the threshold of null values.

**Parameters**

- **X** (*pd.DataFrame*) – Data to transform
- **y** (*pd.Series*, *optional*) – Ignored.

**Returns** Transformed X**Return type** *pd.DataFrame*

**class** evalml.pipelines.components.transformers.preprocessing.**EmailFeaturizer** (*random\_seed=0*,  
\*\**kwargs*)

Transformer that can automatically extract features from emails.

**Parameters** **random\_seed** (*int*) – Seed for the random number generator. Defaults to 0.

### Attributes

<b>hyper-parameter_ranges</b>	{}
<b>model_family</b>	ModelFamily.NONE
<b>modifies_features</b>	True
<b>modifies_target</b>	False
<b>name</b>	Email Featurizer

### Methods

<i>clone</i>	Constructs a new component with the same parameters and random state.
<i>default_parameters</i>	Returns the default parameters for this component.
<i>describe</i>	Describe a component and its parameters
<i>fit</i>	Fits component to data
<i>fit_transform</i>	Fits on X and transforms X
<i>load</i>	Loads component at file path
<i>needs_fitting</i>	Returns boolean determining if component needs fitting before
<i>parameters</i>	Returns the parameters which were used to initialize the component
<i>save</i>	Saves component at file path
<i>transform</i>	Transforms data X.

#### **clone** (*self*)

Constructs a new component with the same parameters and random state.

**Returns** A new instance of this component with identical parameters and random state.

#### **default\_parameters** (*cls*)

Returns the default parameters for this component.

Our convention is that `Component.default_parameters == Component().parameters`.

**Returns** default parameters for this component.

**Return type** dict

#### **describe** (*self*, *print\_name=False*, *return\_dict=False*)

Describe a component and its parameters

##### Parameters

- **print\_name** (*bool*, *optional*) – whether to print name of component
- **return\_dict** (*bool*, *optional*) – whether to return description as dictionary in the format {"name": name, "parameters": parameters}

**Returns** prints and returns dictionary

**Return type** None or dict

#### **fit** (*self*, *X*, *y=None*)

Fits component to data

**Parameters**

- **X** (*list*, *pd.DataFrame* or *np.ndarray*) – The input training data of shape [n\_samples, n\_features]
- **y** (*list*, *pd.Series*, *np.ndarray*, *optional*) – The target training data of length [n\_samples]

**Returns** *self***fit\_transform** (*self*, *X*, *y=None*)

Fits on X and transforms X

**Parameters**

- **X** (*pd.DataFrame*) – Data to fit and transform
- **y** (*pd.Series*) – Target data

**Returns** Transformed X**Return type** *pd.DataFrame***static load** (*file\_path*)

Loads component at file path

**Parameters** **file\_path** (*str*) – Location to load file**Returns** ComponentBase object**needs\_fitting** (*self*)

Returns boolean determining if component needs fitting before calling predict, predict\_proba, transform, or feature\_importances. This can be overridden to False for components that do not need to be fit or whose fit methods do nothing.

**property parameters** (*self*)

Returns the parameters which were used to initialize the component

**save** (*self*, *file\_path*, *pickle\_protocol=cloudpickle.DEFAULT\_PROTOCOL*)

Saves component at file path

**Parameters**

- **file\_path** (*str*) – Location to save file
- **pickle\_protocol** (*int*) – The pickle data stream format.

**Returns** None**transform** (*self*, *X*, *y=None*)

Transforms data X.

**Parameters**

- **X** (*pd.DataFrame*) – Data to transform.
- **y** (*pd.Series*, *optional*) – Target data.

**Returns** Transformed X**Return type** *pd.DataFrame***class** evalml.pipelines.components.transformers.preprocessing.**LogTransformer** (*random\_seed=0*)

Applies a log transformation to the target data.

**Attributes**

<b>hyper-parameter_ranges</b>	{}
<b>model_family</b>	ModelFamily.NONE
<b>modifies_features</b>	False
<b>modifies_target</b>	True
<b>name</b>	Log Transformer

## Methods

<i>clone</i>	Constructs a new component with the same parameters and random state.
<i>default_parameters</i>	Returns the default parameters for this component.
<i>describe</i>	Describe a component and its parameters
<i>fit</i>	Fits the LogTransformer.
<i>fit_transform</i>	Log transforms the target variable.
<i>inverse_transform</i>	Inverts the transformation done by the transform method.
<i>load</i>	Loads component at file path
<i>needs_fitting</i>	Returns boolean determining if component needs fitting before
<i>parameters</i>	Returns the parameters which were used to initialize the component
<i>save</i>	Saves component at file path
<i>transform</i>	Log transforms the target variable.

### **clone** (*self*)

Constructs a new component with the same parameters and random state.

**Returns** A new instance of this component with identical parameters and random state.

### **default\_parameters** (*cls*)

Returns the default parameters for this component.

Our convention is that `Component.default_parameters == Component().parameters`.

**Returns** default parameters for this component.

**Return type** dict

### **describe** (*self*, *print\_name=False*, *return\_dict=False*)

Describe a component and its parameters

#### Parameters

- **print\_name** (*bool*, *optional*) – whether to print name of component
- **return\_dict** (*bool*, *optional*) – whether to return description as dictionary in the format {"name": name, "parameters": parameters}

**Returns** prints and returns dictionary

**Return type** None or dict

### **fit** (*self*, *X*, *y=None*)

Fits the LogTransformer.



**Parameters**

- **X** (*pd.DataFrame* or *np.ndarray*) – Ignored.
- **y** (*pd.Series*, *optional*) – Ignored.

**Returns** *self***fit\_transform** (*self*, *X*, *y=None*)

Log transforms the target variable.

**Parameters**

- **X** (*pd.DataFrame*, *optional*) – Ignored.
- **y** (*pd.Series*) – Target variable to log transform.

**Returns****The input features are returned without modification. The target** variable *y* is log transformed.**Return type** tuple of *pd.DataFrame*, *pd.Series***inverse\_transform** (*self*, *y*)

Inverts the transformation done by the transform method.

**Arguments:** *y* (*pd.Series*): Target transformed by this component.**Returns** Target without the transformation.**Return type** *pd.Series***static load** (*file\_path*)

Loads component at file path

**Parameters** **file\_path** (*str*) – Location to load file**Returns** *ComponentBase* object**needs\_fitting** (*self*)

Returns boolean determining if component needs fitting before calling predict, predict\_proba, transform, or feature\_importances. This can be overridden to False for components that do not need to be fit or whose fit methods do nothing.

**property parameters** (*self*)

Returns the parameters which were used to initialize the component

**save** (*self*, *file\_path*, *pickle\_protocol=cloudpickle.DEFAULT\_PROTOCOL*)

Saves component at file path

**Parameters**

- **file\_path** (*str*) – Location to save file
- **pickle\_protocol** (*int*) – The pickle data stream format.

**Returns** *None***transform** (*self*, *X*, *y=None*)

Log transforms the target variable.

**Parameters**

- **X** (*pd.DataFrame*, *optional*) – Ignored.
- **y** (*pd.Series*) – Target data to log transform.

**Returns**

The input features are returned without modification. The target variable `y` is log transformed.

**Return type** tuple of `pd.DataFrame`, `pd.Series`

**class** `evalml.pipelines.components.transformers.preprocessing.LSA` (*random\_seed=0*,  
\*\**kwargs*)

Transformer to calculate the Latent Semantic Analysis Values of text input.

**Parameters** `random_seed` (*int*) – Seed for the random number generator. Defaults to 0.

**Attributes**

<b>hyper-parameter_ranges</b>	{}
<b>model_family</b>	ModelFamily.NONE
<b>modifies_features</b>	True
<b>modifies_target</b>	False
<b>name</b>	LSA Transformer

**Methods**

<code>clone</code>	Constructs a new component with the same parameters and random state.
<code>default_parameters</code>	Returns the default parameters for this component.
<code>describe</code>	Describe a component and its parameters
<code>fit</code>	Fits component to data
<code>fit_transform</code>	Fits on <code>X</code> and transforms <code>X</code>
<code>load</code>	Loads component at file path
<code>needs_fitting</code>	Returns boolean determining if component needs fitting before
<code>parameters</code>	Returns the parameters which were used to initialize the component
<code>save</code>	Saves component at file path
<code>transform</code>	Transforms data <code>X</code> by applying the LSA pipeline.

**clone** (*self*)

Constructs a new component with the same parameters and random state.

**Returns** A new instance of this component with identical parameters and random state.

**default\_parameters** (*cls*)

Returns the default parameters for this component.

Our convention is that `Component.default_parameters == Component().parameters`.

**Returns** default parameters for this component.

**Return type** dict

**describe** (*self*, *print\_name=False*, *return\_dict=False*)

Describe a component and its parameters

**Parameters**

- **print\_name** (*bool, optional*) – whether to print name of component
- **return\_dict** (*bool, optional*) – whether to return description as dictionary in the format {"name": name, "parameters": parameters}

**Returns** prints and returns dictionary

**Return type** None or dict

**fit** (*self, X, y=None*)

Fits component to data

**Parameters**

- **X** (*list, pd.DataFrame or np.ndarray*) – The input training data of shape [n\_samples, n\_features]
- **y** (*list, pd.Series, np.ndarray, optional*) – The target training data of length [n\_samples]

**Returns** self

**fit\_transform** (*self, X, y=None*)

Fits on X and transforms X

**Parameters**

- **X** (*pd.DataFrame*) – Data to fit and transform
- **y** (*pd.Series*) – Target data

**Returns** Transformed X

**Return type** pd.DataFrame

**static load** (*file\_path*)

Loads component at file path

**Parameters** **file\_path** (*str*) – Location to load file

**Returns** ComponentBase object

**needs\_fitting** (*self*)

Returns boolean determining if component needs fitting before calling predict, predict\_proba, transform, or feature\_importances. This can be overridden to False for components that do not need to be fit or whose fit methods do nothing.

**property parameters** (*self*)

Returns the parameters which were used to initialize the component

**save** (*self, file\_path, pickle\_protocol=cloudpickle.DEFAULT\_PROTOCOL*)

Saves component at file path

**Parameters**

- **file\_path** (*str*) – Location to save file
- **pickle\_protocol** (*int*) – The pickle data stream format.

**Returns** None

**transform** (*self, X, y=None*)

Transforms data X by applying the LSA pipeline.

**Parameters**

- **X** (*pd.DataFrame*) – The data to transform.

- **y** (*pd.Series*, *optional*) – Ignored.

#### Returns

**Transformed X.** The original column is removed and replaced with two columns of the format `LSA(original_column_name)[feature_number]`, where `feature_number` is 0 or 1.

**Return type** `pd.DataFrame`

**class** `evalml.pipelines.components.transformers.preprocessing.PolynomialDetrender` (*degree=1*, *random\_seed=0*, *\*\*kwargs*)

Removes trends from time series by fitting a polynomial to the data.

#### Parameters

- **degree** (*int*) – Degree for the polynomial. If 1, linear model is fit to the data. If 2, quadratic model is fit, etc. Defaults to 1.
- **random\_seed** (*int*) – Seed for the random number generator. Defaults to 0.

#### Attributes

<b>hyper-parameter_ranges</b>	{ “degree”: Integer(1, 3)}
<b>model_family</b>	ModelFamily.NONE
<b>modifies_features</b>	False
<b>modifies_target</b>	True
<b>name</b>	Polynomial Detrender

#### Methods

<code>clone</code>	Constructs a new component with the same parameters and random state.
<code>default_parameters</code>	Returns the default parameters for this component.
<code>describe</code>	Describe a component and its parameters
<code>fit</code>	Fits the PolynomialDetrender.
<code>fit_transform</code>	Removes fitted trend from target variable.
<code>inverse_transform</code>	Adds back fitted trend to target variable.
<code>load</code>	Loads component at file path
<code>needs_fitting</code>	Returns boolean determining if component needs fitting before
<code>parameters</code>	Returns the parameters which were used to initialize the component
<code>save</code>	Saves component at file path
<code>transform</code>	Removes fitted trend from target variable.

**clone** (*self*)

Constructs a new component with the same parameters and random state.

**Returns** A new instance of this component with identical parameters and random state.

**default\_parameters** (*cls*)

Returns the default parameters for this component.

Our convention is that `Component.default_parameters == Component().parameters`.

**Returns** default parameters for this component.

**Return type** dict

**describe** (*self*, *print\_name=False*, *return\_dict=False*)

Describe a component and its parameters

**Parameters**

- **print\_name** (*bool*, *optional*) – whether to print name of component
- **return\_dict** (*bool*, *optional*) – whether to return description as dictionary in the format {"name": name, "parameters": parameters}

**Returns** prints and returns dictionary

**Return type** None or dict

**fit** (*self*, *X*, *y=None*)

Fits the PolynomialDetrender.

**Parameters**

- **X** (*pd.DataFrame*, *optional*) – Ignored.
- **y** (*pd.Series*) – Target variable to detrend.

**Returns** self

**fit\_transform** (*self*, *X*, *y=None*)

Removes fitted trend from target variable.

**Parameters**

- **X** (*pd.DataFrame*, *optional*) – Ignored.
- **y** (*pd.Series*) – Target variable to detrend.

**Returns**

**The first element are the input features returned without modification.** The second element is the target variable y with the fitted trend removed.

**Return type** tuple of *pd.DataFrame*, *pd.Series*

**inverse\_transform** (*self*, *y*)

Adds back fitted trend to target variable.

**Parameters**

- **X** (*pd.DataFrame*, *optional*) – Ignored.
- **y** (*pd.Series*) – Target variable.

**Returns**

**The first element are the input features returned without modification.** The second element is the target variable y with the trend added back.

**Return type** tuple of *pd.DataFrame*, *pd.Series*

**static load** (*file\_path*)

Loads component at file path

**Parameters** **file\_path** (*str*) – Location to load file

**Returns** ComponentBase object

**needs\_fitting** (*self*)

Returns boolean determining if component needs fitting before calling predict, predict\_proba, transform, or feature\_importances. This can be overridden to False for components that do not need to be fit or whose fit methods do nothing.

**property parameters** (*self*)

Returns the parameters which were used to initialize the component

**save** (*self*, *file\_path*, *pickle\_protocol*=cloudpickle.DEFAULT\_PROTOCOL)

Saves component at file path

**Parameters**

- **file\_path** (*str*) – Location to save file
- **pickle\_protocol** (*int*) – The pickle data stream format.

**Returns** None

**transform** (*self*, *X*, *y*=None)

Removes fitted trend from target variable.

**Parameters**

- **X** (*pd.DataFrame*, *optional*) – Ignored.
- **y** (*pd.Series*) – Target variable to detrend.

**Returns**

**The input features are returned without modification. The target** variable *y* is detrended

**Return type** tuple of *pd.DataFrame*, *pd.Series*

**class** evalml.pipelines.components.transformers.preprocessing.**TextFeaturizer** (*random\_seed*=0, *\*\*kwargs*)

Transformer that can automatically featurize text columns using featuretools' nlp\_primitives.

Since models cannot handle non-numeric data, any text must be broken down into features that provide useful information about that text. This component splits each text column into several informative features: Diversity Score, Mean Characters per Word, Polarity Score, and LSA (Latent Semantic Analysis). Calling transform on this component will replace any text columns in the given dataset with these numeric columns.

**Parameters** **random\_seed** (*int*) – Seed for the random number generator. Defaults to 0.

**Attributes**

hyper-parameter_ranges	{}
model_family	ModelFamily.NONE
modifies_features	True
modifies_target	False
name	Text Featurization Component

**Methods**

<code>clone</code>	Constructs a new component with the same parameters and random state.
<code>default_parameters</code>	Returns the default parameters for this component.
<code>describe</code>	Describe a component and its parameters
<code>fit</code>	Fits component to data
<code>fit_transform</code>	Fits on X and transforms X
<code>load</code>	Loads component at file path
<code>needs_fitting</code>	Returns boolean determining if component needs fitting before
<code>parameters</code>	Returns the parameters which were used to initialize the component
<code>save</code>	Saves component at file path
<code>transform</code>	Transforms data X by creating new features using existing text columns

**clone** (*self*)

Constructs a new component with the same parameters and random state.

**Returns** A new instance of this component with identical parameters and random state.

**default\_parameters** (*cls*)

Returns the default parameters for this component.

Our convention is that `Component.default_parameters == Component().parameters`.

**Returns** default parameters for this component.

**Return type** dict

**describe** (*self*, *print\_name=False*, *return\_dict=False*)

Describe a component and its parameters

**Parameters**

- **print\_name** (*bool*, *optional*) – whether to print name of component
- **return\_dict** (*bool*, *optional*) – whether to return description as dictionary in the format {"name": name, "parameters": parameters}

**Returns** prints and returns dictionary

**Return type** None or dict

**fit** (*self*, *X*, *y=None*)

Fits component to data

**Parameters**

- **X** (*pd.DataFrame* or *np.ndarray*) – The input training data of shape [n\_samples, n\_features]
- **y** (*pd.Series*, *np.ndarray*, *optional*) – The target training data of length [n\_samples]

**Returns** self

**fit\_transform** (*self*, *X*, *y=None*)

Fits on X and transforms X

**Parameters**

- **X** (*pd.DataFrame*) – Data to fit and transform

- **y** (*pd.Series*) – Target data

**Returns** Transformed X

**Return type** *pd.DataFrame*

**static load** (*file\_path*)

Loads component at file path

**Parameters** **file\_path** (*str*) – Location to load file

**Returns** *ComponentBase* object

**needs\_fitting** (*self*)

Returns boolean determining if component needs fitting before calling `predict`, `predict_proba`, `transform`, or `feature_importances`. This can be overridden to `False` for components that do not need to be fit or whose fit methods do nothing.

**property parameters** (*self*)

Returns the parameters which were used to initialize the component

**save** (*self*, *file\_path*, *pickle\_protocol=cloudpickle.DEFAULT\_PROTOCOL*)

Saves component at file path

**Parameters**

- **file\_path** (*str*) – Location to save file
- **pickle\_protocol** (*int*) – The pickle data stream format.

**Returns** `None`

**transform** (*self*, *X*, *y=None*)

Transforms data *X* by creating new features using existing text columns

**Parameters**

- **X** (*pd.DataFrame*) – The data to transform.
- **y** (*pd.Series*, *optional*) – Ignored.

**Returns** Transformed X

**Return type** *pd.DataFrame*

**class** `evalml.pipelines.components.transformers.preprocessing.TextTransformer` (*component\_obj=None*, *random\_seed=0*, *\*\*kwargs*)

Base class for all transformers working with text features.

**Parameters**

- **component\_obj** (*obj*) – Third-party objects useful in component implementation. Defaults to `None`.
- **random\_seed** (*int*) – Seed for the random number generator. Defaults to 0.

**Attributes**

<b>model_family</b>	<code>ModelFamily.NONE</code>
<b>modifies_features</b>	<code>True</code>
<b>modifies_target</b>	<code>False</code>



**Methods**

<code>clone</code>	Constructs a new component with the same parameters and random state.
<code>default_parameters</code>	Returns the default parameters for this component.
<code>describe</code>	Describe a component and its parameters
<code>fit</code>	Fits component to data
<code>fit_transform</code>	Fits on X and transforms X
<code>load</code>	Loads component at file path
<code>name</code>	Returns string name of this component
<code>needs_fitting</code>	Returns boolean determining if component needs fitting before
<code>parameters</code>	Returns the parameters which were used to initialize the component
<code>save</code>	Saves component at file path
<code>transform</code>	Transforms data X.

**clone** (*self*)

Constructs a new component with the same parameters and random state.

**Returns** A new instance of this component with identical parameters and random state.

**default\_parameters** (*cls*)

Returns the default parameters for this component.

Our convention is that `Component.default_parameters == Component().parameters`.

**Returns** default parameters for this component.

**Return type** dict

**describe** (*self*, *print\_name=False*, *return\_dict=False*)

Describe a component and its parameters

**Parameters**

- **print\_name** (*bool*, *optional*) – whether to print name of component
- **return\_dict** (*bool*, *optional*) – whether to return description as dictionary in the format {"name": name, "parameters": parameters}

**Returns** prints and returns dictionary

**Return type** None or dict

**fit** (*self*, *X*, *y=None*)

Fits component to data

**Parameters**

- **X** (*list*, *pd.DataFrame* or *np.ndarray*) – The input training data of shape [n\_samples, n\_features]
- **y** (*list*, *pd.Series*, *np.ndarray*, *optional*) – The target training data of length [n\_samples]

**Returns** self

**fit\_transform** (*self*, *X*, *y=None*)

Fits on X and transforms X

**Parameters**

- **x** (*pd.DataFrame*) – Data to fit and transform
- **y** (*pd.Series*) – Target data

**Returns** Transformed X

**Return type** *pd.DataFrame*

**static load** (*file\_path*)

Loads component at file path

**Parameters** **file\_path** (*str*) – Location to load file

**Returns** ComponentBase object

**property name** (*cls*)

Returns string name of this component

**needs\_fitting** (*self*)

Returns boolean determining if component needs fitting before calling predict, predict\_proba, transform, or feature\_importances. This can be overridden to False for components that do not need to be fit or whose fit methods do nothing.

**property parameters** (*self*)

Returns the parameters which were used to initialize the component

**save** (*self, file\_path, pickle\_protocol=cloudpickle.DEFAULT\_PROTOCOL*)

Saves component at file path

**Parameters**

- **file\_path** (*str*) – Location to save file
- **pickle\_protocol** (*int*) – The pickle data stream format.

**Returns** None

**transform** (*self, X, y=None*)

Transforms data X.

**Parameters**

- **x** (*pd.DataFrame*) – Data to transform.
- **y** (*pd.Series, optional*) – Target data.

**Returns** Transformed X

**Return type** *pd.DataFrame*

**class** evalml.pipelines.components.transformers.preprocessing.**URLFeaturizer** (*random\_seed=0, \*\*kwargs*)

Transformer that can automatically extract features from URL.

**Parameters** **random\_seed** (*int*) – Seed for the random number generator. Defaults to 0.

**Attributes**

<b>hyper-parameter_ranges</b>	{}
<b>model_family</b>	ModelFamily.NONE
<b>modifies_features</b>	True
<b>modifies_target</b>	False
<b>name</b>	URL Featurizer

## Methods

<i>clone</i>	Constructs a new component with the same parameters and random state.
<i>default_parameters</i>	Returns the default parameters for this component.
<i>describe</i>	Describe a component and its parameters
<i>fit</i>	Fits component to data
<i>fit_transform</i>	Fits on X and transforms X
<i>load</i>	Loads component at file path
<i>needs_fitting</i>	Returns boolean determining if component needs fitting before
<i>parameters</i>	Returns the parameters which were used to initialize the component
<i>save</i>	Saves component at file path
<i>transform</i>	Transforms data X.

### **clone** (*self*)

Constructs a new component with the same parameters and random state.

**Returns** A new instance of this component with identical parameters and random state.

### **default\_parameters** (*cls*)

Returns the default parameters for this component.

Our convention is that `Component.default_parameters == Component().parameters`.

**Returns** default parameters for this component.

**Return type** dict

### **describe** (*self*, *print\_name=False*, *return\_dict=False*)

Describe a component and its parameters

#### Parameters

- **print\_name** (*bool*, *optional*) – whether to print name of component
- **return\_dict** (*bool*, *optional*) – whether to return description as dictionary in the format {"name": name, "parameters": parameters}

**Returns** prints and returns dictionary

**Return type** None or dict

### **fit** (*self*, *X*, *y=None*)

Fits component to data

#### Parameters

- **X** (*list*, *pd.DataFrame* or *np.ndarray*) – The input training data of shape [n\_samples, n\_features]
- **y** (*list*, *pd.Series*, *np.ndarray*, *optional*) – The target training data of length [n\_samples]

**Returns** self

**fit\_transform** (*self*, *X*, *y=None*)

Fits on X and transforms X

**Parameters**

- **X** (*pd.DataFrame*) – Data to fit and transform
- **y** (*pd.Series*) – Target data

**Returns** Transformed X

**Return type** *pd.DataFrame*

**static load** (*file\_path*)

Loads component at file path

**Parameters** **file\_path** (*str*) – Location to load file

**Returns** ComponentBase object

**needs\_fitting** (*self*)

Returns boolean determining if component needs fitting before calling predict, predict\_proba, transform, or feature\_importances. This can be overridden to False for components that do not need to be fit or whose fit methods do nothing.

**property parameters** (*self*)

Returns the parameters which were used to initialize the component

**save** (*self*, *file\_path*, *pickle\_protocol=cloudpickle.DEFAULT\_PROTOCOL*)

Saves component at file path

**Parameters**

- **file\_path** (*str*) – Location to save file
- **pickle\_protocol** (*int*) – The pickle data stream format.

**Returns** None

**transform** (*self*, *X*, *y=None*)

Transforms data X.

**Parameters**

- **X** (*pd.DataFrame*) – Data to transform.
- **y** (*pd.Series*, *optional*) – Target data.

**Returns** Transformed X

**Return type** *pd.DataFrame*

samplers

Submodules

base\_sampler

Module Contents

Classes Summary

<i>BaseOversampler</i>	Base Oversampler component. Used as the base class of all imbalance-learn oversampler components.
<i>BaseSampler</i>	Base Sampler component. Used as the base class of all sampler components.

Contents

**class** evalml.pipelines.components.transformers.samplers.base\_sampler.**BaseOversampler** (*sampler*, *sampling\_ratio*, *sampling\_ratio\_dict*, *k\_neighbors\_default*, *n\_jobs*, *random\_seed*, *\*\*kwargs*)

Base Oversampler component. Used as the base class of all imbalance-learn oversampler components.

Parameters

- **sampler** (*obj*) – Sampler object to use.
- **sampling\_ratio** (*float*) – This is the goal ratio of the minority to majority class, with range (0, 1]. A value of 0.25 means we want a 1:4 ratio of the minority to majority class after oversampling. We will create the a sampling dictionary using this ratio, with the keys corresponding to the class and the values responding to the number of samples. Defaults to 0.25.
- **sampling\_ratio\_dict** (*dict*) – A dictionary specifying the desired balanced ratio for each target value. For instance, in a binary case where class 1 is the minority, we could specify: *sampling\_ratio\_dict*={0: 0.5, 1: 1}, which means we would undersample class 0 to have twice the number of samples as class 1 (minority:majority ratio = 0.5), and don't sample class 1. Overrides *sampling\_ratio* if provided. Defaults to None.
- **k\_neighbors\_default** (*int*) – The number of nearest neighbors used to construct synthetic samples. This is the default value used, but the actual *k\_neighbors* value might be smaller if there are less samples. Defaults to 5.
- **n\_jobs** (*int*) – The number of CPU cores to use. Defaults to -1.
- **random\_seed** (*int*) – The seed to use for random sampling. Defaults to 0.

## Attributes

<b>model_family</b>	ModelFamily.NONE
<b>modifies_features</b>	True
<b>modifies_target</b>	True

## Methods

<i>clone</i>	Constructs a new component with the same parameters and random state.
<i>default_parameters</i>	Returns the default parameters for this component.
<i>describe</i>	Describe a component and its parameters
<i>fit</i>	Fits the sampler to the data.
<i>fit_transform</i>	Fits on X and transforms X
<i>load</i>	Loads component at file path
<i>name</i>	Returns string name of this component
<i>needs_fitting</i>	Returns boolean determining if component needs fitting before
<i>parameters</i>	Returns the parameters which were used to initialize the component
<i>save</i>	Saves component at file path
<i>transform</i>	Transforms the input data by sampling the data.

### **clone** (*self*)

Constructs a new component with the same parameters and random state.

**Returns** A new instance of this component with identical parameters and random state.

### **default\_parameters** (*cls*)

Returns the default parameters for this component.

Our convention is that `Component.default_parameters == Component().parameters`.

**Returns** default parameters for this component.

**Return type** dict

### **describe** (*self*, *print\_name=False*, *return\_dict=False*)

Describe a component and its parameters

#### Parameters

- **print\_name** (*bool*, *optional*) – whether to print name of component
- **return\_dict** (*bool*, *optional*) – whether to return description as dictionary in the format {"name": name, "parameters": parameters}

**Returns** prints and returns dictionary

**Return type** None or dict

### **fit** (*self*, *X*, *y*)

Fits the sampler to the data.

#### Parameters

- **X** (*pd.DataFrame*) – Input features.

- **y** (*pd.Series*) – Target.

**Returns** self

**fit\_transform** (*self*, *X*, *y*)

Fits on *X* and transforms *X*

**Parameters**

- **x** (*pd.DataFrame*) – Data to fit and transform
- **y** (*pd.Series*) – Target data

**Returns** Transformed *X*

**Return type** *pd.DataFrame*

**static load** (*file\_path*)

Loads component at file path

**Parameters** **file\_path** (*str*) – Location to load file

**Returns** ComponentBase object

**property name** (*cls*)

Returns string name of this component

**needs\_fitting** (*self*)

Returns boolean determining if component needs fitting before calling predict, predict\_proba, transform, or feature\_importances. This can be overridden to False for components that do not need to be fit or whose fit methods do nothing.

**property parameters** (*self*)

Returns the parameters which were used to initialize the component

**save** (*self*, *file\_path*, *pickle\_protocol=cloudpickle.DEFAULT\_PROTOCOL*)

Saves component at file path

**Parameters**

- **file\_path** (*str*) – Location to save file
- **pickle\_protocol** (*int*) – The pickle data stream format.

**Returns** None

**transform** (*self*, *X*, *y=None*)

Transforms the input data by sampling the data.

**Parameters**

- **x** (*pd.DataFrame*) – Training features.
- **y** (*pd.Series*) – Target.

**Returns** Transformed features and target.

**Return type** *pd.DataFrame*, *pd.Series*

**class** evalml.pipelines.components.transformers.samplers.base\_sampler.**BaseSampler** (*parameters=Non*  
*com-*  
*po-*  
*nent\_obj=None*  
*ran-*  
*dom\_seed=0*,  
*\*\*kwargs*)

Base Sampler component. Used as the base class of all sampler components.

### Parameters

- **parameters** (*dict*) – Dictionary of parameters for the component. Defaults to None.
- **component\_obj** (*obj*) – Third-party objects useful in component implementation. Defaults to None.
- **random\_seed** (*int*) – Seed for the random number generator. Defaults to 0.

### Attributes

<b>model_family</b>	ModelFamily.NONE
<b>modifies_features</b>	True
<b>modifies_target</b>	True

### Methods

<i>clone</i>	Constructs a new component with the same parameters and random state.
<i>default_parameters</i>	Returns the default parameters for this component.
<i>describe</i>	Describe a component and its parameters
<i>fit</i>	Fits the sampler to the data.
<i>fit_transform</i>	Fits on X and transforms X
<i>load</i>	Loads component at file path
<i>name</i>	Returns string name of this component
<i>needs_fitting</i>	Returns boolean determining if component needs fitting before
<i>parameters</i>	Returns the parameters which were used to initialize the component
<i>save</i>	Saves component at file path
<i>transform</i>	Transforms the input data by sampling the data.

#### **clone** (*self*)

Constructs a new component with the same parameters and random state.

**Returns** A new instance of this component with identical parameters and random state.

#### **default\_parameters** (*cls*)

Returns the default parameters for this component.

Our convention is that `Component.default_parameters == Component().parameters`.

**Returns** default parameters for this component.

**Return type** dict

#### **describe** (*self*, *print\_name=False*, *return\_dict=False*)

Describe a component and its parameters

#### Parameters

- **print\_name** (*bool*, *optional*) – whether to print name of component
- **return\_dict** (*bool*, *optional*) – whether to return description as dictionary in the format {"name": name, "parameters": parameters}

**Returns** prints and returns dictionary



**Return type** None or dict

**fit** (*self*, *X*, *y*)

Fits the sampler to the data.

**Parameters**

- **X** (*pd.DataFrame*) – Input features.
- **y** (*pd.Series*) – Target.

**Returns** *self*

**fit\_transform** (*self*, *X*, *y*)

Fits on X and transforms X

**Parameters**

- **X** (*pd.DataFrame*) – Data to fit and transform
- **y** (*pd.Series*) – Target data

**Returns** Transformed X

**Return type** *pd.DataFrame*

**static load** (*file\_path*)

Loads component at file path

**Parameters** **file\_path** (*str*) – Location to load file

**Returns** ComponentBase object

**property name** (*cls*)

Returns string name of this component

**needs\_fitting** (*self*)

Returns boolean determining if component needs fitting before calling predict, predict\_proba, transform, or feature\_importances. This can be overridden to False for components that do not need to be fit or whose fit methods do nothing.

**property parameters** (*self*)

Returns the parameters which were used to initialize the component

**save** (*self*, *file\_path*, *pickle\_protocol=cloudpickle.DEFAULT\_PROTOCOL*)

Saves component at file path

**Parameters**

- **file\_path** (*str*) – Location to save file
- **pickle\_protocol** (*int*) – The pickle data stream format.

**Returns** None

**transform** (*self*, *X*, *y=None*)

Transforms the input data by sampling the data.

**Parameters**

- **X** (*pd.DataFrame*) – Training features.
- **y** (*pd.Series*) – Target.

**Returns** Transformed features and target.

**Return type** *pd.DataFrame*, *pd.Series*

oversamplers

Module Contents

Classes Summary

<i>SMOTENCOverSampler</i>	SMOTENC Oversampler component. Uses SMOTENC to generate synthetic samples. Works on a mix of numerical and categorical columns.
<i>SMOTENOverSampler</i>	SMOTEN Oversampler component. Uses SMOTEN to generate synthetic samples. Works for purely categorical datasets.
<i>SMOTEOverSampler</i>	SMOTE Oversampler component. Works on numerical datasets only. This component is only run during training and not during predict.

Contents

**class** evalml.pipelines.components.transformers.samplers.oversamplers.**SMOTENCOverSampler** (*sampling\_ratio*, *k\_neighbors\_default*, *n\_jobs*, *random\_seed*)

SMOTENC Oversampler component. Uses SMOTENC to generate synthetic samples. Works on a mix of numerical and categorical columns. Input data must be Woodwork type, and this component is only run during training and not during predict.

Parameters

- **sampling\_ratio** (*float*) – This is the goal ratio of the minority to majority class, with range (0, 1]. A value of 0.25 means we want a 1:4 ratio of the minority to majority class after oversampling. We will create the a sampling dictionary using this ratio, with the keys corresponding to the class and the values responding to the number of samples. Defaults to 0.25.
- **k\_neighbors\_default** (*int*) – The number of nearest neighbors used to construct synthetic samples. This is the default value used, but the actual k\_neighbors value might be smaller if there are less samples. Defaults to 5.
- **n\_jobs** (*int*) – The number of CPU cores to use. Defaults to -1.
- **random\_seed** (*int*) – The seed to use for random sampling. Defaults to 0.

Attributes

<b>hyper-parameter_ranges</b>	{}
<b>model_family</b>	ModelFamily.NONE
<b>modifies_features</b>	True
<b>modifies_target</b>	True
<b>name</b>	SMOTENC Oversampler

## Methods

<i>clone</i>	Constructs a new component with the same parameters and random state.
<i>default_parameters</i>	Returns the default parameters for this component.
<i>describe</i>	Describe a component and its parameters
<i>fit</i>	Fits the sampler to the data.
<i>fit_transform</i>	Fits on X and transforms X
<i>load</i>	Loads component at file path
<i>needs_fitting</i>	Returns boolean determining if component needs fitting before
<i>parameters</i>	Returns the parameters which were used to initialize the component
<i>save</i>	Saves component at file path
<i>transform</i>	Transforms the input data by sampling the data.

### **clone** (*self*)

Constructs a new component with the same parameters and random state.

**Returns** A new instance of this component with identical parameters and random state.

### **default\_parameters** (*cls*)

Returns the default parameters for this component.

Our convention is that `Component.default_parameters == Component().parameters`.

**Returns** default parameters for this component.

**Return type** dict

### **describe** (*self*, *print\_name=False*, *return\_dict=False*)

Describe a component and its parameters

#### Parameters

- **print\_name** (*bool*, *optional*) – whether to print name of component
- **return\_dict** (*bool*, *optional*) – whether to return description as dictionary in the format {"name": name, "parameters": parameters}

**Returns** prints and returns dictionary

**Return type** None or dict

### **fit** (*self*, *X*, *y*)

Fits the sampler to the data.

#### Parameters

- **X** (*pd.DataFrame*) – Input features.
- **y** (*pd.Series*) – Target.

**Returns** *self*

**fit\_transform** (*self*, *X*, *y*)  
Fits on *X* and transforms *X*

**Parameters**

- **X** (*pd.DataFrame*) – Data to fit and transform
- **y** (*pd.Series*) – Target data

**Returns** Transformed *X*

**Return type** *pd.DataFrame*

**static load** (*file\_path*)  
Loads component at file path

**Parameters** **file\_path** (*str*) – Location to load file

**Returns** *ComponentBase* object

**needs\_fitting** (*self*)  
Returns boolean determining if component needs fitting before calling *predict*, *predict\_proba*, *transform*, or *feature\_importances*. This can be overridden to *False* for components that do not need to be fit or whose fit methods do nothing.

**property parameters** (*self*)  
Returns the parameters which were used to initialize the component

**save** (*self*, *file\_path*, *pickle\_protocol=cloudpickle.DEFAULT\_PROTOCOL*)  
Saves component at file path

**Parameters**

- **file\_path** (*str*) – Location to save file
- **pickle\_protocol** (*int*) – The pickle data stream format.

**Returns** *None*

**transform** (*self*, *X*, *y=None*)  
Transforms the input data by sampling the data.

**Parameters**

- **X** (*pd.DataFrame*) – Training features.
- **y** (*pd.Series*) – Target.

**Returns** Transformed features and target.

**Return type** *pd.DataFrame*, *pd.Series*

**class** evalml.pipelines.components.transformers.samplers.oversamplers.**SMOTENOverSampler** (*samp*  
*k\_ne*  
*n\_jo*  
  
*l*,  
*ran*-  
*dom*-  
*\*\*kw*

SMOTEN Oversampler component. Uses SMOTEN to generate synthetic samples. Works for purely categorical

datasets. This component is only run during training and not during predict.

### Parameters

- **sampling\_ratio** (*float*) – This is the goal ratio of the minority to majority class, with range (0, 1]. A value of 0.25 means we want a 1:4 ratio of the minority to majority class after oversampling. We will create the a sampling dictionary using this ratio, with the keys corresponding to the class and the values responding to the number of samples. Defaults to 0.25.
- **k\_neighbors\_default** (*int*) – The number of nearest neighbors used to construct synthetic samples. This is the default value used, but the actual k\_neighbors value might be smaller if there are less samples. Defaults to 5.
- **n\_jobs** (*int*) – The number of CPU cores to use. Defaults to -1.
- **random\_seed** (*int*) – The seed to use for random sampling. Defaults to 0.

### Attributes

<b>hyper-parameter_ranges</b>	{}
<b>model_family</b>	ModelFamily.NONE
<b>modifies_features</b>	True
<b>modifies_target</b>	True
<b>name</b>	SMOTEN Oversampler

### Methods

<i>clone</i>	Constructs a new component with the same parameters and random state.
<i>default_parameters</i>	Returns the default parameters for this component.
<i>describe</i>	Describe a component and its parameters
<i>fit</i>	Fits the sampler to the data.
<i>fit_transform</i>	Fits on X and transforms X
<i>load</i>	Loads component at file path
<i>needs_fitting</i>	Returns boolean determining if component needs fitting before
<i>parameters</i>	Returns the parameters which were used to initialize the component
<i>save</i>	Saves component at file path
<i>transform</i>	Transforms the input data by sampling the data.

#### **clone** (*self*)

Constructs a new component with the same parameters and random state.

**Returns** A new instance of this component with identical parameters and random state.

#### **default\_parameters** (*cls*)

Returns the default parameters for this component.

Our convention is that `Component.default_parameters == Component().parameters`.

**Returns** default parameters for this component.

**Return type** dict

**describe** (*self*, *print\_name=False*, *return\_dict=False*)

Describe a component and its parameters

**Parameters**

- **print\_name** (*bool*, *optional*) – whether to print name of component
- **return\_dict** (*bool*, *optional*) – whether to return description as dictionary in the format {"name": name, "parameters": parameters}

**Returns** prints and returns dictionary

**Return type** None or dict

**fit** (*self*, *X*, *y*)

Fits the sampler to the data.

**Parameters**

- **X** (*pd.DataFrame*) – Input features.
- **y** (*pd.Series*) – Target.

**Returns** self

**fit\_transform** (*self*, *X*, *y*)

Fits on X and transforms X

**Parameters**

- **X** (*pd.DataFrame*) – Data to fit and transform
- **y** (*pd.Series*) – Target data

**Returns** Transformed X

**Return type** pd.DataFrame

**static load** (*file\_path*)

Loads component at file path

**Parameters** **file\_path** (*str*) – Location to load file

**Returns** ComponentBase object

**needs\_fitting** (*self*)

Returns boolean determining if component needs fitting before calling predict, predict\_proba, transform, or feature\_importances. This can be overridden to False for components that do not need to be fit or whose fit methods do nothing.

**property parameters** (*self*)

Returns the parameters which were used to initialize the component

**save** (*self*, *file\_path*, *pickle\_protocol=cloudpickle.DEFAULT\_PROTOCOL*)

Saves component at file path

**Parameters**

- **file\_path** (*str*) – Location to save file
- **pickle\_protocol** (*int*) – The pickle data stream format.

**Returns** None

**transform** (*self*, *X*, *y=None*)

Transforms the input data by sampling the data.

**Parameters**

- **X** (*pd.DataFrame*) – Training features.
- **y** (*pd.Series*) – Target.

**Returns** Transformed features and target.

**Return type** *pd.DataFrame*, *pd.Series*

```
class evalml.pipelines.components.transformers.samplers.oversamplers.SMOTEOverSampler (sampling_ratio, k_neighbors_default, n_jobs, random_seed, hyperparameter_ranges, model_family, modifies_features, modifies_target, name)
```

SMOTE Oversampler component. Works on numerical datasets only. This component is only run during training and not during predict.

**Parameters**

- **sampling\_ratio** (*float*) – This is the goal ratio of the minority to majority class, with range (0, 1]. A value of 0.25 means we want a 1:4 ratio of the minority to majority class after oversampling. We will create the a sampling dictionary using this ratio, with the keys corresponding to the class and the values responding to the number of samples. Defaults to 0.25.
- **k\_neighbors\_default** (*int*) – The number of nearest neighbors used to construct synthetic samples. This is the default value used, but the actual *k\_neighbors* value might be smaller if there are less samples. Defaults to 5.
- **n\_jobs** (*int*) – The number of CPU cores to use. Defaults to -1.
- **random\_seed** (*int*) – The seed to use for random sampling. Defaults to 0.

**Attributes**

<b>hyperparameter_ranges</b>	{}
<b>model_family</b>	ModelFamily.NONE
<b>modifies_features</b>	True
<b>modifies_target</b>	True
<b>name</b>	SMOTE Oversampler

**Methods**

<i>clone</i>	Constructs a new component with the same parameters and random state.
<i>default_parameters</i>	Returns the default parameters for this component.
<i>describe</i>	Describe a component and its parameters
<i>fit</i>	Fits the sampler to the data.
<i>fit_transform</i>	Fits on X and transforms X

continues on next page

Table 481 – continued from previous page

<code>load</code>	Loads component at file path
<code>needs_fitting</code>	Returns boolean determining if component needs fitting before
<code>parameters</code>	Returns the parameters which were used to initialize the component
<code>save</code>	Saves component at file path
<code>transform</code>	Transforms the input data by sampling the data.

**clone** (*self*)

Constructs a new component with the same parameters and random state.

**Returns** A new instance of this component with identical parameters and random state.

**default\_parameters** (*cls*)

Returns the default parameters for this component.

Our convention is that `Component.default_parameters == Component().parameters`.

**Returns** default parameters for this component.

**Return type** dict

**describe** (*self*, *print\_name=False*, *return\_dict=False*)

Describe a component and its parameters

**Parameters**

- **print\_name** (*bool*, *optional*) – whether to print name of component
- **return\_dict** (*bool*, *optional*) – whether to return description as dictionary in the format {"name": name, "parameters": parameters}

**Returns** prints and returns dictionary

**Return type** None or dict

**fit** (*self*, *X*, *y*)

Fits the sampler to the data.

**Parameters**

- **X** (*pd.DataFrame*) – Input features.
- **y** (*pd.Series*) – Target.

**Returns** self

**fit\_transform** (*self*, *X*, *y*)

Fits on X and transforms X

**Parameters**

- **X** (*pd.DataFrame*) – Data to fit and transform
- **y** (*pd.Series*) – Target data

**Returns** Transformed X

**Return type** pd.DataFrame

**static load** (*file\_path*)

Loads component at file path

**Parameters** **file\_path** (*str*) – Location to load file



**Returns** ComponentBase object

**needs\_fitting** (*self*)  
Returns boolean determining if component needs fitting before calling predict, predict\_proba, transform, or feature\_importances. This can be overridden to False for components that do not need to be fit or whose fit methods do nothing.

**property parameters** (*self*)  
Returns the parameters which were used to initialize the component

**save** (*self*, *file\_path*, *pickle\_protocol*=cloudpickle.DEFAULT\_PROTOCOL)  
Saves component at file path

**Parameters**

- **file\_path** (*str*) – Location to save file
- **pickle\_protocol** (*int*) – The pickle data stream format.

**Returns** None

**transform** (*self*, *X*, *y*=None)  
Transforms the input data by sampling the data.

**Parameters**

- **X** (*pd.DataFrame*) – Training features.
- **y** (*pd.Series*) – Target.

**Returns** Transformed features and target.

**Return type** pd.DataFrame, pd.Series

**undersampler**

**Module Contents**

**Classes Summary**

<i>Undersampler</i>	Initializes an undersampling transformer to downsample the majority classes in the dataset.
---------------------	---

**Contents**

**class** evalml.pipelines.components.transformers.samplers.undersampler.**Undersampler** (*sampling\_ratio*, *sampling\_ratio\_min\_samples*, *min\_percent*, *random\_seed*=0, *\*\*kwargs*)

Initializes an undersampling transformer to downsample the majority classes in the dataset.

This component is only run during training and not during predict.

**Parameters**

- **sampling\_ratio** (*float*) – The smallest minority:majority ratio that is accepted as ‘balanced’. For instance, a 1:4 ratio would be represented as 0.25, while a 1:1 ratio is 1.0. Must be between 0 and 1, inclusive. Defaults to 0.25.
- **sampling\_ratio\_dict** (*dict*) – A dictionary specifying the desired balanced ratio for each target value. For instance, in a binary case where class 1 is the minority, we could specify: `sampling_ratio_dict={0: 0.5, 1: 1}`, which means we would undersample class 0 to have twice the number of samples as class 1 (minority:majority ratio = 0.5), and don’t sample class 1. Overrides `sampling_ratio` if provided. Defaults to None.
- **min\_samples** (*int*) – The minimum number of samples that we must have for any class, pre or post sampling. If a class must be downsampled, it will not be downsampled past this value. To determine severe imbalance, the minority class must occur less often than this and must have a class ratio below `min_percentage`. Must be greater than 0. Defaults to 100.
- **min\_percentage** (*float*) – The minimum percentage of the minimum class to total dataset that we tolerate, as long as it is above `min_samples`. If `min_percentage` and `min_samples` are not met, treat this as severely imbalanced, and we will not resample the data. Must be between 0 and 0.5, inclusive. Defaults to 0.1.
- **random\_seed** (*int*) – The seed to use for random sampling. Defaults to 0.

#### Attributes

<b>hyper-parameter_ranges</b>	{}
<b>model_family</b>	ModelFamily.NONE
<b>modifies_features</b>	True
<b>modifies_target</b>	True
<b>name</b>	Undersampler

#### Methods

<code>clone</code>	Constructs a new component with the same parameters and random state.
<code>default_parameters</code>	Returns the default parameters for this component.
<code>describe</code>	Describe a component and its parameters
<code>fit</code>	Fits the sampler to the data.
<code>fit_transform</code>	Fits on X and transforms X
<code>load</code>	Loads component at file path
<code>needs_fitting</code>	Returns boolean determining if component needs fitting before
<code>parameters</code>	Returns the parameters which were used to initialize the component
<code>save</code>	Saves component at file path
<code>transform</code>	Transforms the input data by sampling the data.

#### **clone** (*self*)

Constructs a new component with the same parameters and random state.

**Returns** A new instance of this component with identical parameters and random state.

#### **default\_parameters** (*cls*)

Returns the default parameters for this component.

Our convention is that `Component.default_parameters == Component().parameters`.

**Returns** default parameters for this component.

**Return type** dict

**describe** (*self*, *print\_name=False*, *return\_dict=False*)

Describe a component and its parameters

**Parameters**

- **print\_name** (*bool*, *optional*) – whether to print name of component
- **return\_dict** (*bool*, *optional*) – whether to return description as dictionary in the format {"name": name, "parameters": parameters}

**Returns** prints and returns dictionary

**Return type** None or dict

**fit** (*self*, *X*, *y*)

Fits the sampler to the data.

**Parameters**

- **X** (*pd.DataFrame*) – Input features.
- **y** (*pd.Series*) – Target.

**Returns** self

**fit\_transform** (*self*, *X*, *y*)

Fits on X and transforms X

**Parameters**

- **X** (*pd.DataFrame*) – Data to fit and transform
- **y** (*pd.Series*) – Target data

**Returns** Transformed X

**Return type** pd.DataFrame

**static load** (*file\_path*)

Loads component at file path

**Parameters** **file\_path** (*str*) – Location to load file

**Returns** ComponentBase object

**needs\_fitting** (*self*)

Returns boolean determining if component needs fitting before calling predict, predict\_proba, transform, or feature\_importances. This can be overridden to False for components that do not need to be fit or whose fit methods do nothing.

**property parameters** (*self*)

Returns the parameters which were used to initialize the component

**save** (*self*, *file\_path*, *pickle\_protocol=cloudpickle.DEFAULT\_PROTOCOL*)

Saves component at file path

**Parameters**

- **file\_path** (*str*) – Location to save file

- **pickle\_protocol** (*int*) – The pickle data stream format.

**Returns** None

**transform** (*self, X, y=None*)

Transforms the input data by sampling the data.

**Parameters**

- **X** (*pd.DataFrame*) – Training features.
- **y** (*pd.Series*) – Target.

**Returns** Transformed features and target.

**Return type** *pd.DataFrame, pd.Series*

## Package Contents

### Classes Summary

<i>SMOTENCOverSampler</i>	SMOTENC Oversampler component. Uses SMOTENC to generate synthetic samples. Works on a mix of numerical and categorical columns.
<i>SMOTENOverSampler</i>	SMOTEN Oversampler component. Uses SMOTEN to generate synthetic samples. Works for purely categorical datasets.
<i>SMOTEOverSampler</i>	SMOTE Oversampler component. Works on numerical datasets only. This component is only run during training and not during predict.
<i>Undersampler</i>	Initializes an undersampling transformer to downsample the majority classes in the dataset.

### Contents

```
class evalml.pipelines.components.transformers.samplers.SMOTENCOverSampler (sampling_ratio=0.25,  
                                                                    k_neighbors_default=5,  
                                                                    n_jobs=-1,  
                                                                    random_seed=0,  
                                                                    **kwargs)
```

SMOTENC Oversampler component. Uses SMOTENC to generate synthetic samples. Works on a mix of numerical and categorical columns. Input data must be Woodwork type, and this component is only run during training and not during predict.

**Parameters**

- **sampling\_ratio** (*float*) – This is the goal ratio of the minority to majority class, with range (0, 1]. A value of 0.25 means we want a 1:4 ratio of the minority to majority class after oversampling. We will create the a sampling dictionary using this ratio, with the keys corresponding to the class and the values responding to the number of samples. Defaults to 0.25.

- **k\_neighbors\_default** (*int*) – The number of nearest neighbors used to construct synthetic samples. This is the default value used, but the actual k\_neighbors value might be smaller if there are less samples. Defaults to 5.
- **n\_jobs** (*int*) – The number of CPU cores to use. Defaults to -1.
- **random\_seed** (*int*) – The seed to use for random sampling. Defaults to 0.

#### Attributes

<b>hyper-parameter_ranges</b>	{}
<b>model_family</b>	ModelFamily.NONE
<b>modifies_features</b>	True
<b>modifies_target</b>	True
<b>name</b>	SMOTENC Oversampler

#### Methods

<i>clone</i>	Constructs a new component with the same parameters and random state.
<i>default_parameters</i>	Returns the default parameters for this component.
<i>describe</i>	Describe a component and its parameters
<i>fit</i>	Fits the sampler to the data.
<i>fit_transform</i>	Fits on X and transforms X
<i>load</i>	Loads component at file path
<i>needs_fitting</i>	Returns boolean determining if component needs fitting before
<i>parameters</i>	Returns the parameters which were used to initialize the component
<i>save</i>	Saves component at file path
<i>transform</i>	Transforms the input data by sampling the data.

#### **clone** (*self*)

Constructs a new component with the same parameters and random state.

**Returns** A new instance of this component with identical parameters and random state.

#### **default\_parameters** (*cls*)

Returns the default parameters for this component.

Our convention is that `Component.default_parameters == Component().parameters`.

**Returns** default parameters for this component.

**Return type** dict

#### **describe** (*self*, *print\_name=False*, *return\_dict=False*)

Describe a component and its parameters

#### Parameters

- **print\_name** (*bool*, *optional*) – whether to print name of component

- **return\_dict** (*bool, optional*) – whether to return description as dictionary in the format {"name": name, "parameters": parameters}

**Returns** prints and returns dictionary

**Return type** None or dict

**fit** (*self, X, y*)

Fits the sampler to the data.

**Parameters**

- **X** (*pd.DataFrame*) – Input features.
- **y** (*pd.Series*) – Target.

**Returns** self

**fit\_transform** (*self, X, y*)

Fits on X and transforms X

**Parameters**

- **X** (*pd.DataFrame*) – Data to fit and transform
- **y** (*pd.Series*) – Target data

**Returns** Transformed X

**Return type** pd.DataFrame

**static load** (*file\_path*)

Loads component at file path

**Parameters** **file\_path** (*str*) – Location to load file

**Returns** ComponentBase object

**needs\_fitting** (*self*)

Returns boolean determining if component needs fitting before calling predict, predict\_proba, transform, or feature\_importances. This can be overridden to False for components that do not need to be fit or whose fit methods do nothing.

**property parameters** (*self*)

Returns the parameters which were used to initialize the component

**save** (*self, file\_path, pickle\_protocol=cloudpickle.DEFAULT\_PROTOCOL*)

Saves component at file path

**Parameters**

- **file\_path** (*str*) – Location to save file
- **pickle\_protocol** (*int*) – The pickle data stream format.

**Returns** None

**transform** (*self, X, y=None*)

Transforms the input data by sampling the data.

**Parameters**

- **X** (*pd.DataFrame*) – Training features.
- **y** (*pd.Series*) – Target.

**Returns** Transformed features and target.

**Return type** `pd.DataFrame, pd.Series`

```
class evalml.pipelines.components.transformers.samplers.SMOTENOversampler (sampling_ratio=0.25,
                                                                    k_neighbors_default=5,
                                                                    n_jobs=-
                                                                    1,
                                                                    ran-
                                                                    dom_seed=0,
                                                                    **kwargs)
```

SMOTEN Oversampler component. Uses SMOTEN to generate synthetic samples. Works for purely categorical datasets. This component is only run during training and not during predict.

#### Parameters

- **sampling\_ratio** (*float*) – This is the goal ratio of the minority to majority class, with range (0, 1]. A value of 0.25 means we want a 1:4 ratio of the minority to majority class after oversampling. We will create the a sampling dictionary using this ratio, with the keys corresponding to the class and the values responding to the number of samples. Defaults to 0.25.
- **k\_neighbors\_default** (*int*) – The number of nearest neighbors used to construct synthetic samples. This is the default value used, but the actual `k_neighbors` value might be smaller if there are less samples. Defaults to 5.
- **n\_jobs** (*int*) – The number of CPU cores to use. Defaults to -1.
- **random\_seed** (*int*) – The seed to use for random sampling. Defaults to 0.

#### Attributes

<b>hyper-parameter_ranges</b>	<code>{}</code>
<b>model_family</b>	<code>ModelFamily.NONE</code>
<b>modifies_features</b>	<code>True</code>
<b>modifies_target</b>	<code>True</code>
<b>name</b>	SMOTEN Oversampler

#### Methods

<code>clone</code>	Constructs a new component with the same parameters and random state.
<code>default_parameters</code>	Returns the default parameters for this component.
<code>describe</code>	Describe a component and its parameters
<code>fit</code>	Fits the sampler to the data.
<code>fit_transform</code>	Fits on X and transforms X
<code>load</code>	Loads component at file path
<code>needs_fitting</code>	Returns boolean determining if component needs fitting before
<code>parameters</code>	Returns the parameters which were used to initialize the component
<code>save</code>	Saves component at file path

continues on next page

Table 486 – continued from previous page

<i>transform</i>	Transforms the input data by sampling the data.
<b>clone</b> ( <i>self</i> )	
Constructs a new component with the same parameters and random state.	
<b>Returns</b> A new instance of this component with identical parameters and random state.	
<b>default_parameters</b> ( <i>cls</i> )	
Returns the default parameters for this component.	
Our convention is that <code>Component.default_parameters == Component().parameters</code> .	
<b>Returns</b> default parameters for this component.	
<b>Return type</b> dict	
<b>describe</b> ( <i>self</i> , <i>print_name=False</i> , <i>return_dict=False</i> )	
Describe a component and its parameters	
<b>Parameters</b>	
<ul style="list-style-type: none"> <li>• <b>print_name</b> (<i>bool</i>, <i>optional</i>) – whether to print name of component</li> <li>• <b>return_dict</b> (<i>bool</i>, <i>optional</i>) – whether to return description as dictionary in the format {"name": name, "parameters": parameters}</li> </ul>	
<b>Returns</b> prints and returns dictionary	
<b>Return type</b> None or dict	
<b>fit</b> ( <i>self</i> , <i>X</i> , <i>y</i> )	
Fits the sampler to the data.	
<b>Parameters</b>	
<ul style="list-style-type: none"> <li>• <b>X</b> (<i>pd.DataFrame</i>) – Input features.</li> <li>• <b>y</b> (<i>pd.Series</i>) – Target.</li> </ul>	
<b>Returns</b> self	
<b>fit_transform</b> ( <i>self</i> , <i>X</i> , <i>y</i> )	
Fits on X and transforms X	
<b>Parameters</b>	
<ul style="list-style-type: none"> <li>• <b>X</b> (<i>pd.DataFrame</i>) – Data to fit and transform</li> <li>• <b>y</b> (<i>pd.Series</i>) – Target data</li> </ul>	
<b>Returns</b> Transformed X	
<b>Return type</b> pd.DataFrame	
<b>static load</b> ( <i>file_path</i> )	
Loads component at file path	
<b>Parameters</b> <b>file_path</b> ( <i>str</i> ) – Location to load file	
<b>Returns</b> ComponentBase object	
<b>needs_fitting</b> ( <i>self</i> )	
Returns boolean determining if component needs fitting before calling predict, predict_proba, transform, or feature_importances. This can be overridden to False for components that do not need to be fit or whose fit methods do nothing.	



**property parameters** (*self*)

Returns the parameters which were used to initialize the component

**save** (*self*, *file\_path*, *pickle\_protocol*=*cloudpickle.DEFAULT\_PROTOCOL*)

Saves component at file path

**Parameters**

- **file\_path** (*str*) – Location to save file
- **pickle\_protocol** (*int*) – The pickle data stream format.

**Returns** None

**transform** (*self*, *X*, *y*=*None*)

Transforms the input data by sampling the data.

**Parameters**

- **X** (*pd.DataFrame*) – Training features.
- **y** (*pd.Series*) – Target.

**Returns** Transformed features and target.

**Return type** *pd.DataFrame*, *pd.Series*

```
class evalml.pipelines.components.transformers.samplers.SMOTEOversampler (sampling_ratio=0.25,
                                                                    k_neighbors_default=5,
                                                                    n_jobs=-1,
                                                                    random_seed=0,
                                                                    **kwargs)
```

SMOTE Oversampler component. Works on numerical datasets only. This component is only run during training and not during predict.

**Parameters**

- **sampling\_ratio** (*float*) – This is the goal ratio of the minority to majority class, with range (0, 1]. A value of 0.25 means we want a 1:4 ratio of the minority to majority class after oversampling. We will create the a sampling dictionary using this ratio, with the keys corresponding to the class and the values responding to the number of samples. Defaults to 0.25.
- **k\_neighbors\_default** (*int*) – The number of nearest neighbors used to construct synthetic samples. This is the default value used, but the actual *k\_neighbors* value might be smaller if there are less samples. Defaults to 5.
- **n\_jobs** (*int*) – The number of CPU cores to use. Defaults to -1.
- **random\_seed** (*int*) – The seed to use for random sampling. Defaults to 0.

**Attributes**

<b>hyper-parameter_ranges</b>	{}
<b>model_family</b>	ModelFamily.NONE
<b>modifies_features</b>	True
<b>modifies_target</b>	True
<b>name</b>	SMOTE Oversampler

## Methods

<i>clone</i>	Constructs a new component with the same parameters and random state.
<i>default_parameters</i>	Returns the default parameters for this component.
<i>describe</i>	Describe a component and its parameters
<i>fit</i>	Fits the sampler to the data.
<i>fit_transform</i>	Fits on X and transforms X
<i>load</i>	Loads component at file path
<i>needs_fitting</i>	Returns boolean determining if component needs fitting before
<i>parameters</i>	Returns the parameters which were used to initialize the component
<i>save</i>	Saves component at file path
<i>transform</i>	Transforms the input data by sampling the data.

### **clone** (*self*)

Constructs a new component with the same parameters and random state.

**Returns** A new instance of this component with identical parameters and random state.

### **default\_parameters** (*cls*)

Returns the default parameters for this component.

Our convention is that `Component.default_parameters == Component().parameters`.

**Returns** default parameters for this component.

**Return type** dict

### **describe** (*self*, *print\_name=False*, *return\_dict=False*)

Describe a component and its parameters

#### Parameters

- **print\_name** (*bool*, *optional*) – whether to print name of component
- **return\_dict** (*bool*, *optional*) – whether to return description as dictionary in the format {"name": name, "parameters": parameters}

**Returns** prints and returns dictionary

**Return type** None or dict

### **fit** (*self*, *X*, *y*)

Fits the sampler to the data.

#### Parameters

- **X** (*pd.DataFrame*) – Input features.
- **y** (*pd.Series*) – Target.

**Returns** *self*

**fit\_transform** (*self*, *X*, *y*)  
Fits on *X* and transforms *X*

**Parameters**

- **X** (*pd.DataFrame*) – Data to fit and transform
- **y** (*pd.Series*) – Target data

**Returns** Transformed *X*

**Return type** *pd.DataFrame*

**static load** (*file\_path*)  
Loads component at file path

**Parameters** **file\_path** (*str*) – Location to load file

**Returns** *ComponentBase* object

**needs\_fitting** (*self*)  
Returns boolean determining if component needs fitting before calling *predict*, *predict\_proba*, *transform*, or *feature\_importances*. This can be overridden to *False* for components that do not need to be fit or whose fit methods do nothing.

**property parameters** (*self*)  
Returns the parameters which were used to initialize the component

**save** (*self*, *file\_path*, *pickle\_protocol=cloudpickle.DEFAULT\_PROTOCOL*)  
Saves component at file path

**Parameters**

- **file\_path** (*str*) – Location to save file
- **pickle\_protocol** (*int*) – The pickle data stream format.

**Returns** *None*

**transform** (*self*, *X*, *y=None*)  
Transforms the input data by sampling the data.

**Parameters**

- **X** (*pd.DataFrame*) – Training features.
- **y** (*pd.Series*) – Target.

**Returns** Transformed features and target.

**Return type** *pd.DataFrame*, *pd.Series*

**class** *evalml.pipelines.components.transformers.samplers.Undersampler* (*sampling\_ratio=0.25*,  
*sampling\_ratio\_dict=None*,  
*min\_samples=100*,  
*min\_percentage=0.1*,  
*random\_seed=0*,  
*\*\*kwargs*)

Initializes an undersampling transformer to downsample the majority classes in the dataset.

This component is only run during training and not during predict.

### Parameters

- **sampling\_ratio** (*float*) – The smallest minority:majority ratio that is accepted as ‘balanced’. For instance, a 1:4 ratio would be represented as 0.25, while a 1:1 ratio is 1.0. Must be between 0 and 1, inclusive. Defaults to 0.25.
- **sampling\_ratio\_dict** (*dict*) – A dictionary specifying the desired balanced ratio for each target value. For instance, in a binary case where class 1 is the minority, we could specify: `sampling_ratio_dict={0: 0.5, 1: 1}`, which means we would undersample class 0 to have twice the number of samples as class 1 (minority:majority ratio = 0.5), and don’t sample class 1. Overrides `sampling_ratio` if provided. Defaults to None.
- **min\_samples** (*int*) – The minimum number of samples that we must have for any class, pre or post sampling. If a class must be downsampled, it will not be downsampled past this value. To determine severe imbalance, the minority class must occur less often than this and must have a class ratio below `min_percentage`. Must be greater than 0. Defaults to 100.
- **min\_percentage** (*float*) – The minimum percentage of the minimum class to total dataset that we tolerate, as long as it is above `min_samples`. If `min_percentage` and `min_samples` are not met, treat this as severely imbalanced, and we will not resample the data. Must be between 0 and 0.5, inclusive. Defaults to 0.1.
- **random\_seed** (*int*) – The seed to use for random sampling. Defaults to 0.

### Attributes

<b>hyper-parameter_ranges</b>	{}
<b>model_family</b>	ModelFamily.NONE
<b>modifies_features</b>	True
<b>modifies_target</b>	True
<b>name</b>	Undersampler

### Methods

<code>clone</code>	Constructs a new component with the same parameters and random state.
<code>default_parameters</code>	Returns the default parameters for this component.
<code>describe</code>	Describe a component and its parameters
<code>fit</code>	Fits the sampler to the data.
<code>fit_transform</code>	Fits on X and transforms X
<code>load</code>	Loads component at file path
<code>needs_fitting</code>	Returns boolean determining if component needs fitting before
<code>parameters</code>	Returns the parameters which were used to initialize the component
<code>save</code>	Saves component at file path
<code>transform</code>	Transforms the input data by sampling the data.

#### **clone** (*self*)

Constructs a new component with the same parameters and random state.

**Returns** A new instance of this component with identical parameters and random state.

**default\_parameters** (*cls*)

Returns the default parameters for this component.

Our convention is that `Component.default_parameters == Component().parameters`.

**Returns** default parameters for this component.

**Return type** dict

**describe** (*self*, *print\_name=False*, *return\_dict=False*)

Describe a component and its parameters

**Parameters**

- **print\_name** (*bool*, *optional*) – whether to print name of component
- **return\_dict** (*bool*, *optional*) – whether to return description as dictionary in the format {"name": name, "parameters": parameters}

**Returns** prints and returns dictionary

**Return type** None or dict

**fit** (*self*, *X*, *y*)

Fits the sampler to the data.

**Parameters**

- **X** (*pd.DataFrame*) – Input features.
- **y** (*pd.Series*) – Target.

**Returns** self

**fit\_transform** (*self*, *X*, *y*)

Fits on X and transforms X

**Parameters**

- **X** (*pd.DataFrame*) – Data to fit and transform
- **y** (*pd.Series*) – Target data

**Returns** Transformed X

**Return type** pd.DataFrame

**static load** (*file\_path*)

Loads component at file path

**Parameters** **file\_path** (*str*) – Location to load file

**Returns** ComponentBase object

**needs\_fitting** (*self*)

Returns boolean determining if component needs fitting before calling predict, predict\_proba, transform, or feature\_importances. This can be overridden to False for components that do not need to be fit or whose fit methods do nothing.

**property parameters** (*self*)

Returns the parameters which were used to initialize the component

**save** (*self*, *file\_path*, *pickle\_protocol=cloudpickle.DEFAULT\_PROTOCOL*)

Saves component at file path

**Parameters**

- `file_path` (*str*) – Location to save file
- `pickle_protocol` (*int*) – The pickle data stream format.

**Returns** None

**transform** (*self*, *X*, *y=None*)  
Transforms the input data by sampling the data.

**Parameters**

- **X** (*pd.DataFrame*) – Training features.
- **y** (*pd.Series*) – Target.

**Returns** Transformed features and target.

**Return type** *pd.DataFrame*, *pd.Series*

scalers

Submodules

standard\_scaler

Module Contents

Classes Summary

<i>StandardScaler</i>	A transformer that standardizes input features by removing the mean and scaling to unit variance.
-----------------------	---

Contents

**class** evalml.pipelines.components.transformers.scalers.standard\_scaler.**StandardScaler** (*random* *\*\*kwargs*)

A transformer that standardizes input features by removing the mean and scaling to unit variance.

**Parameters** `random_seed` (*int*) – Seed for the random number generator. Defaults to 0.

**Attributes**

<b>hyper-parameter_ranges</b>	{}
<b>model_family</b>	ModelFamily.NONE
<b>modifies_features</b>	True
<b>modifies_target</b>	False
<b>name</b>	Standard Scaler

**Methods**

<code>clone</code>	Constructs a new component with the same parameters and random state.
<code>default_parameters</code>	Returns the default parameters for this component.
<code>describe</code>	Describe a component and its parameters
<code>fit</code>	Fits component to data
<code>fit_transform</code>	Fits on X and transforms X
<code>load</code>	Loads component at file path
<code>needs_fitting</code>	Returns boolean determining if component needs fitting before
<code>parameters</code>	Returns the parameters which were used to initialize the component
<code>save</code>	Saves component at file path
<code>transform</code>	Transforms data X.

**clone** (*self*)

Constructs a new component with the same parameters and random state.

**Returns** A new instance of this component with identical parameters and random state.

**default\_parameters** (*cls*)

Returns the default parameters for this component.

Our convention is that `Component.default_parameters == Component().parameters`.

**Returns** default parameters for this component.

**Return type** dict

**describe** (*self*, *print\_name=False*, *return\_dict=False*)

Describe a component and its parameters

**Parameters**

- **print\_name** (*bool*, *optional*) – whether to print name of component
- **return\_dict** (*bool*, *optional*) – whether to return description as dictionary in the format {"name": name, "parameters": parameters}

**Returns** prints and returns dictionary

**Return type** None or dict

**fit** (*self*, *X*, *y=None*)

Fits component to data

**Parameters**

- **X** (*list*, *pd.DataFrame* or *np.ndarray*) – The input training data of shape [n\_samples, n\_features]
- **y** (*list*, *pd.Series*, *np.ndarray*, *optional*) – The target training data of length [n\_samples]

**Returns** self

**fit\_transform** (*self*, *X*, *y=None*)

Fits on X and transforms X

**Parameters**

- **X** (*pd.DataFrame*) – Data to fit and transform

- **y** (*pd.Series*) – Target data

**Returns** Transformed X

**Return type** *pd.DataFrame*

**static load** (*file\_path*)

Loads component at file path

**Parameters** **file\_path** (*str*) – Location to load file

**Returns** *ComponentBase* object

**needs\_fitting** (*self*)

Returns boolean determining if component needs fitting before calling *predict*, *predict\_proba*, *transform*, or *feature\_importances*. This can be overridden to *False* for components that do not need to be fit or whose fit methods do nothing.

**property parameters** (*self*)

Returns the parameters which were used to initialize the component

**save** (*self, file\_path, pickle\_protocol=cloudpickle.DEFAULT\_PROTOCOL*)

Saves component at file path

**Parameters**

- **file\_path** (*str*) – Location to save file
- **pickle\_protocol** (*int*) – The pickle data stream format.

**Returns** *None*

**transform** (*self, X, y=None*)

Transforms data X.

**Parameters**

- **X** (*pd.DataFrame*) – Data to transform.
- **y** (*pd.Series, optional*) – Target data.

**Returns** Transformed X

**Return type** *pd.DataFrame*

## Package Contents

## Classes Summary

---

*StandardScaler*

A transformer that standardizes input features by removing the mean and scaling to unit variance.

---



## Contents

**class** evalml.pipelines.components.transformers.scalers.**StandardScaler** (*random\_seed=0*, *\*\*kwargs*)

A transformer that standardizes input features by removing the mean and scaling to unit variance.

**Parameters** **random\_seed** (*int*) – Seed for the random number generator. Defaults to 0.

### Attributes

<b>hyper-parameter_ranges</b>	{}
<b>model_family</b>	ModelFamily.NONE
<b>modifies_features</b>	True
<b>modifies_target</b>	False
<b>name</b>	Standard Scaler

### Methods

<i>clone</i>	Constructs a new component with the same parameters and random state.
<i>default_parameters</i>	Returns the default parameters for this component.
<i>describe</i>	Describe a component and its parameters
<i>fit</i>	Fits component to data
<i>fit_transform</i>	Fits on X and transforms X
<i>load</i>	Loads component at file path
<i>needs_fitting</i>	Returns boolean determining if component needs fitting before
<i>parameters</i>	Returns the parameters which were used to initialize the component
<i>save</i>	Saves component at file path
<i>transform</i>	Transforms data X.

**clone** (*self*)

Constructs a new component with the same parameters and random state.

**Returns** A new instance of this component with identical parameters and random state.

**default\_parameters** (*cls*)

Returns the default parameters for this component.

Our convention is that `Component.default_parameters == Component().parameters`.

**Returns** default parameters for this component.

**Return type** dict

**describe** (*self*, *print\_name=False*, *return\_dict=False*)

Describe a component and its parameters

**Parameters**

- **print\_name** (*bool*, *optional*) – whether to print name of component

- **return\_dict** (*bool, optional*) – whether to return description as dictionary in the format {"name": name, "parameters": parameters}

**Returns** prints and returns dictionary

**Return type** None or dict

**fit** (*self, X, y=None*)

Fits component to data

**Parameters**

- **X** (*list, pd.DataFrame or np.ndarray*) – The input training data of shape [n\_samples, n\_features]
- **y** (*list, pd.Series, np.ndarray, optional*) – The target training data of length [n\_samples]

**Returns** self

**fit\_transform** (*self, X, y=None*)

Fits on X and transforms X

**Parameters**

- **X** (*pd.DataFrame*) – Data to fit and transform
- **y** (*pd.Series*) – Target data

**Returns** Transformed X

**Return type** pd.DataFrame

**static load** (*file\_path*)

Loads component at file path

**Parameters** **file\_path** (*str*) – Location to load file

**Returns** ComponentBase object

**needs\_fitting** (*self*)

Returns boolean determining if component needs fitting before calling predict, predict\_proba, transform, or feature\_importances. This can be overridden to False for components that do not need to be fit or whose fit methods do nothing.

**property parameters** (*self*)

Returns the parameters which were used to initialize the component

**save** (*self, file\_path, pickle\_protocol=cloudpickle.DEFAULT\_PROTOCOL*)

Saves component at file path

**Parameters**

- **file\_path** (*str*) – Location to save file
- **pickle\_protocol** (*int*) – The pickle data stream format.

**Returns** None

**transform** (*self, X, y=None*)

Transforms data X.

**Parameters**

- **X** (*pd.DataFrame*) – Data to transform.
- **y** (*pd.Series, optional*) – Target data.

**Returns** Transformed X**Return type** pd.DataFrame

## Submodules

### column\_selectors

## Module Contents

## Classes Summary

<i>ColumnSelector</i>	Initializes an transformer that drops specified columns in input data.
<i>DropColumns</i>	Drops specified columns in input data.
<i>SelectByType</i>	Selects columns by specified Woodwork logical type or semantic tag in input data.
<i>SelectColumns</i>	Selects specified columns in input data.

## Contents

**class** evalml.pipelines.components.transformers.column\_selectors.**ColumnSelector** (*columns=None, random\_seed=0, \*\*kwargs*)

Initializes an transformer that drops specified columns in input data.

### Parameters

- **columns** (*list(string)*) – List of column names, used to determine which columns to select.
- **random\_seed** (*int*) – Seed for the random number generator. Defaults to 0.

### Attributes

<b>model_family</b>	ModelFamily.NONE
<b>modifies_features</b>	True
<b>modifies_target</b>	False

### Methods

<i>clone</i>	Constructs a new component with the same parameters and random state.
<i>default_parameters</i>	Returns the default parameters for this component.
<i>describe</i>	Describe a component and its parameters
<i>fit</i>	Fits the transformer by checking if column names are present in the dataset.

continues on next page

Table 494 – continued from previous page

<code>fit_transform</code>	Fits on X and transforms X
<code>load</code>	Loads component at file path
<code>name</code>	Returns string name of this component
<code>needs_fitting</code>	Returns boolean determining if component needs fitting before
<code>parameters</code>	Returns the parameters which were used to initialize the component
<code>save</code>	Saves component at file path
<code>transform</code>	Transforms data X.

**clone** (*self*)

Constructs a new component with the same parameters and random state.

**Returns** A new instance of this component with identical parameters and random state.

**default\_parameters** (*cls*)

Returns the default parameters for this component.

Our convention is that `Component.default_parameters == Component().parameters`.

**Returns** default parameters for this component.

**Return type** dict

**describe** (*self*, *print\_name=False*, *return\_dict=False*)

Describe a component and its parameters

**Parameters**

- **print\_name** (*bool*, *optional*) – whether to print name of component
- **return\_dict** (*bool*, *optional*) – whether to return description as dictionary in the format {"name": name, "parameters": parameters}

**Returns** prints and returns dictionary

**Return type** None or dict

**fit** (*self*, *X*, *y=None*)

Fits the transformer by checking if column names are present in the dataset.

**Parameters**

- **X** (*pd.DataFrame*) – Data to check.
- **y** (*pd.Series*, *optional*) – Targets.

**Returns** self

**fit\_transform** (*self*, *X*, *y=None*)

Fits on X and transforms X

**Parameters**

- **X** (*pd.DataFrame*) – Data to fit and transform
- **y** (*pd.Series*) – Target data

**Returns** Transformed X

**Return type** pd.DataFrame

**static load** (*file\_path*)

Loads component at file path

**Parameters** `file_path` (*str*) – Location to load file

**Returns** ComponentBase object

**property** `name` (*cls*)

Returns string name of this component

**needs\_fitting** (*self*)

Returns boolean determining if component needs fitting before calling `predict`, `predict_proba`, `transform`, or `feature_importances`. This can be overridden to `False` for components that do not need to be fit or whose fit methods do nothing.

**property** `parameters` (*self*)

Returns the parameters which were used to initialize the component

**save** (*self*, *file\_path*, *pickle\_protocol*=`cloudpickle.DEFAULT_PROTOCOL`)

Saves component at file path

**Parameters**

- `file_path` (*str*) – Location to save file
- `pickle_protocol` (*int*) – The pickle data stream format.

**Returns** None

**transform** (*self*, *X*, *y*=*None*)

Transforms data *X*.

**Parameters**

- `X` (*pd.DataFrame*) – Data to transform.
- `y` (*pd.Series*, *optional*) – Target data.

**Returns** Transformed *X*

**Return type** `pd.DataFrame`

**class** `evalml.pipelines.components.transformers.column_selectors.DropColumns` (*columns*=*None*, *random\_seed*=0, *\*\*kwargs*)

Drops specified columns in input data.

**Parameters**

- `columns` (*list(string)*) – List of column names, used to determine which columns to drop.
- `random_seed` (*int*) – Seed for the random number generator. Defaults to 0.

**Attributes**

<b>hyper-parameter_ranges</b>	{}
<b>model_family</b>	ModelFamily.NONE
<b>modifies_features</b>	True
<b>modifies_target</b>	False
<b>name</b>	Drop Columns Transformer
<b>needs_fitting</b>	False

## Methods

<code>clone</code>	Constructs a new component with the same parameters and random state.
<code>default_parameters</code>	Returns the default parameters for this component.
<code>describe</code>	Describe a component and its parameters
<code>fit</code>	Fits the transformer by checking if column names are present in the dataset.
<code>fit_transform</code>	Fits on X and transforms X
<code>load</code>	Loads component at file path
<code>parameters</code>	Returns the parameters which were used to initialize the component
<code>save</code>	Saves component at file path
<code>transform</code>	Transforms data X by dropping columns.

### **clone** (*self*)

Constructs a new component with the same parameters and random state.

**Returns** A new instance of this component with identical parameters and random state.

### **default\_parameters** (*cls*)

Returns the default parameters for this component.

Our convention is that `Component.default_parameters == Component().parameters`.

**Returns** default parameters for this component.

**Return type** dict

### **describe** (*self*, *print\_name=False*, *return\_dict=False*)

Describe a component and its parameters

#### Parameters

- **print\_name** (*bool*, *optional*) – whether to print name of component
- **return\_dict** (*bool*, *optional*) – whether to return description as dictionary in the format {"name": name, "parameters": parameters}

**Returns** prints and returns dictionary

**Return type** None or dict

### **fit** (*self*, *X*, *y=None*)

Fits the transformer by checking if column names are present in the dataset.

#### Parameters

- **X** (*pd.DataFrame*) – Data to check.
- **y** (*pd.Series*, *optional*) – Targets.

**Returns** self

### **fit\_transform** (*self*, *X*, *y=None*)

Fits on X and transforms X

#### Parameters

- **X** (*pd.DataFrame*) – Data to fit and transform
- **y** (*pd.Series*) – Target data

**Returns** Transformed X

**Return type** `pd.DataFrame`

**static load** (*file\_path*)

Loads component at file path

**Parameters** *file\_path* (*str*) – Location to load file

**Returns** `ComponentBase` object

**property parameters** (*self*)

Returns the parameters which were used to initialize the component

**save** (*self*, *file\_path*, *pickle\_protocol=cloudpickle.DEFAULT\_PROTOCOL*)

Saves component at file path

**Parameters**

- *file\_path* (*str*) – Location to save file
- *pickle\_protocol* (*int*) – The pickle data stream format.

**Returns** `None`

**transform** (*self*, *X*, *y=None*)

Transforms data *X* by dropping columns.

**Parameters**

- *X* (*pd.DataFrame*) – Data to transform.
- *y* (*pd.Series*, *optional*) – Targets.

**Returns** Transformed *X*.

**Return type** `pd.DataFrame`

**class** `evalml.pipelines.components.transformers.column_selectors.SelectByType` (*column\_types=None*, *random\_seed=0*, *\*\*kwargs*)

Selects columns by specified Woodwork logical type or semantic tag in input data.

**Parameters**

- **column\_types** (*string*, *ww.LogicalType*, *list(string)*, *list(ww.LogicalType)*) – List of Woodwork types or tags, used to determine which columns to select.
- **random\_seed** (*int*) – Seed for the random number generator. Defaults to 0.

**Attributes**

<b>hyper-parameter_ranges</b>	{}
<b>model_family</b>	<code>ModelFamily.NONE</code>
<b>modifies_features</b>	<code>True</code>
<b>modifies_target</b>	<code>False</code>
<b>name</b>	Select Columns By Type Transformer
<b>needs_fitting</b>	<code>False</code>

## Methods

<code>clone</code>	Constructs a new component with the same parameters and random state.
<code>default_parameters</code>	Returns the default parameters for this component.
<code>describe</code>	Describe a component and its parameters
<code>fit</code>	Fits the transformer by checking if column names are present in the dataset.
<code>fit_transform</code>	Fits on $X$ and transforms $X$
<code>load</code>	Loads component at file path
<code>parameters</code>	Returns the parameters which were used to initialize the component
<code>save</code>	Saves component at file path
<code>transform</code>	Transforms data $X$ .

### **clone** (*self*)

Constructs a new component with the same parameters and random state.

**Returns** A new instance of this component with identical parameters and random state.

### **default\_parameters** (*cls*)

Returns the default parameters for this component.

Our convention is that `Component.default_parameters == Component().parameters`.

**Returns** default parameters for this component.

**Return type** dict

### **describe** (*self*, *print\_name=False*, *return\_dict=False*)

Describe a component and its parameters

#### Parameters

- **print\_name** (*bool*, *optional*) – whether to print name of component
- **return\_dict** (*bool*, *optional*) – whether to return description as dictionary in the format {"name": name, "parameters": parameters}

**Returns** prints and returns dictionary

**Return type** None or dict

### **fit** (*self*, $X$ , *y=None*)

Fits the transformer by checking if column names are present in the dataset.

#### Parameters

- **X** (*pd.DataFrame*) – Data to check.
- **y** (*pd.Series*, *optional*) – Targets.

**Returns** self

### **fit\_transform** (*self*, $X$ , *y=None*)

Fits on  $X$  and transforms  $X$

#### Parameters

- **X** (*pd.DataFrame*) – Data to fit and transform
- **y** (*pd.Series*) – Target data

**Returns** Transformed  $X$



**Return type** `pd.DataFrame`

**static load** (*file\_path*)

Loads component at file path

**Parameters** *file\_path* (*str*) – Location to load file

**Returns** `ComponentBase` object

**property parameters** (*self*)

Returns the parameters which were used to initialize the component

**save** (*self*, *file\_path*, *pickle\_protocol=cloudpickle.DEFAULT\_PROTOCOL*)

Saves component at file path

**Parameters**

- *file\_path* (*str*) – Location to save file
- *pickle\_protocol* (*int*) – The pickle data stream format.

**Returns** `None`

**transform** (*self*, *X*, *y=None*)

Transforms data *X*.

**Parameters**

- *X* (*pd.DataFrame*) – Data to transform.
- *y* (*pd.Series*, *optional*) – Target data.

**Returns** Transformed *X*

**Return type** `pd.DataFrame`

**class** `evalml.pipelines.components.transformers.column_selectors.SelectColumns` (*columns=None*, *random\_seed=0*, *\*\*kwargs*)

Selects specified columns in input data.

**Parameters**

- *columns* (*list(string)*) – List of column names, used to determine which columns to select.
- *random\_seed* (*int*) – Seed for the random number generator. Defaults to 0.

**Attributes**

<b>hyper-parameter_ranges</b>	{}
<b>model_family</b>	<code>ModelFamily.NONE</code>
<b>modifies_features</b>	<code>True</code>
<b>modifies_target</b>	<code>False</code>
<b>name</b>	Select Columns Transformer
<b>needs_fitting</b>	<code>False</code>

**Methods**

<code>clone</code>	Constructs a new component with the same parameters and random state.
<code>default_parameters</code>	Returns the default parameters for this component.
<code>describe</code>	Describe a component and its parameters
<code>fit</code>	Fits the transformer by checking if column names are present in the dataset.
<code>fit_transform</code>	Fits on X and transforms X
<code>load</code>	Loads component at file path
<code>parameters</code>	Returns the parameters which were used to initialize the component
<code>save</code>	Saves component at file path
<code>transform</code>	Transforms data X by selecting columns.

**clone** (*self*)

Constructs a new component with the same parameters and random state.

**Returns** A new instance of this component with identical parameters and random state.

**default\_parameters** (*cls*)

Returns the default parameters for this component.

Our convention is that `Component.default_parameters == Component().parameters`.

**Returns** default parameters for this component.

**Return type** dict

**describe** (*self*, *print\_name=False*, *return\_dict=False*)

Describe a component and its parameters

**Parameters**

- **print\_name** (*bool*, *optional*) – whether to print name of component
- **return\_dict** (*bool*, *optional*) – whether to return description as dictionary in the format {"name": name, "parameters": parameters}

**Returns** prints and returns dictionary

**Return type** None or dict

**fit** (*self*, *X*, *y=None*)

Fits the transformer by checking if column names are present in the dataset.

**Parameters**

- **X** (*pd.DataFrame*) – Data to check.
- **y** (*pd.Series*, *optional*) – Targets.

**Returns** self

**fit\_transform** (*self*, *X*, *y=None*)

Fits on X and transforms X

**Parameters**

- **X** (*pd.DataFrame*) – Data to fit and transform
- **y** (*pd.Series*) – Target data

**Returns** Transformed X

**Return type** `pd.DataFrame`

**static load** (*file\_path*)

Loads component at file path

**Parameters** *file\_path* (*str*) – Location to load file

**Returns** `ComponentBase` object

**property parameters** (*self*)

Returns the parameters which were used to initialize the component

**save** (*self*, *file\_path*, *pickle\_protocol=cloudpickle.DEFAULT\_PROTOCOL*)

Saves component at file path

**Parameters**

- *file\_path* (*str*) – Location to save file
- *pickle\_protocol* (*int*) – The pickle data stream format.

**Returns** `None`

**transform** (*self*, *X*, *y=None*)

Transforms data *X* by selecting columns.

**Parameters**

- *X* (*pd.DataFrame*) – Data to transform.
- *y* (*pd.Series*, *optional*) – Targets.

**Returns** Transformed *X*.

**Return type** `pd.DataFrame`

## transformer

## Module Contents

## Classes Summary

<i>TargetTransformer</i>	A component that transforms the target.
<i>Transformer</i>	A component that may or may not need fitting that transforms data.

## Contents

**class** `evalml.pipelines.components.transformers.transformer.TargetTransformer` (*parameters=None*, *component\_obj=None*, *random\_seed=0*, *\*\*kwargs*)

A component that transforms the target.

**Attributes**

<b>model_family</b>	ModelFamily.NONE
<b>modifies_features</b>	False
<b>modifies_target</b>	True

## Methods

<i>clone</i>	Constructs a new component with the same parameters and random state.
<i>default_parameters</i>	Returns the default parameters for this component.
<i>describe</i>	Describe a component and its parameters
<i>fit</i>	Fits component to data
<i>fit_transform</i>	Fits on X and transforms X
<i>inverse_transform</i>	Inverts the transformation done by the transform method.
<i>load</i>	Loads component at file path
<i>name</i>	Returns string name of this component
<i>needs_fitting</i>	Returns boolean determining if component needs fitting before
<i>parameters</i>	Returns the parameters which were used to initialize the component
<i>save</i>	Saves component at file path
<i>transform</i>	Transforms data X.

### **clone** (*self*)

Constructs a new component with the same parameters and random state.

**Returns** A new instance of this component with identical parameters and random state.

### **default\_parameters** (*cls*)

Returns the default parameters for this component.

Our convention is that `Component.default_parameters == Component().parameters`.

**Returns** default parameters for this component.

**Return type** dict

### **describe** (*self*, *print\_name=False*, *return\_dict=False*)

Describe a component and its parameters

#### Parameters

- **print\_name** (*bool*, *optional*) – whether to print name of component
- **return\_dict** (*bool*, *optional*) – whether to return description as dictionary in the format {"name": name, "parameters": parameters}

**Returns** prints and returns dictionary

**Return type** None or dict

### **fit** (*self*, *X*, *y=None*)

Fits component to data

#### Parameters

- **X** (*list, pd.DataFrame or np.ndarray*) – The input training data of shape [n\_samples, n\_features]
- **y** (*list, pd.Series, np.ndarray, optional*) – The target training data of length [n\_samples]

**Returns** self

**fit\_transform** (*self, X, y=None*)

Fits on X and transforms X

**Parameters**

- **X** (*pd.DataFrame*) – Data to fit and transform
- **y** (*pd.Series*) – Target data

**Returns** Transformed X

**Return type** pd.DataFrame

**abstract inverse\_transform** (*self, y*)

Inverts the transformation done by the transform method.

**Arguments:** y (*pd.Series*): Target transformed by this component.

**Returns** Target without the transformation.

**Return type** pd.Series

**static load** (*file\_path*)

Loads component at file path

**Parameters** **file\_path** (*str*) – Location to load file

**Returns** ComponentBase object

**property name** (*cls*)

Returns string name of this component

**needs\_fitting** (*self*)

Returns boolean determining if component needs fitting before calling predict, predict\_proba, transform, or feature\_importances. This can be overridden to False for components that do not need to be fit or whose fit methods do nothing.

**property parameters** (*self*)

Returns the parameters which were used to initialize the component

**save** (*self, file\_path, pickle\_protocol=cloudpickle.DEFAULT\_PROTOCOL*)

Saves component at file path

**Parameters**

- **file\_path** (*str*) – Location to save file
- **pickle\_protocol** (*int*) – The pickle data stream format.

**Returns** None

**transform** (*self, X, y=None*)

Transforms data X.

**Parameters**

- **X** (*pd.DataFrame*) – Data to transform.

- **y** (*pd.Series, optional*) – Target data.

**Returns** Transformed X

**Return type** `pd.DataFrame`

```
class evalml.pipelines.components.transformers.transformer.Transformer (parameters=None,  
com-  
po-  
nent_obj=None,  
ran-  
dom_seed=0,  
**kwargs)
```

A component that may or may not need fitting that transforms data. These components are used before an estimator.

To implement a new Transformer, define your own class which is a subclass of Transformer, including a name and a list of acceptable ranges for any parameters to be tuned during the automl search (hyperparameters). Define an `__init__` method which sets up any necessary state and objects. Make sure your `__init__` only uses standard keyword arguments and calls `super().__init__()` with a parameters dict. You may also override the `fit`, `transform`, `fit_transform` and other methods in this class if appropriate.

To see some examples, check out the definitions of any Transformer component.

#### Parameters

- **parameters** (*dict*) – Dictionary of parameters for the component. Defaults to None.
- **component\_obj** (*obj*) – Third-party objects useful in component implementation. Defaults to None.
- **random\_seed** (*int*) – Seed for the random number generator. Defaults to 0.

#### Attributes

<b>model_family</b>	ModelFamily.NONE
<b>modifies_features</b>	True
<b>modifies_target</b>	False

#### Methods

<code>clone</code>	Constructs a new component with the same parameters and random state.
<code>default_parameters</code>	Returns the default parameters for this component.
<code>describe</code>	Describe a component and its parameters
<code>fit</code>	Fits component to data
<code>fit_transform</code>	Fits on X and transforms X
<code>load</code>	Loads component at file path
<code>name</code>	Returns string name of this component
<code>needs_fitting</code>	Returns boolean determining if component needs fitting before
<code>parameters</code>	Returns the parameters which were used to initialize the component
<code>save</code>	Saves component at file path
<code>transform</code>	Transforms data X.

**clone** (*self*)

Constructs a new component with the same parameters and random state.

**Returns** A new instance of this component with identical parameters and random state.

**default\_parameters** (*cls*)

Returns the default parameters for this component.

Our convention is that `Component.default_parameters == Component().parameters`.

**Returns** default parameters for this component.

**Return type** dict

**describe** (*self*, *print\_name=False*, *return\_dict=False*)

Describe a component and its parameters

**Parameters**

- **print\_name** (*bool*, *optional*) – whether to print name of component
- **return\_dict** (*bool*, *optional*) – whether to return description as dictionary in the format {"name": name, "parameters": parameters}

**Returns** prints and returns dictionary

**Return type** None or dict

**fit** (*self*, *X*, *y=None*)

Fits component to data

**Parameters**

- **X** (*list*, *pd.DataFrame* or *np.ndarray*) – The input training data of shape [n\_samples, n\_features]
- **y** (*list*, *pd.Series*, *np.ndarray*, *optional*) – The target training data of length [n\_samples]

**Returns** self

**fit\_transform** (*self*, *X*, *y=None*)

Fits on X and transforms X

**Parameters**

- **X** (*pd.DataFrame*) – Data to fit and transform
- **y** (*pd.Series*) – Target data

**Returns** Transformed X

**Return type** pd.DataFrame

**static load** (*file\_path*)

Loads component at file path

**Parameters** **file\_path** (*str*) – Location to load file

**Returns** ComponentBase object

**property name** (*cls*)

Returns string name of this component

**needs\_fitting** (*self*)

Returns boolean determining if component needs fitting before calling predict, predict\_proba, transform,

or `feature_importances`. This can be overridden to `False` for components that do not need to be fit or whose fit methods do nothing.

**property** `parameters` (*self*)

Returns the parameters which were used to initialize the component

**save** (*self*, *file\_path*, *pickle\_protocol*=`cloudpickle.DEFAULT_PROTOCOL`)

Saves component at file path

**Parameters**

- **file\_path** (*str*) – Location to save file
- **pickle\_protocol** (*int*) – The pickle data stream format.

**Returns** `None`

**transform** (*self*, *X*, *y*=`None`)

Transforms data *X*.

**Parameters**

- **X** (*pd.DataFrame*) – Data to transform.
- **y** (*pd.Series*, *optional*) – Target data.

**Returns** Transformed *X*

**Return type** `pd.DataFrame`

## Package Contents

### Classes Summary

<i>DateTimeFeaturizer</i>	Transformer that can automatically extract features from datetime columns.
<i>DelayedFeatureTransformer</i>	Transformer that delays input features and target variable for time series problems.
<i>DFSTransformer</i>	Featuretools DFS component that generates features for the input features.
<i>DropColumns</i>	Drops specified columns in input data.
<i>DropNullColumns</i>	Transformer to drop features whose percentage of NaN values exceeds a specified threshold.
<i>EmailFeaturizer</i>	Transformer that can automatically extract features from emails.
<i>FeatureSelector</i>	Selects top features based on importance weights.
<i>Imputer</i>	Imputes missing data according to a specified imputation strategy.
<i>LinearDiscriminantAnalysis</i>	Reduces the number of features by using Linear Discriminant Analysis.
<i>LogTransformer</i>	Applies a log transformation to the target data.
<i>LSA</i>	Transformer to calculate the Latent Semantic Analysis Values of text input.
<i>OneHotEncoder</i>	A transformer that encodes categorical features in a one-hot numeric array.

continues on next page



Table 501 – continued from previous page

<i>PCA</i>	Reduces the number of features by using Principal Component Analysis (PCA).
<i>PerColumnImputer</i>	Imputes missing data according to a specified imputation strategy per column.
<i>PolynomialDetrender</i>	Removes trends from time series by fitting a polynomial to the data.
<i>RFClassifierSelectFromModel</i>	Selects top features based on importance weights using a Random Forest classifier.
<i>RFRegressorSelectFromModel</i>	Selects top features based on importance weights using a Random Forest regressor.
<i>SelectByType</i>	Selects columns by specified Woodwork logical type or semantic tag in input data.
<i>SelectColumns</i>	Selects specified columns in input data.
<i>SimpleImputer</i>	Imputes missing data according to a specified imputation strategy.
<i>SMOTENCOverSampler</i>	SMOTENC Oversampler component. Uses SMOTENC to generate synthetic samples. Works on a mix of numerical and categorical columns.
<i>SMOTENOverSampler</i>	SMOTEN Oversampler component. Uses SMOTEN to generate synthetic samples. Works for purely categorical datasets.
<i>SMOTEOverSampler</i>	SMOTE Oversampler component. Works on numerical datasets only. This component is only run during training and not during predict.
<i>StandardScaler</i>	A transformer that standardizes input features by removing the mean and scaling to unit variance.
<i>TargetEncoder</i>	A transformer that encodes categorical features into target encodings.
<i>TargetImputer</i>	Imputes missing target data according to a specified imputation strategy.
<i>TextFeaturizer</i>	Transformer that can automatically featurize text columns using featuretools' nlp_primitives.
<i>Transformer</i>	A component that may or may not need fitting that transforms data.
<i>Undersampler</i>	Initializes an undersampling transformer to downsample the majority classes in the dataset.
<i>URLFeaturizer</i>	Transformer that can automatically extract features from URL.

## Contents

**class** evalml.pipelines.components.transformers.**DateTimeFeaturizer** (*features\_to\_extract=None, en-  
code\_as\_categories=False, date\_index=None, ran-  
dom\_seed=0, \*\*kwargs*)

Transformer that can automatically extract features from datetime columns.

### Parameters

- **features\_to\_extract** (*list*) – List of features to extract. Valid options include “year”, “month”, “day\_of\_week”, “hour”. Defaults to None.
- **encode\_as\_categories** (*bool*) – Whether day-of-week and month features should be encoded as pandas “category” dtype. This allows OneHotEncoders to encode these features. Defaults to False.
- **date\_index** (*str*) – Name of the column containing the datetime information used to order the data. Ignored.
- **random\_seed** (*int*) – Seed for the random number generator. Defaults to 0.

#### Attributes

<b>hyper-parameter_ranges</b>	{}
<b>model_family</b>	ModelFamily.NONE
<b>modifies_features</b>	True
<b>modifies_target</b>	False
<b>name</b>	DateTime Featurization Component

#### Methods

<i>clone</i>	Constructs a new component with the same parameters and random state.
<i>default_parameters</i>	Returns the default parameters for this component.
<i>describe</i>	Describe a component and its parameters
<i>fit</i>	Fits component to data
<i>fit_transform</i>	Fits on X and transforms X
<i>get_feature_names</i>	Gets the categories of each datetime feature.
<i>load</i>	Loads component at file path
<i>needs_fitting</i>	Returns boolean determining if component needs fitting before
<i>parameters</i>	Returns the parameters which were used to initialize the component
<i>save</i>	Saves component at file path
<i>transform</i>	Transforms data X by creating new features using existing DateTime columns, and then dropping those DateTime columns

#### **clone** (*self*)

Constructs a new component with the same parameters and random state.

**Returns** A new instance of this component with identical parameters and random state.

#### **default\_parameters** (*cls*)

Returns the default parameters for this component.

Our convention is that `Component.default_parameters == Component().parameters`.

**Returns** default parameters for this component.

**Return type** dict

**describe** (*self*, *print\_name=False*, *return\_dict=False*)

Describe a component and its parameters

**Parameters**

- **print\_name** (*bool*, *optional*) – whether to print name of component
- **return\_dict** (*bool*, *optional*) – whether to return description as dictionary in the format {"name": name, "parameters": parameters}

**Returns** prints and returns dictionary

**Return type** None or dict

**fit** (*self*, *X*, *y=None*)

Fits component to data

**Parameters**

- **X** (*list*, *pd.DataFrame* or *np.ndarray*) – The input training data of shape [n\_samples, n\_features]
- **y** (*list*, *pd.Series*, *np.ndarray*, *optional*) – The target training data of length [n\_samples]

**Returns** self

**fit\_transform** (*self*, *X*, *y=None*)

Fits on X and transforms X

**Parameters**

- **X** (*pd.DataFrame*) – Data to fit and transform
- **y** (*pd.Series*) – Target data

**Returns** Transformed X

**Return type** pd.DataFrame

**get\_feature\_names** (*self*)

Gets the categories of each datetime feature.

**Returns** Dictionary, where each key-value pair is a column name and a dictionary mapping the unique feature values to their integer encoding.

**static load** (*file\_path*)

Loads component at file path

**Parameters** **file\_path** (*str*) – Location to load file

**Returns** ComponentBase object

**needs\_fitting** (*self*)

Returns boolean determining if component needs fitting before calling predict, predict\_proba, transform, or feature\_importances. This can be overridden to False for components that do not need to be fit or whose fit methods do nothing.

**property parameters** (*self*)

Returns the parameters which were used to initialize the component

**save** (*self*, *file\_path*, *pickle\_protocol=cloudpickle.DEFAULT\_PROTOCOL*)

Saves component at file path

**Parameters**

- **file\_path** (*str*) – Location to save file

- **pickle\_protocol** (*int*) – The pickle data stream format.

**Returns** None

**transform** (*self*, *X*, *y=None*)

Transforms data *X* by creating new features using existing DateTime columns, and then dropping those DateTime columns

**Parameters**

- **X** (*pd.DataFrame*) – Data to transform
- **y** (*pd.Series*, *optional*) – Ignored.

**Returns** Transformed *X*

**Return type** *pd.DataFrame*

```
class evalml.pipelines.components.transformers.DelayedFeatureTransformer (date_index=None,
                                                                    max_delay=2,
                                                                    de-
                                                                    lay_features=True,
                                                                    de-
                                                                    lay_target=True,
                                                                    gap=1,
                                                                    ran-
                                                                    dom_seed=0,
                                                                    **kwargs)
```

Transformer that delays input features and target variable for time series problems.

**Parameters**

- **date\_index** (*str*) – Name of the column containing the datetime information used to order the data. Ignored.
- **max\_delay** (*int*) – Maximum number of time units to delay each feature. Defaults to 2.
- **delay\_features** (*bool*) – Whether to delay the input features. Defaults to True.
- **delay\_target** (*bool*) – Whether to delay the target. Defaults to True.
- **gap** (*int*) – The number of time units between when the features are collected and when the target is collected. For example, if you are predicting the next time step’s target, *gap=1*. This is only needed because when *gap=0*, we need to be sure to start the lagging of the target variable at 1. Defaults to 1.
- **random\_seed** (*int*) – Seed for the random number generator. This transformer performs the same regardless of the random seed provided.

**Attributes**

<b>hyper-parameter_ranges</b>	{}
<b>model_family</b>	ModelFamily.NONE
<b>modifies_features</b>	True
<b>modifies_target</b>	False
<b>name</b>	Delayed Feature Transformer
<b>needs_fitting</b>	False

## Methods

<code>clone</code>	Constructs a new component with the same parameters and random state.
<code>default_parameters</code>	Returns the default parameters for this component.
<code>describe</code>	Describe a component and its parameters
<code>fit</code>	Fits the DelayFeatureTransformer.
<code>fit_transform</code>	Fits on X and transforms X
<code>load</code>	Loads component at file path
<code>parameters</code>	Returns the parameters which were used to initialize the component
<code>save</code>	Saves component at file path
<code>transform</code>	Computes the delayed features for all features in X and y.

**clone** (*self*)

Constructs a new component with the same parameters and random state.

**Returns** A new instance of this component with identical parameters and random state.

**default\_parameters** (*cls*)

Returns the default parameters for this component.

Our convention is that `Component.default_parameters == Component().parameters`.

**Returns** default parameters for this component.

**Return type** dict

**describe** (*self*, *print\_name=False*, *return\_dict=False*)

Describe a component and its parameters

**Parameters**

- **print\_name** (*bool*, *optional*) – whether to print name of component
- **return\_dict** (*bool*, *optional*) – whether to return description as dictionary in the format {"name": name, "parameters": parameters}

**Returns** prints and returns dictionary

**Return type** None or dict

**fit** (*self*, *X*, *y=None*)

Fits the DelayFeatureTransformer.

**Parameters**

- **X** (*pd.DataFrame* or *np.ndarray*) – The input training data of shape [n\_samples, n\_features]
- **y** (*pd.Series*, *optional*) – The target training data of length [n\_samples]

**Returns** self

**fit\_transform** (*self*, *X*, *y*)

Fits on X and transforms X

**Parameters**

- **X** (*pd.DataFrame*) – Data to fit and transform
- **y** (*pd.Series*) – Target data

**Returns** Transformed X

**Return type** `pd.DataFrame`

**static load** (*file\_path*)

Loads component at file path

**Parameters** `file_path` (*str*) – Location to load file

**Returns** ComponentBase object

**property parameters** (*self*)

Returns the parameters which were used to initialize the component

**save** (*self*, *file\_path*, *pickle\_protocol=cloudpickle.DEFAULT\_PROTOCOL*)

Saves component at file path

**Parameters**

- `file_path` (*str*) – Location to save file
- `pickle_protocol` (*int*) – The pickle data stream format.

**Returns** None

**transform** (*self*, *X*, *y=None*)

Computes the delayed features for all features in X and y.

For each feature in X, it will add a column to the output dataframe for each delay in the (inclusive) range [1, `max_delay`]. The values of each delayed feature are simply the original feature shifted forward in time by the delay amount. For example, a delay of 3 units means that the feature value at row n will be taken from the n-3rd row of that feature

If y is not None, it will also compute the delayed values for the target variable.

**Parameters**

- `X` (*pd.DataFrame* or *None*) – Data to transform. None is expected when only the target variable is being used.
- `y` (*pd.Series*, or *None*) – Target.

**Returns** Transformed X.

**Return type** `pd.DataFrame`

```
class evalml.pipelines.components.transformers.DFSTransformer (index='index',  
                                                             random_seed=0,  
                                                             **kwargs)
```

Featuretools DFS component that generates features for the input features.

**Parameters**

- `index` (*string*) – The name of the column that contains the indices. If no column with this name exists, then `featuretools.EntitySet()` creates a column with this name to serve as the index column. Defaults to 'index'.
- `random_seed` (*int*) – Seed for the random number generator. Defaults to 0.

**Attributes**

<b>hyper-parameter_ranges</b>	{}
<b>model_family</b>	ModelFamily.NONE
<b>modifies_features</b>	True
<b>modifies_target</b>	False
<b>name</b>	DFS Transformer

## Methods

<i>clone</i>	Constructs a new component with the same parameters and random state.
<i>default_parameters</i>	Returns the default parameters for this component.
<i>describe</i>	Describe a component and its parameters
<i>fit</i>	Fits the DFSTransformer Transformer component.
<i>fit_transform</i>	Fits on X and transforms X
<i>load</i>	Loads component at file path
<i>needs_fitting</i>	Returns boolean determining if component needs fitting before
<i>parameters</i>	Returns the parameters which were used to initialize the component
<i>save</i>	Saves component at file path
<i>transform</i>	Computes the feature matrix for the input X using featuretools' dfs algorithm.

### **clone** (*self*)

Constructs a new component with the same parameters and random state.

**Returns** A new instance of this component with identical parameters and random state.

### **default\_parameters** (*cls*)

Returns the default parameters for this component.

Our convention is that `Component.default_parameters == Component().parameters`.

**Returns** default parameters for this component.

**Return type** dict

### **describe** (*self*, *print\_name=False*, *return\_dict=False*)

Describe a component and its parameters

#### Parameters

- **print\_name** (*bool*, *optional*) – whether to print name of component
- **return\_dict** (*bool*, *optional*) – whether to return description as dictionary in the format {"name": name, "parameters": parameters}

**Returns** prints and returns dictionary

**Return type** None or dict

### **fit** (*self*, *X*, *y=None*)

Fits the DFSTransformer Transformer component.

**Parameters**

- **X** (*pd.DataFrame, np.array*) – The input data to transform, of shape [n\_samples, n\_features]
- **y** (*pd.Series, np.ndarray, optional*) – The target training data of length [n\_samples]

**Returns** self**fit\_transform** (*self, X, y=None*)

Fits on X and transforms X

**Parameters**

- **X** (*pd.DataFrame*) – Data to fit and transform
- **y** (*pd.Series*) – Target data

**Returns** Transformed X**Return type** pd.DataFrame**static load** (*file\_path*)

Loads component at file path

**Parameters** **file\_path** (*str*) – Location to load file**Returns** ComponentBase object**needs\_fitting** (*self*)

Returns boolean determining if component needs fitting before calling predict, predict\_proba, transform, or feature\_importances. This can be overridden to False for components that do not need to be fit or whose fit methods do nothing.

**property parameters** (*self*)

Returns the parameters which were used to initialize the component

**save** (*self, file\_path, pickle\_protocol=cloudpickle.DEFAULT\_PROTOCOL*)

Saves component at file path

**Parameters**

- **file\_path** (*str*) – Location to save file
- **pickle\_protocol** (*int*) – The pickle data stream format.

**Returns** None**transform** (*self, X, y=None*)

Computes the feature matrix for the input X using featuretools' dfs algorithm.

**Parameters**

- **X** (*pd.DataFrame or np.ndarray*) – The input training data to transform. Has shape [n\_samples, n\_features]
- **y** (*pd.Series, optional*) – Ignored.

**Returns** Feature matrix**Return type** pd.DataFrame

```
class evalml.pipelines.components.transformers.DropColumns (columns=None,  
                                                         random_seed=0,  
                                                         **kwargs)
```

Drops specified columns in input data.



### Parameters

- **columns** (*list(string)*) – List of column names, used to determine which columns to drop.
- **random\_seed** (*int*) – Seed for the random number generator. Defaults to 0.

### Attributes

<b>hyper-parameter_ranges</b>	{}
<b>model_family</b>	ModelFamily.NONE
<b>modifies_features</b>	True
<b>modifies_target</b>	False
<b>name</b>	Drop Columns Transformer
<b>needs_fitting</b>	False

### Methods

<i>clone</i>	Constructs a new component with the same parameters and random state.
<i>default_parameters</i>	Returns the default parameters for this component.
<i>describe</i>	Describe a component and its parameters
<i>fit</i>	Fits the transformer by checking if column names are present in the dataset.
<i>fit_transform</i>	Fits on X and transforms X
<i>load</i>	Loads component at file path
<i>parameters</i>	Returns the parameters which were used to initialize the component
<i>save</i>	Saves component at file path
<i>transform</i>	Transforms data X by dropping columns.

#### **clone** (*self*)

Constructs a new component with the same parameters and random state.

**Returns** A new instance of this component with identical parameters and random state.

#### **default\_parameters** (*cls*)

Returns the default parameters for this component.

Our convention is that `Component.default_parameters == Component().parameters`.

**Returns** default parameters for this component.

**Return type** dict

#### **describe** (*self*, *print\_name=False*, *return\_dict=False*)

Describe a component and its parameters

#### Parameters

- **print\_name** (*bool*, *optional*) – whether to print name of component
- **return\_dict** (*bool*, *optional*) – whether to return description as dictionary in the format {"name": name, "parameters": parameters}

**Returns** prints and returns dictionary

**Return type** None or dict

**fit** (*self*, *X*, *y=None*)

Fits the transformer by checking if column names are present in the dataset.

**Parameters**

- **X** (*pd.DataFrame*) – Data to check.
- **y** (*pd.Series*, *optional*) – Targets.

**Returns** self

**fit\_transform** (*self*, *X*, *y=None*)

Fits on X and transforms X

**Parameters**

- **X** (*pd.DataFrame*) – Data to fit and transform
- **y** (*pd.Series*) – Target data

**Returns** Transformed X

**Return type** *pd.DataFrame*

**static load** (*file\_path*)

Loads component at file path

**Parameters** **file\_path** (*str*) – Location to load file

**Returns** *ComponentBase* object

**property parameters** (*self*)

Returns the parameters which were used to initialize the component

**save** (*self*, *file\_path*, *pickle\_protocol=cloudpickle.DEFAULT\_PROTOCOL*)

Saves component at file path

**Parameters**

- **file\_path** (*str*) – Location to save file
- **pickle\_protocol** (*int*) – The pickle data stream format.

**Returns** None

**transform** (*self*, *X*, *y=None*)

Transforms data X by dropping columns.

**Parameters**

- **X** (*pd.DataFrame*) – Data to transform.
- **y** (*pd.Series*, *optional*) – Targets.

**Returns** Transformed X.

**Return type** *pd.DataFrame*

**class** evalml.pipelines.components.transformers.**DropNullColumns** (*pct\_null\_threshold=1.0*,  
*random\_seed=0*,  
*\*\*kwargs*)

Transformer to drop features whose percentage of NaN values exceeds a specified threshold.

**Parameters**

- **pct\_null\_threshold** (*float*) – The percentage of NaN values in an input feature to drop. Must be a value between [0, 1] inclusive. If equal to 0.0, will drop columns with any null values. If equal to 1.0, will drop columns with all null values. Defaults to 0.95.
- **random\_seed** (*int*) – Seed for the random number generator. Defaults to 0.

#### Attributes

<b>hyper-parameter_ranges</b>	{}
<b>model_family</b>	ModelFamily.NONE
<b>modifies_features</b>	True
<b>modifies_target</b>	False
<b>name</b>	Drop Null Columns Transformer

#### Methods

<i>clone</i>	Constructs a new component with the same parameters and random state.
<i>default_parameters</i>	Returns the default parameters for this component.
<i>describe</i>	Describe a component and its parameters
<i>fit</i>	Fits component to data
<i>fit_transform</i>	Fits on X and transforms X
<i>load</i>	Loads component at file path
<i>needs_fitting</i>	Returns boolean determining if component needs fitting before
<i>parameters</i>	Returns the parameters which were used to initialize the component
<i>save</i>	Saves component at file path
<i>transform</i>	Transforms data X by dropping columns that exceed the threshold of null values.

#### **clone** (*self*)

Constructs a new component with the same parameters and random state.

**Returns** A new instance of this component with identical parameters and random state.

#### **default\_parameters** (*cls*)

Returns the default parameters for this component.

Our convention is that `Component.default_parameters == Component().parameters`.

**Returns** default parameters for this component.

**Return type** dict

#### **describe** (*self*, *print\_name=False*, *return\_dict=False*)

Describe a component and its parameters

#### Parameters

- **print\_name** (*bool*, *optional*) – whether to print name of component
- **return\_dict** (*bool*, *optional*) – whether to return description as dictionary in the format {"name": name, "parameters": parameters}

**Returns** prints and returns dictionary

**Return type** None or dict

**fit** (*self*, *X*, *y=None*)

Fits component to data

**Parameters**

- **X** (*list*, *pd.DataFrame* or *np.ndarray*) – The input training data of shape [n\_samples, n\_features]
- **y** (*list*, *pd.Series*, *np.ndarray*, *optional*) – The target training data of length [n\_samples]

**Returns** self

**fit\_transform** (*self*, *X*, *y=None*)

Fits on X and transforms X

**Parameters**

- **X** (*pd.DataFrame*) – Data to fit and transform
- **y** (*pd.Series*) – Target data

**Returns** Transformed X

**Return type** pd.DataFrame

**static load** (*file\_path*)

Loads component at file path

**Parameters** **file\_path** (*str*) – Location to load file

**Returns** ComponentBase object

**needs\_fitting** (*self*)

Returns boolean determining if component needs fitting before calling predict, predict\_proba, transform, or feature\_importances. This can be overridden to False for components that do not need to be fit or whose fit methods do nothing.

**property parameters** (*self*)

Returns the parameters which were used to initialize the component

**save** (*self*, *file\_path*, *pickle\_protocol=cloudpickle.DEFAULT\_PROTOCOL*)

Saves component at file path

**Parameters**

- **file\_path** (*str*) – Location to save file
- **pickle\_protocol** (*int*) – The pickle data stream format.

**Returns** None

**transform** (*self*, *X*, *y=None*)

Transforms data X by dropping columns that exceed the threshold of null values.

**Parameters**

- **X** (*pd.DataFrame*) – Data to transform
- **y** (*pd.Series*, *optional*) – Ignored.

**Returns** Transformed X

**Return type** pd.DataFrame

```
class evalml.pipelines.components.transformers.EmailFeaturizer (random_seed=0,  
                                                         **kwargs)
```

Transformer that can automatically extract features from emails.

**Parameters** **random\_seed** (*int*) – Seed for the random number generator. Defaults to 0.

#### Attributes

<b>hyper-parameter_ranges</b>	{}
<b>model_family</b>	ModelFamily.NONE
<b>modifies_features</b>	True
<b>modifies_target</b>	False
<b>name</b>	Email Featurizer

#### Methods

<i>clone</i>	Constructs a new component with the same parameters and random state.
<i>default_parameters</i>	Returns the default parameters for this component.
<i>describe</i>	Describe a component and its parameters
<i>fit</i>	Fits component to data
<i>fit_transform</i>	Fits on X and transforms X
<i>load</i>	Loads component at file path
<i>needs_fitting</i>	Returns boolean determining if component needs fitting before
<i>parameters</i>	Returns the parameters which were used to initialize the component
<i>save</i>	Saves component at file path
<i>transform</i>	Transforms data X.

**clone** (*self*)

Constructs a new component with the same parameters and random state.

**Returns** A new instance of this component with identical parameters and random state.

**default\_parameters** (*cls*)

Returns the default parameters for this component.

Our convention is that `Component.default_parameters == Component().parameters`.

**Returns** default parameters for this component.

**Return type** dict

**describe** (*self*, *print\_name=False*, *return\_dict=False*)

Describe a component and its parameters

#### Parameters

- **print\_name** (*bool*, *optional*) – whether to print name of component
- **return\_dict** (*bool*, *optional*) – whether to return description as dictionary in the format {"name": name, "parameters": parameters}

**Returns** prints and returns dictionary

**Return type** None or dict

**fit** (*self*, *X*, *y=None*)

Fits component to data

**Parameters**

- **X** (*list*, *pd.DataFrame* or *np.ndarray*) – The input training data of shape [n\_samples, n\_features]
- **y** (*list*, *pd.Series*, *np.ndarray*, *optional*) – The target training data of length [n\_samples]

**Returns** self

**fit\_transform** (*self*, *X*, *y=None*)

Fits on X and transforms X

**Parameters**

- **X** (*pd.DataFrame*) – Data to fit and transform
- **y** (*pd.Series*) – Target data

**Returns** Transformed X

**Return type** pd.DataFrame

**static load** (*file\_path*)

Loads component at file path

**Parameters** **file\_path** (*str*) – Location to load file

**Returns** ComponentBase object

**needs\_fitting** (*self*)

Returns boolean determining if component needs fitting before calling predict, predict\_proba, transform, or feature\_importances. This can be overridden to False for components that do not need to be fit or whose fit methods do nothing.

**property parameters** (*self*)

Returns the parameters which were used to initialize the component

**save** (*self*, *file\_path*, *pickle\_protocol=cloudpickle.DEFAULT\_PROTOCOL*)

Saves component at file path

**Parameters**

- **file\_path** (*str*) – Location to save file
- **pickle\_protocol** (*int*) – The pickle data stream format.

**Returns** None

**transform** (*self*, *X*, *y=None*)

Transforms data X.

**Parameters**

- **X** (*pd.DataFrame*) – Data to transform.
- **y** (*pd.Series*, *optional*) – Target data.

**Returns** Transformed X

**Return type** pd.DataFrame

```
class evalml.pipelines.components.transformers.FeatureSelector (parameters=None,  

component_obj=None,  

random_seed=0,  

**kwargs)
```

Selects top features based on importance weights.

#### Parameters

- **parameters** (*dict*) – Dictionary of parameters for the component. Defaults to None.
- **component\_obj** (*obj*) – Third-party objects useful in component implementation. Defaults to None.
- **random\_seed** (*int*) – Seed for the random number generator. Defaults to 0.

#### Attributes

<b>model_family</b>	ModelFamily.NONE
<b>modifies_features</b>	True
<b>modifies_target</b>	False

#### Methods

<i>clone</i>	Constructs a new component with the same parameters and random state.
<i>default_parameters</i>	Returns the default parameters for this component.
<i>describe</i>	Describe a component and its parameters
<i>fit</i>	Fits component to data
<i>fit_transform</i>	Fits on X and transforms X
<i>get_names</i>	Get names of selected features.
<i>load</i>	Loads component at file path
<i>name</i>	Returns string name of this component
<i>needs_fitting</i>	Returns boolean determining if component needs fitting before
<i>parameters</i>	Returns the parameters which were used to initialize the component
<i>save</i>	Saves component at file path
<i>transform</i>	Transforms input data by selecting features. If the component_obj does not have a transform method, will raise an MethodPropertyNotFoundError exception.

#### **clone** (*self*)

Constructs a new component with the same parameters and random state.

**Returns** A new instance of this component with identical parameters and random state.

#### **default\_parameters** (*cls*)

Returns the default parameters for this component.

Our convention is that `Component.default_parameters == Component().parameters`.

**Returns** default parameters for this component.

**Return type** dict

**describe** (*self*, *print\_name=False*, *return\_dict=False*)

Describe a component and its parameters

**Parameters**

- **print\_name** (*bool*, *optional*) – whether to print name of component
- **return\_dict** (*bool*, *optional*) – whether to return description as dictionary in the format {"name": name, "parameters": parameters}

**Returns** prints and returns dictionary

**Return type** None or dict

**fit** (*self*, *X*, *y=None*)

Fits component to data

**Parameters**

- **X** (*list*, *pd.DataFrame* or *np.ndarray*) – The input training data of shape [n\_samples, n\_features]
- **y** (*list*, *pd.Series*, *np.ndarray*, *optional*) – The target training data of length [n\_samples]

**Returns** self

**fit\_transform** (*self*, *X*, *y=None*)

Fits on X and transforms X

**Parameters**

- **X** (*pd.DataFrame*) – Data to fit and transform
- **y** (*pd.Series*) – Target data

**Returns** Transformed X

**Return type** pd.DataFrame

**get\_names** (*self*)

Get names of selected features.

**Returns** List of the names of features selected

**Return type** list[str]

**static load** (*file\_path*)

Loads component at file path

**Parameters** **file\_path** (*str*) – Location to load file

**Returns** ComponentBase object

**property name** (*cls*)

Returns string name of this component

**needs\_fitting** (*self*)

Returns boolean determining if component needs fitting before calling predict, predict\_proba, transform, or feature\_importances. This can be overridden to False for components that do not need to be fit or whose fit methods do nothing.

**property parameters** (*self*)

Returns the parameters which were used to initialize the component



**save** (*self*, *file\_path*, *pickle\_protocol*=*cloudpickle.DEFAULT\_PROTOCOL*)

Saves component at file path

#### Parameters

- **file\_path** (*str*) – Location to save file
- **pickle\_protocol** (*int*) – The pickle data stream format.

**Returns** None

**transform** (*self*, *X*, *y=None*)

Transforms input data by selecting features. If the component\_obj does not have a transform method, will raise an `MethodPropertyNotFoundError` exception.

#### Parameters

- **X** (*pd.DataFrame*) – Data to transform.
- **y** (*pd.Series*, *optional*) – Target data. Ignored.

**Returns** Transformed X

**Return type** `pd.DataFrame`

```
class evalml.pipelines.components.transformers.Imputer (categorical_impute_strategy='most_frequent',
                                                    categori-
                                                    cal_fill_value=None, nu-
                                                    meric_impute_strategy='mean',
                                                    numeric_fill_value=None,
                                                    random_seed=0, **kwargs)
```

Imputes missing data according to a specified imputation strategy.

#### Parameters

- **categorical\_impute\_strategy** (*string*) – Impute strategy to use for string, object, boolean, categorical dtypes. Valid values include “most\_frequent” and “constant”.
- **numeric\_impute\_strategy** (*string*) – Impute strategy to use for numeric columns. Valid values include “mean”, “median”, “most\_frequent”, and “constant”.
- **categorical\_fill\_value** (*string*) – When `categorical_impute_strategy == “constant”`, `fill_value` is used to replace missing data. The default value of None will fill with the string “missing\_value”.
- **numeric\_fill\_value** (*int*, *float*) – When `numeric_impute_strategy == “constant”`, `fill_value` is used to replace missing data. The default value of None will fill with 0.
- **random\_seed** (*int*) – Seed for the random number generator. Defaults to 0.

#### Attributes

<b>hyper-parameter_ranges</b>	{ “categorical_impute_strategy”: [“most_frequent”], “numeric_impute_strategy”: [“mean”, “median”, “most_frequent”], }
<b>model_family</b>	ModelFamily.NONE
<b>modifies_features</b>	True
<b>modifies_target</b>	False
<b>name</b>	Imputer

## Methods

<code>clone</code>	Constructs a new component with the same parameters and random state.
<code>default_parameters</code>	Returns the default parameters for this component.
<code>describe</code>	Describe a component and its parameters
<code>fit</code>	Fits imputer to data. 'None' values are converted to np.nan before imputation and are
<code>fit_transform</code>	Fits on X and transforms X
<code>load</code>	Loads component at file path
<code>needs_fitting</code>	Returns boolean determining if component needs fitting before
<code>parameters</code>	Returns the parameters which were used to initialize the component
<code>save</code>	Saves component at file path
<code>transform</code>	Transforms data X by imputing missing values. 'None' values are converted to np.nan before imputation and are

### **clone** (*self*)

Constructs a new component with the same parameters and random state.

**Returns** A new instance of this component with identical parameters and random state.

### **default\_parameters** (*cls*)

Returns the default parameters for this component.

Our convention is that `Component.default_parameters == Component().parameters`.

**Returns** default parameters for this component.

**Return type** dict

### **describe** (*self*, *print\_name=False*, *return\_dict=False*)

Describe a component and its parameters

#### Parameters

- **print\_name** (*bool*, *optional*) – whether to print name of component
- **return\_dict** (*bool*, *optional*) – whether to return description as dictionary in the format {"name": name, "parameters": parameters}

**Returns** prints and returns dictionary

**Return type** None or dict

### **fit** (*self*, *X*, *y=None*)

**Fits imputer to data. 'None' values are converted to np.nan before imputation and are** treated as the same.

#### Parameters

- **X** (*pd.DataFrame*, *np.ndarray*) – The input training data of shape [n\_samples, n\_features]
- **y** (*pd.Series*, *optional*) – The target training data of length [n\_samples]

**Returns** self

**fit\_transform** (*self*, *X*, *y=None*)

Fits on *X* and transforms *X*

**Parameters**

- **X** (*pd.DataFrame*) – Data to fit and transform
- **y** (*pd.Series*) – Target data

**Returns** Transformed *X*

**Return type** *pd.DataFrame*

**static load** (*file\_path*)

Loads component at file path

**Parameters** **file\_path** (*str*) – Location to load file

**Returns** *ComponentBase* object

**needs\_fitting** (*self*)

Returns boolean determining if component needs fitting before calling *predict*, *predict\_proba*, *transform*, or *feature\_importances*. This can be overridden to *False* for components that do not need to be fit or whose fit methods do nothing.

**property parameters** (*self*)

Returns the parameters which were used to initialize the component

**save** (*self*, *file\_path*, *pickle\_protocol=cloudpickle.DEFAULT\_PROTOCOL*)

Saves component at file path

**Parameters**

- **file\_path** (*str*) – Location to save file
- **pickle\_protocol** (*int*) – The pickle data stream format.

**Returns** *None*

**transform** (*self*, *X*, *y=None*)

Transforms data *X* by imputing missing values. ‘None’ values are converted to *np.nan* before imputation and are treated as the same.

**Parameters**

- **X** (*pd.DataFrame*) – Data to transform
- **y** (*pd.Series*, *optional*) – Ignored.

**Returns** Transformed *X*

**Return type** *pd.DataFrame*

**class** *evalml.pipelines.components.transformers.LinearDiscriminantAnalysis* (*n\_components=None*,  
*random\_seed=0*,  
*\*\*kwargs*)

Reduces the number of features by using Linear Discriminant Analysis.

**Parameters**

- **n\_components** (*int*) – The number of features to maintain after computation. Defaults to *None*.
- **random\_seed** (*int*) – Seed for the random number generator. Defaults to 0.

### Attributes

<b>hyper-parameter_ranges</b>	{}
<b>model_family</b>	ModelFamily.NONE
<b>modifies_features</b>	True
<b>modifies_target</b>	False
<b>name</b>	Linear Discriminant Analysis Transformer

### Methods

<i>clone</i>	Constructs a new component with the same parameters and random state.
<i>default_parameters</i>	Returns the default parameters for this component.
<i>describe</i>	Describe a component and its parameters
<i>fit</i>	Fits component to data
<i>fit_transform</i>	Fits on X and transforms X
<i>load</i>	Loads component at file path
<i>needs_fitting</i>	Returns boolean determining if component needs fitting before
<i>parameters</i>	Returns the parameters which were used to initialize the component
<i>save</i>	Saves component at file path
<i>transform</i>	Transforms data X.

#### **clone** (*self*)

Constructs a new component with the same parameters and random state.

**Returns** A new instance of this component with identical parameters and random state.

#### **default\_parameters** (*cls*)

Returns the default parameters for this component.

Our convention is that `Component.default_parameters == Component().parameters`.

**Returns** default parameters for this component.

**Return type** dict

#### **describe** (*self*, *print\_name=False*, *return\_dict=False*)

Describe a component and its parameters

##### Parameters

- **print\_name** (*bool*, *optional*) – whether to print name of component
- **return\_dict** (*bool*, *optional*) – whether to return description as dictionary in the format {"name": name, "parameters": parameters}

**Returns** prints and returns dictionary

**Return type** None or dict

#### **fit** (*self*, *X*, *y*)

Fits component to data

**Parameters**

- **X** (*list, pd.DataFrame or np.ndarray*) – The input training data of shape [n\_samples, n\_features]
- **y** (*list, pd.Series, np.ndarray, optional*) – The target training data of length [n\_samples]

**Returns** self**fit\_transform** (*self, X, y=None*)

Fits on X and transforms X

**Parameters**

- **X** (*pd.DataFrame*) – Data to fit and transform
- **y** (*pd.Series*) – Target data

**Returns** Transformed X**Return type** pd.DataFrame**static load** (*file\_path*)

Loads component at file path

**Parameters** **file\_path** (*str*) – Location to load file**Returns** ComponentBase object**needs\_fitting** (*self*)

Returns boolean determining if component needs fitting before calling predict, predict\_proba, transform, or feature\_importances. This can be overridden to False for components that do not need to be fit or whose fit methods do nothing.

**property parameters** (*self*)

Returns the parameters which were used to initialize the component

**save** (*self, file\_path, pickle\_protocol=cloudpickle.DEFAULT\_PROTOCOL*)

Saves component at file path

**Parameters**

- **file\_path** (*str*) – Location to save file
- **pickle\_protocol** (*int*) – The pickle data stream format.

**Returns** None**transform** (*self, X, y=None*)

Transforms data X.

**Parameters**

- **X** (*pd.DataFrame*) – Data to transform.
- **y** (*pd.Series, optional*) – Target data.

**Returns** Transformed X**Return type** pd.DataFrame**class** evalml.pipelines.components.transformers.**LogTransformer** (*random\_seed=0*)

Applies a log transformation to the target data.

**Attributes**

<b>hyper-parameter_ranges</b>	{}
<b>model_family</b>	ModelFamily.NONE
<b>modifies_features</b>	False
<b>modifies_target</b>	True
<b>name</b>	Log Transformer

## Methods

<i>clone</i>	Constructs a new component with the same parameters and random state.
<i>default_parameters</i>	Returns the default parameters for this component.
<i>describe</i>	Describe a component and its parameters
<i>fit</i>	Fits the LogTransformer.
<i>fit_transform</i>	Log transforms the target variable.
<i>inverse_transform</i>	Inverts the transformation done by the transform method.
<i>load</i>	Loads component at file path
<i>needs_fitting</i>	Returns boolean determining if component needs fitting before
<i>parameters</i>	Returns the parameters which were used to initialize the component
<i>save</i>	Saves component at file path
<i>transform</i>	Log transforms the target variable.

### **clone** (*self*)

Constructs a new component with the same parameters and random state.

**Returns** A new instance of this component with identical parameters and random state.

### **default\_parameters** (*cls*)

Returns the default parameters for this component.

Our convention is that `Component.default_parameters == Component().parameters`.

**Returns** default parameters for this component.

**Return type** dict

### **describe** (*self*, *print\_name=False*, *return\_dict=False*)

Describe a component and its parameters

#### Parameters

- **print\_name** (*bool*, *optional*) – whether to print name of component
- **return\_dict** (*bool*, *optional*) – whether to return description as dictionary in the format {"name": name, "parameters": parameters}

**Returns** prints and returns dictionary

**Return type** None or dict

### **fit** (*self*, *X*, *y=None*)

Fits the LogTransformer.

**Parameters**

- **X** (*pd.DataFrame* or *np.ndarray*) – Ignored.
- **y** (*pd.Series*, *optional*) – Ignored.

**Returns** *self*

**fit\_transform** (*self*, *X*, *y=None*)

Log transforms the target variable.

**Parameters**

- **X** (*pd.DataFrame*, *optional*) – Ignored.
- **y** (*pd.Series*) – Target variable to log transform.

**Returns**

**The input features are returned without modification. The target** variable *y* is log transformed.

**Return type** tuple of *pd.DataFrame*, *pd.Series*

**inverse\_transform** (*self*, *y*)

Inverts the transformation done by the transform method.

**Arguments:** *y* (*pd.Series*): Target transformed by this component.

**Returns** Target without the transformation.

**Return type** *pd.Series*∅

**static load** (*file\_path*)

Loads component at file path

**Parameters** **file\_path** (*str*) – Location to load file

**Returns** *ComponentBase* object

**needs\_fitting** (*self*)

Returns boolean determining if component needs fitting before calling *predict*, *predict\_proba*, *transform*, or *feature\_importances*. This can be overridden to *False* for components that do not need to be fit or whose fit methods do nothing.

**property parameters** (*self*)

Returns the parameters which were used to initialize the component

**save** (*self*, *file\_path*, *pickle\_protocol=cloudpickle.DEFAULT\_PROTOCOL*)

Saves component at file path

**Parameters**

- **file\_path** (*str*) – Location to save file
- **pickle\_protocol** (*int*) – The pickle data stream format.

**Returns** *None*

**transform** (*self*, *X*, *y=None*)

Log transforms the target variable.

**Parameters**

- **X** (*pd.DataFrame*, *optional*) – Ignored.
- **y** (*pd.Series*) – Target data to log transform.

**Returns**

The input features are returned without modification. The target variable `y` is log transformed.

**Return type** tuple of `pd.DataFrame`, `pd.Series`

**class** `evalml.pipelines.components.transformers.LSA` (*random\_seed=0, \*\*kwargs*)

Transformer to calculate the Latent Semantic Analysis Values of text input.

**Parameters** `random_seed` (*int*) – Seed for the random number generator. Defaults to 0.

**Attributes**

<b>hyper-parameter_ranges</b>	{}
<b>model_family</b>	ModelFamily.NONE
<b>modifies_features</b>	True
<b>modifies_target</b>	False
<b>name</b>	LSA Transformer

**Methods**

<code>clone</code>	Constructs a new component with the same parameters and random state.
<code>default_parameters</code>	Returns the default parameters for this component.
<code>describe</code>	Describe a component and its parameters
<code>fit</code>	Fits component to data
<code>fit_transform</code>	Fits on <code>X</code> and transforms <code>X</code>
<code>load</code>	Loads component at file path
<code>needs_fitting</code>	Returns boolean determining if component needs fitting before
<code>parameters</code>	Returns the parameters which were used to initialize the component
<code>save</code>	Saves component at file path
<code>transform</code>	Transforms data <code>X</code> by applying the LSA pipeline.

**clone** (*self*)

Constructs a new component with the same parameters and random state.

**Returns** A new instance of this component with identical parameters and random state.

**default\_parameters** (*cls*)

Returns the default parameters for this component.

Our convention is that `Component.default_parameters == Component().parameters`.

**Returns** default parameters for this component.

**Return type** dict

**describe** (*self*, *print\_name=False*, *return\_dict=False*)

Describe a component and its parameters

**Parameters**



- **print\_name** (*bool, optional*) – whether to print name of component
- **return\_dict** (*bool, optional*) – whether to return description as dictionary in the format {"name": name, "parameters": parameters}

**Returns** prints and returns dictionary

**Return type** None or dict

**fit** (*self, X, y=None*)

Fits component to data

**Parameters**

- **X** (*list, pd.DataFrame or np.ndarray*) – The input training data of shape [n\_samples, n\_features]
- **y** (*list, pd.Series, np.ndarray, optional*) – The target training data of length [n\_samples]

**Returns** self

**fit\_transform** (*self, X, y=None*)

Fits on X and transforms X

**Parameters**

- **X** (*pd.DataFrame*) – Data to fit and transform
- **y** (*pd.Series*) – Target data

**Returns** Transformed X

**Return type** pd.DataFrame

**static load** (*file\_path*)

Loads component at file path

**Parameters** **file\_path** (*str*) – Location to load file

**Returns** ComponentBase object

**needs\_fitting** (*self*)

Returns boolean determining if component needs fitting before calling predict, predict\_proba, transform, or feature\_importances. This can be overridden to False for components that do not need to be fit or whose fit methods do nothing.

**property parameters** (*self*)

Returns the parameters which were used to initialize the component

**save** (*self, file\_path, pickle\_protocol=cloudpickle.DEFAULT\_PROTOCOL*)

Saves component at file path

**Parameters**

- **file\_path** (*str*) – Location to save file
- **pickle\_protocol** (*int*) – The pickle data stream format.

**Returns** None

**transform** (*self, X, y=None*)

Transforms data X by applying the LSA pipeline.

**Parameters**

- **X** (*pd.DataFrame*) – The data to transform.

- `y` (`pd.Series`, *optional*) – Ignored.

#### Returns

**Transformed X.** The original column is removed and replaced with two columns of the format `LSA(original_column_name)[feature_number]`, where `feature_number` is 0 or 1.

**Return type** `pd.DataFrame`

```
class evalml.pipelines.components.transformers.OneHotEncoder (top_n=10, features_to_encode=None, categories=None, drop='if_binary', handle_unknown='ignore', handle_missing='error', random_seed=0, **kwargs)
```

A transformer that encodes categorical features in a one-hot numeric array.

#### Parameters

- **`top_n`** (*int*) – Number of categories per column to encode. If `None`, all categories will be encoded. Otherwise, the *n* most frequent will be encoded and all others will be dropped. Defaults to 10.
- **`features_to_encode`** (*list[str]*) – List of columns to encode. All other columns will remain untouched. If `None`, all appropriate columns will be encoded. Defaults to `None`.
- **`categories`** (*list*) – A two dimensional list of categories, where `categories[i]` is a list of the categories for the column at index *i*. This can also be `None`, or “*auto*” if `top_n` is not `None`. Defaults to `None`.
- **`drop`** (*string*, *list*) – Method (“first” or “if\_binary”) to use to drop one category per feature. Can also be a list specifying which categories to drop for each feature. Defaults to “if\_binary”.
- **`handle_unknown`** (*string*) – Whether to ignore or error for unknown categories for a feature encountered during *fit* or *transform*. If either `top_n` or `categories` is used to limit the number of categories per column, this must be “ignore”. Defaults to “ignore”.
- **`handle_missing`** (*string*) – Options for how to handle missing (NaN) values encountered during *fit* or *transform*. If this is set to “as\_category” and NaN values are within the *n* most frequent, “nan” values will be encoded as their own column. If this is set to “error”, any missing values encountered will raise an error. Defaults to “error”.
- **`random_seed`** (*int*) – Seed for the random number generator. Defaults to 0.

#### Attributes

<b>hyper-parameter_ranges</b>	{ }
<b>model_family</b>	ModelFamily.NONE
<b>modifies_features</b>	True
<b>modifies_target</b>	False
<b>name</b>	One Hot Encoder

**Methods**

<code>categories</code>	Returns a list of the unique categories to be encoded for the particular feature, in order.
<code>clone</code>	Constructs a new component with the same parameters and random state.
<code>default_parameters</code>	Returns the default parameters for this component.
<code>describe</code>	Describe a component and its parameters
<code>fit</code>	Fits component to data
<code>fit_transform</code>	Fits on $X$ and transforms $X$
<code>get_feature_names</code>	Return feature names for the categorical features after fitting.
<code>load</code>	Loads component at file path
<code>needs_fitting</code>	Returns boolean determining if component needs fitting before
<code>parameters</code>	Returns the parameters which were used to initialize the component
<code>save</code>	Saves component at file path
<code>transform</code>	One-hot encode the input data.

**categories** (*self*, *feature\_name*)

Returns a list of the unique categories to be encoded for the particular feature, in order.

**Parameters** **feature\_name** (*str*) – the name of any feature provided to one-hot encoder during fit

**Returns** the unique categories, in the same dtype as they were provided during fit

**Return type** `np.ndarray`

**clone** (*self*)

Constructs a new component with the same parameters and random state.

**Returns** A new instance of this component with identical parameters and random state.

**default\_parameters** (*cls*)

Returns the default parameters for this component.

Our convention is that `Component.default_parameters == Component().parameters`.

**Returns** default parameters for this component.

**Return type** `dict`

**describe** (*self*, *print\_name=False*, *return\_dict=False*)

Describe a component and its parameters

**Parameters**

- **print\_name** (*bool*, *optional*) – whether to print name of component
- **return\_dict** (*bool*, *optional*) – whether to return description as dictionary in the format {"name": name, "parameters": parameters}

**Returns** prints and returns dictionary

**Return type** `None` or `dict`

**fit** (*self*,  $X$ , *y=None*)

Fits component to data

**Parameters**

- **X** (*list*, *pd.DataFrame* or *np.ndarray*) – The input training data of shape [n\_samples, n\_features]
- **y** (*list*, *pd.Series*, *np.ndarray*, *optional*) – The target training data of length [n\_samples]

**Returns** self**fit\_transform** (*self*, *X*, *y=None*)

Fits on X and transforms X

**Parameters**

- **X** (*pd.DataFrame*) – Data to fit and transform
- **y** (*pd.Series*) – Target data

**Returns** Transformed X**Return type** *pd.DataFrame***get\_feature\_names** (*self*)

Return feature names for the categorical features after fitting.

Feature names are formatted as {column name}\_{category name}. In the event of a duplicate name, an integer will be added at the end of the feature name to distinguish it.

For example, consider a dataframe with a column called “A” and category “x\_y” and another column called “A\_x” with “y”. In this example, the feature names would be “A\_x\_y” and “A\_x\_y\_1”.

**Returns** The feature names after encoding, provided in the same order as input\_features.**Return type** *np.ndarray***static load** (*file\_path*)

Loads component at file path

**Parameters** **file\_path** (*str*) – Location to load file**Returns** ComponentBase object**needs\_fitting** (*self*)

Returns boolean determining if component needs fitting before calling predict, predict\_proba, transform, or feature\_importances. This can be overridden to False for components that do not need to be fit or whose fit methods do nothing.

**property parameters** (*self*)

Returns the parameters which were used to initialize the component

**save** (*self*, *file\_path*, *pickle\_protocol=cloudpickle.DEFAULT\_PROTOCOL*)

Saves component at file path

**Parameters**

- **file\_path** (*str*) – Location to save file
- **pickle\_protocol** (*int*) – The pickle data stream format.

**Returns** None**transform** (*self*, *X*, *y=None*)

One-hot encode the input data.

**Parameters**

- **X** (*pd.DataFrame*) – Features to one-hot encode.
- **y** (*pd.Series*) – Ignored.

**Returns** Transformed data, where each categorical feature has been encoded into numerical columns using one-hot encoding.

**Return type** *pd.DataFrame*

**class** `evalml.pipelines.components.transformers.PCA` (*variance=0.95*,  
*n\_components=None*, *random\_seed=0*, *\*\*kwargs*)

Reduces the number of features by using Principal Component Analysis (PCA).

#### Parameters

- **variance** (*float*) – The percentage of the original data variance that should be preserved when reducing the number of features. Defaults to 0.95.
- **n\_components** (*int*) – The number of features to maintain after computing SVD. Defaults to None, but will override variance variable if set.
- **random\_seed** (*int*) – Seed for the random number generator. Defaults to 0.

#### Attributes

<b>hyper-parameter_ranges</b>	Real(0.25, 1)}:type: {"variance"}
<b>model_family</b>	ModelFamily.NONE
<b>modifies_features</b>	True
<b>modifies_target</b>	False
<b>name</b>	PCA Transformer

#### Methods

<i>clone</i>	Constructs a new component with the same parameters and random state.
<i>default_parameters</i>	Returns the default parameters for this component.
<i>describe</i>	Describe a component and its parameters
<i>fit</i>	Fits component to data
<i>fit_transform</i>	Fits on X and transforms X
<i>load</i>	Loads component at file path
<i>needs_fitting</i>	Returns boolean determining if component needs fitting before
<i>parameters</i>	Returns the parameters which were used to initialize the component
<i>save</i>	Saves component at file path
<i>transform</i>	Transforms data X.

**clone** (*self*)

Constructs a new component with the same parameters and random state.

**Returns** A new instance of this component with identical parameters and random state.

**default\_parameters** (*cls*)

Returns the default parameters for this component.

Our convention is that `Component.default_parameters == Component().parameters`.

**Returns** default parameters for this component.

**Return type** dict

**describe** (*self*, *print\_name=False*, *return\_dict=False*)

Describe a component and its parameters

**Parameters**

- **print\_name** (*bool*, *optional*) – whether to print name of component
- **return\_dict** (*bool*, *optional*) – whether to return description as dictionary in the format {"name": name, "parameters": parameters}

**Returns** prints and returns dictionary

**Return type** None or dict

**fit** (*self*, *X*, *y=None*)

Fits component to data

**Parameters**

- **X** (*list*, *pd.DataFrame* or *np.ndarray*) – The input training data of shape [n\_samples, n\_features]
- **y** (*list*, *pd.Series*, *np.ndarray*, *optional*) – The target training data of length [n\_samples]

**Returns** self

**fit\_transform** (*self*, *X*, *y=None*)

Fits on X and transforms X

**Parameters**

- **X** (*pd.DataFrame*) – Data to fit and transform
- **y** (*pd.Series*) – Target data

**Returns** Transformed X

**Return type** pd.DataFrame

**static load** (*file\_path*)

Loads component at file path

**Parameters** **file\_path** (*str*) – Location to load file

**Returns** ComponentBase object

**needs\_fitting** (*self*)

Returns boolean determining if component needs fitting before calling predict, predict\_proba, transform, or feature\_importances. This can be overridden to False for components that do not need to be fit or whose fit methods do nothing.

**property parameters** (*self*)

Returns the parameters which were used to initialize the component

**save** (*self*, *file\_path*, *pickle\_protocol=cloudpickle.DEFAULT\_PROTOCOL*)

Saves component at file path

**Parameters**

- **file\_path** (*str*) – Location to save file
- **pickle\_protocol** (*int*) – The pickle data stream format.

**Returns** None

**transform** (*self*, *X*, *y=None*)

Transforms data X.

**Parameters**

- **X** (*pd.DataFrame*) – Data to transform.
- **y** (*pd.Series*, *optional*) – Target data.

**Returns** Transformed X

**Return type** *pd.DataFrame*

```
class evalml.pipelines.components.transformers.PerColumnImputer (impute_strategies=None,
                                                                de-
                                                                fault_impute_strategy='most_frequent',
                                                                ran-
                                                                dom_seed=0,
                                                                **kwargs)
```

Imputes missing data according to a specified imputation strategy per column.

**Parameters**

- **impute\_strategies** (*dict*) – Column and {"impute\_strategy": strategy, "fill\_value":value} pairings. Valid values for impute strategy include "mean", "median", "most\_frequent", "constant" for numerical data, and "most\_frequent", "constant" for object data types. Defaults to None, which uses "most\_frequent" for all columns. When impute\_strategy == "constant", fill\_value is used to replace missing data. When None, uses 0 when imputing numerical data and "missing\_value" for strings or object data types.
- **default\_impute\_strategy** (*str*) – Impute strategy to fall back on when none is provided for a certain column. Valid values include "mean", "median", "most\_frequent", "constant" for numerical data, and "most\_frequent", "constant" for object data types. Defaults to "most\_frequent".
- **random\_seed** (*int*) – Seed for the random number generator. Defaults to 0.

**Attributes**

<b>hyper-parameter_ranges</b>	{}
<b>model_family</b>	ModelFamily.NONE
<b>modifies_features</b>	True
<b>modifies_target</b>	False
<b>name</b>	Per Column Imputer

**Methods**

<i>clone</i>	Constructs a new component with the same parameters and random state.
--------------	---

continues on next page

Table 515 – continued from previous page

<code>default_parameters</code>	Returns the default parameters for this component.
<code>describe</code>	Describe a component and its parameters
<code>fit</code>	Fits imputers on input data
<code>fit_transform</code>	Fits on X and transforms X
<code>load</code>	Loads component at file path
<code>needs_fitting</code>	Returns boolean determining if component needs fitting before
<code>parameters</code>	Returns the parameters which were used to initialize the component
<code>save</code>	Saves component at file path
<code>transform</code>	Transforms input data by imputing missing values.

**clone** (*self*)

Constructs a new component with the same parameters and random state.

**Returns** A new instance of this component with identical parameters and random state.

**default\_parameters** (*cls*)

Returns the default parameters for this component.

Our convention is that `Component.default_parameters == Component().parameters`.

**Returns** default parameters for this component.

**Return type** dict

**describe** (*self*, *print\_name=False*, *return\_dict=False*)

Describe a component and its parameters

**Parameters**

- **print\_name** (*bool*, *optional*) – whether to print name of component
- **return\_dict** (*bool*, *optional*) – whether to return description as dictionary in the format {"name": name, "parameters": parameters}

**Returns** prints and returns dictionary

**Return type** None or dict

**fit** (*self*, *X*, *y=None*)

Fits imputers on input data

**Parameters**

- **X** (*pd.DataFrame* or *np.ndarray*) – The input training data of shape [n\_samples, n\_features] to fit.
- **y** (*pd.Series*, *optional*) – The target training data of length [n\_samples]. Ignored.

**Returns** self

**fit\_transform** (*self*, *X*, *y=None*)

Fits on X and transforms X

**Parameters**

- **X** (*pd.DataFrame*) – Data to fit and transform
- **y** (*pd.Series*) – Target data

**Returns** Transformed X



**Return type** `pd.DataFrame`

**static load** (*file\_path*)

Loads component at file path

**Parameters** `file_path` (*str*) – Location to load file

**Returns** `ComponentBase` object

**needs\_fitting** (*self*)

Returns boolean determining if component needs fitting before calling `predict`, `predict_proba`, `transform`, or `feature_importances`. This can be overridden to `False` for components that do not need to be fit or whose fit methods do nothing.

**property parameters** (*self*)

Returns the parameters which were used to initialize the component

**save** (*self*, *file\_path*, *pickle\_protocol=cloudpickle.DEFAULT\_PROTOCOL*)

Saves component at file path

**Parameters**

- `file_path` (*str*) – Location to save file
- `pickle_protocol` (*int*) – The pickle data stream format.

**Returns** `None`

**transform** (*self*, *X*, *y=None*)

Transforms input data by imputing missing values.

**Parameters**

- `X` (*pd.DataFrame* or *np.ndarray*) – The input training data of shape `[n_samples, n_features]` to transform.
- `y` (*pd.Series*, *optional*) – The target training data of length `[n_samples]`. Ignored.

**Returns** Transformed `X`

**Return type** `pd.DataFrame`

```
class evalml.pipelines.components.transformers.PolynomialDetrender (degree=1,
                                                                    ran-
                                                                    dom_seed=0,
                                                                    **kwargs)
```

Removes trends from time series by fitting a polynomial to the data.

**Parameters**

- `degree` (*int*) – Degree for the polynomial. If 1, linear model is fit to the data. If 2, quadratic model is fit, etc. Defaults to 1.
- `random_seed` (*int*) – Seed for the random number generator. Defaults to 0.

**Attributes**

<b>hyper-parameter_ranges</b>	{ "degree": Integer(1, 3)}
<b>model_family</b>	ModelFamily.NONE
<b>modifies_features</b>	False
<b>modifies_target</b>	True
<b>name</b>	Polynomial Detrender

## Methods

<i>clone</i>	Constructs a new component with the same parameters and random state.
<i>default_parameters</i>	Returns the default parameters for this component.
<i>describe</i>	Describe a component and its parameters
<i>fit</i>	Fits the PolynomialDetrender.
<i>fit_transform</i>	Removes fitted trend from target variable.
<i>inverse_transform</i>	Adds back fitted trend to target variable.
<i>load</i>	Loads component at file path
<i>needs_fitting</i>	Returns boolean determining if component needs fitting before
<i>parameters</i>	Returns the parameters which were used to initialize the component
<i>save</i>	Saves component at file path
<i>transform</i>	Removes fitted trend from target variable.

### **clone** (*self*)

Constructs a new component with the same parameters and random state.

**Returns** A new instance of this component with identical parameters and random state.

### **default\_parameters** (*cls*)

Returns the default parameters for this component.

Our convention is that `Component.default_parameters == Component().parameters`.

**Returns** default parameters for this component.

**Return type** dict

### **describe** (*self*, *print\_name=False*, *return\_dict=False*)

Describe a component and its parameters

#### Parameters

- **print\_name** (*bool*, *optional*) – whether to print name of component
- **return\_dict** (*bool*, *optional*) – whether to return description as dictionary in the format {"name": name, "parameters": parameters}

**Returns** prints and returns dictionary

**Return type** None or dict

### **fit** (*self*, *X*, *y=None*)

Fits the PolynomialDetrender.

**Parameters**

- **X** (*pd.DataFrame*, *optional*) – Ignored.
- **y** (*pd.Series*) – Target variable to detrend.

**Returns** *self*

**fit\_transform** (*self*, *X*, *y=None*)

Removes fitted trend from target variable.

**Parameters**

- **X** (*pd.DataFrame*, *optional*) – Ignored.
- **y** (*pd.Series*) – Target variable to detrend.

**Returns**

**The first element are the input features returned without modification.** The second element is the target variable *y* with the fitted trend removed.

**Return type** tuple of *pd.DataFrame*, *pd.Series*

**inverse\_transform** (*self*, *y*)

Adds back fitted trend to target variable.

**Parameters**

- **X** (*pd.DataFrame*, *optional*) – Ignored.
- **y** (*pd.Series*) – Target variable.

**Returns**

**The first element are the input features returned without modification.** The second element is the target variable *y* with the trend added back.

**Return type** tuple of *pd.DataFrame*, *pd.Series*

**static load** (*file\_path*)

Loads component at file path

**Parameters** **file\_path** (*str*) – Location to load file

**Returns** *ComponentBase* object

**needs\_fitting** (*self*)

Returns boolean determining if component needs fitting before calling *predict*, *predict\_proba*, *transform*, or *feature\_importances*. This can be overridden to *False* for components that do not need to be fit or whose fit methods do nothing.

**property parameters** (*self*)

Returns the parameters which were used to initialize the component

**save** (*self*, *file\_path*, *pickle\_protocol=cloudpickle.DEFAULT\_PROTOCOL*)

Saves component at file path

**Parameters**

- **file\_path** (*str*) – Location to save file
- **pickle\_protocol** (*int*) – The pickle data stream format.

**Returns** *None*

**transform** (*self*, *X*, *y=None*)

Removes fitted trend from target variable.

**Parameters**

- **X** (*pd.DataFrame, optional*) – Ignored.
- **y** (*pd.Series*) – Target variable to detrend.

**Returns**

The input features are returned without modification. The target variable `y` is detrended

**Return type** tuple of `pd.DataFrame`, `pd.Series`

```
class evalml.pipelines.components.transformers.RFClassifierSelectFromModel (number_features=None,  
                                                                    n_estimators=10,  
                                                                    max_depth=None,  
                                                                    per-  
                                                                    cent_features=0.5,  
                                                                    threshold=-  
  
                                                                    np.inf,  
                                                                    n_jobs=-  
  
                                                                    1,  
                                                                    ran-  
                                                                    dom_seed=0,  
                                                                    **kwargs)
```

Selects top features based on importance weights using a Random Forest classifier.

**Parameters**

- **number\_features** (*int*) – The maximum number of features to select. If both `percent_features` and `number_features` are specified, take the greater number of features. Defaults to 0.5. Defaults to None.
- **n\_estimators** (*float*) – The number of trees in the forest. Defaults to 100.
- **max\_depth** (*int*) – Maximum tree depth for base learners. Defaults to 6.
- **percent\_features** (*float*) – Percentage of features to use. If both `percent_features` and `number_features` are specified, take the greater number of features. Defaults to 0.5.
- **threshold** (*string or float*) – The threshold value to use for feature selection. Features whose importance is greater or equal are kept while the others are discarded. If “median”, then the threshold value is the median of the feature importances. A scaling factor (e.g., “1.25\*mean”) may also be used. Defaults to `-np.inf`.
- **n\_jobs** (*int or None*) – Number of jobs to run in parallel. `-1` uses all processes. Defaults to `-1`.
- **random\_seed** (*int*) – Seed for the random number generator. Defaults to 0.

**Attributes**

<b>hyper-parameter_ranges</b>	{ “percent_features”: Real(0.01, 1), “threshold”: [“mean”, -np.inf], }
<b>model_family</b>	ModelFamily.NONE
<b>modifies_features</b>	True
<b>modifies_target</b>	False
<b>name</b>	RF Classifier Select From Model

## Methods

<i>clone</i>	Constructs a new component with the same parameters and random state.
<i>default_parameters</i>	Returns the default parameters for this component.
<i>describe</i>	Describe a component and its parameters
<i>fit</i>	Fits component to data
<i>fit_transform</i>	Fits on X and transforms X
<i>get_names</i>	Get names of selected features.
<i>load</i>	Loads component at file path
<i>needs_fitting</i>	Returns boolean determining if component needs fitting before
<i>parameters</i>	Returns the parameters which were used to initialize the component
<i>save</i>	Saves component at file path
<i>transform</i>	Transforms input data by selecting features. If the component_obj does not have a transform method, will raise an MethodPropertyNotFoundError exception.

### **clone** (*self*)

Constructs a new component with the same parameters and random state.

**Returns** A new instance of this component with identical parameters and random state.

### **default\_parameters** (*cls*)

Returns the default parameters for this component.

Our convention is that `Component.default_parameters == Component().parameters`.

**Returns** default parameters for this component.

**Return type** dict

### **describe** (*self*, *print\_name=False*, *return\_dict=False*)

Describe a component and its parameters

#### Parameters

- **print\_name** (*bool*, *optional*) – whether to print name of component
- **return\_dict** (*bool*, *optional*) – whether to return description as dictionary in the format {“name”: name, “parameters”: parameters}

**Returns** prints and returns dictionary

**Return type** None or dict

**fit** (*self*, *X*, *y=None*)

Fits component to data

**Parameters**

- **X** (*list*, *pd.DataFrame* or *np.ndarray*) – The input training data of shape [n\_samples, n\_features]
- **y** (*list*, *pd.Series*, *np.ndarray*, *optional*) – The target training data of length [n\_samples]

**Returns** *self*

**fit\_transform** (*self*, *X*, *y=None*)

Fits on X and transforms X

**Parameters**

- **X** (*pd.DataFrame*) – Data to fit and transform
- **y** (*pd.Series*) – Target data

**Returns** Transformed X

**Return type** *pd.DataFrame*

**get\_names** (*self*)

Get names of selected features.

**Returns** List of the names of features selected

**Return type** *list[str]*

**static load** (*file\_path*)

Loads component at file path

**Parameters** **file\_path** (*str*) – Location to load file

**Returns** *ComponentBase* object

**needs\_fitting** (*self*)

Returns boolean determining if component needs fitting before calling predict, predict\_proba, transform, or feature\_importances. This can be overridden to False for components that do not need to be fit or whose fit methods do nothing.

**property parameters** (*self*)

Returns the parameters which were used to initialize the component

**save** (*self*, *file\_path*, *pickle\_protocol=cloudpickle.DEFAULT\_PROTOCOL*)

Saves component at file path

**Parameters**

- **file\_path** (*str*) – Location to save file
- **pickle\_protocol** (*int*) – The pickle data stream format.

**Returns** *None*

**transform** (*self*, *X*, *y=None*)

Transforms input data by selecting features. If the component\_obj does not have a transform method, will raise an *MethodPropertyNotFoundError* exception.

**Parameters**

- **X** (*pd.DataFrame*) – Data to transform.

- **y** (*pd.Series, optional*) – Target data. Ignored.

**Returns** Transformed X

**Return type** `pd.DataFrame`

```
class evalml.pipelines.components.transformers.RFRegressorSelectFromModel (number_features=None,
                                                                    n_estimators=10,
                                                                    max_depth=None,
                                                                    per-
                                                                    cent_features=0.5,
                                                                    threshold=-
                                                                    np.inf,
                                                                    n_jobs=-
                                                                    1,
                                                                    ran-
                                                                    dom_seed=0,
                                                                    **kwargs)
```

Selects top features based on importance weights using a Random Forest regressor.

#### Parameters

- **number\_features** (*int*) – The maximum number of features to select. If both `percent_features` and `number_features` are specified, take the greater number of features. Defaults to 0.5. Defaults to None.
- **n\_estimators** (*float*) – The number of trees in the forest. Defaults to 100.
- **max\_depth** (*int*) – Maximum tree depth for base learners. Defaults to 6.
- **percent\_features** (*float*) – Percentage of features to use. If both `percent_features` and `number_features` are specified, take the greater number of features. Defaults to 0.5.
- **threshold** (*string or float*) – The threshold value to use for feature selection. Features whose importance is greater or equal are kept while the others are discarded. If “median”, then the threshold value is the median of the feature importances. A scaling factor (e.g., “1.25\*mean”) may also be used. Defaults to `-np.inf`.
- **n\_jobs** (*int or None*) – Number of jobs to run in parallel. -1 uses all processes. Defaults to -1.
- **random\_seed** (*int*) – Seed for the random number generator. Defaults to 0.

#### Attributes

<b>hyper-parameter_ranges</b>	{ “percent_features”: Real(0.01, 1), “threshold”: [“mean”, -np.inf], }
<b>model_family</b>	ModelFamily.NONE
<b>modifies_features</b>	True
<b>modifies_target</b>	False
<b>name</b>	RF Regressor Select From Model

#### Methods

<code>clone</code>	Constructs a new component with the same parameters and random state.
<code>default_parameters</code>	Returns the default parameters for this component.
<code>describe</code>	Describe a component and its parameters
<code>fit</code>	Fits component to data
<code>fit_transform</code>	Fits on X and transforms X
<code>get_names</code>	Get names of selected features.
<code>load</code>	Loads component at file path
<code>needs_fitting</code>	Returns boolean determining if component needs fitting before
<code>parameters</code>	Returns the parameters which were used to initialize the component
<code>save</code>	Saves component at file path
<code>transform</code>	Transforms input data by selecting features. If the component_obj does not have a transform method, will raise an <code>MethodPropertyNotFoundError</code> exception.

**clone** (*self*)

Constructs a new component with the same parameters and random state.

**Returns** A new instance of this component with identical parameters and random state.

**default\_parameters** (*cls*)

Returns the default parameters for this component.

Our convention is that `Component.default_parameters == Component().parameters`.

**Returns** default parameters for this component.

**Return type** dict

**describe** (*self*, *print\_name=False*, *return\_dict=False*)

Describe a component and its parameters

**Parameters**

- **print\_name** (*bool*, *optional*) – whether to print name of component
- **return\_dict** (*bool*, *optional*) – whether to return description as dictionary in the format {"name": name, "parameters": parameters}

**Returns** prints and returns dictionary

**Return type** None or dict

**fit** (*self*, *X*, *y=None*)

Fits component to data

**Parameters**

- **X** (*list*, *pd.DataFrame* or *np.ndarray*) – The input training data of shape [n\_samples, n\_features]
- **y** (*list*, *pd.Series*, *np.ndarray*, *optional*) – The target training data of length [n\_samples]

**Returns** self

**fit\_transform** (*self*, *X*, *y=None*)

Fits on X and transforms X



**Parameters**

- **x** (*pd.DataFrame*) – Data to fit and transform
- **y** (*pd.Series*) – Target data

**Returns** Transformed X**Return type** *pd.DataFrame***get\_names** (*self*)

Get names of selected features.

**Returns** List of the names of features selected**Return type** *list[str]***static load** (*file\_path*)

Loads component at file path

**Parameters** **file\_path** (*str*) – Location to load file**Returns** *ComponentBase* object**needs\_fitting** (*self*)

Returns boolean determining if component needs fitting before calling predict, predict\_proba, transform, or feature\_importances. This can be overridden to False for components that do not need to be fit or whose fit methods do nothing.

**property parameters** (*self*)

Returns the parameters which were used to initialize the component

**save** (*self, file\_path, pickle\_protocol=cloudpickle.DEFAULT\_PROTOCOL*)

Saves component at file path

**Parameters**

- **file\_path** (*str*) – Location to save file
- **pickle\_protocol** (*int*) – The pickle data stream format.

**Returns** None**transform** (*self, X, y=None*)Transforms input data by selecting features. If the component\_obj does not have a transform method, will raise an *MethodPropertyNotFoundError* exception.**Parameters**

- **x** (*pd.DataFrame*) – Data to transform.
- **y** (*pd.Series, optional*) – Target data. Ignored.

**Returns** Transformed X**Return type** *pd.DataFrame*

**class** evalml.pipelines.components.transformers.**SelectByType** (*column\_types=None, random\_seed=0, \*\*kwargs*)

Selects columns by specified Woodwork logical type or semantic tag in input data.

**Parameters**

- **column\_types** (*string, ww.LogicalType, list(string), list(ww.LogicalType)*) – List of Woodwork types or tags, used to determine which columns to select.

- **random\_seed** (*int*) – Seed for the random number generator. Defaults to 0.

**Attributes**

<b>hyper-parameter_ranges</b>	{}
<b>model_family</b>	ModelFamily.NONE
<b>modifies_features</b>	True
<b>modifies_target</b>	False
<b>name</b>	Select Columns By Type Transformer
<b>needs_fitting</b>	False

**Methods**

<i>clone</i>	Constructs a new component with the same parameters and random state.
<i>default_parameters</i>	Returns the default parameters for this component.
<i>describe</i>	Describe a component and its parameters
<i>fit</i>	Fits the transformer by checking if column names are present in the dataset.
<i>fit_transform</i>	Fits on X and transforms X
<i>load</i>	Loads component at file path
<i>parameters</i>	Returns the parameters which were used to initialize the component
<i>save</i>	Saves component at file path
<i>transform</i>	Transforms data X.

**clone** (*self*)

Constructs a new component with the same parameters and random state.

**Returns** A new instance of this component with identical parameters and random state.

**default\_parameters** (*cls*)

Returns the default parameters for this component.

Our convention is that `Component.default_parameters == Component().parameters`.

**Returns** default parameters for this component.

**Return type** dict

**describe** (*self*, *print\_name=False*, *return\_dict=False*)

Describe a component and its parameters

**Parameters**

- **print\_name** (*bool*, *optional*) – whether to print name of component
- **return\_dict** (*bool*, *optional*) – whether to return description as dictionary in the format {"name": name, "parameters": parameters}

**Returns** prints and returns dictionary

**Return type** None or dict

**fit** (*self*, *X*, *y=None*)

Fits the transformer by checking if column names are present in the dataset.

**Parameters**

- **X** (*pd.DataFrame*) – Data to check.
- **y** (*pd.Series*, *optional*) – Targets.

**Returns** *self*

**fit\_transform** (*self*, *X*, *y=None*)

Fits on X and transforms X

**Parameters**

- **X** (*pd.DataFrame*) – Data to fit and transform
- **y** (*pd.Series*) – Target data

**Returns** Transformed X

**Return type** *pd.DataFrame*

**static load** (*file\_path*)

Loads component at file path

**Parameters** **file\_path** (*str*) – Location to load file

**Returns** *ComponentBase* object

**property parameters** (*self*)

Returns the parameters which were used to initialize the component

**save** (*self*, *file\_path*, *pickle\_protocol=cloudpickle.DEFAULT\_PROTOCOL*)

Saves component at file path

**Parameters**

- **file\_path** (*str*) – Location to save file
- **pickle\_protocol** (*int*) – The pickle data stream format.

**Returns** *None*

**transform** (*self*, *X*, *y=None*)

Transforms data X.

**Parameters**

- **X** (*pd.DataFrame*) – Data to transform.
- **y** (*pd.Series*, *optional*) – Target data.

**Returns** Transformed X

**Return type** *pd.DataFrame*

**class** evalml.pipelines.components.transformers.**SelectColumns** (*columns=None*,  
*random\_seed=0*,  
*\*\*kwargs*)

Selects specified columns in input data.

**Parameters**

- **columns** (*list(string)*) – List of column names, used to determine which columns to select.
- **random\_seed** (*int*) – Seed for the random number generator. Defaults to 0.

### Attributes

<b>hyper-parameter_ranges</b>	{}
<b>model_family</b>	ModelFamily.NONE
<b>modifies_features</b>	True
<b>modifies_target</b>	False
<b>name</b>	Select Columns Transformer
<b>needs_fitting</b>	False

### Methods

<i>clone</i>	Constructs a new component with the same parameters and random state.
<i>default_parameters</i>	Returns the default parameters for this component.
<i>describe</i>	Describe a component and its parameters
<i>fit</i>	Fits the transformer by checking if column names are present in the dataset.
<i>fit_transform</i>	Fits on X and transforms X
<i>load</i>	Loads component at file path
<i>parameters</i>	Returns the parameters which were used to initialize the component
<i>save</i>	Saves component at file path
<i>transform</i>	Transforms data X by selecting columns.

#### **clone** (*self*)

Constructs a new component with the same parameters and random state.

**Returns** A new instance of this component with identical parameters and random state.

#### **default\_parameters** (*cls*)

Returns the default parameters for this component.

Our convention is that `Component.default_parameters == Component().parameters`.

**Returns** default parameters for this component.

**Return type** dict

#### **describe** (*self*, *print\_name=False*, *return\_dict=False*)

Describe a component and its parameters

##### Parameters

- **print\_name** (*bool*, *optional*) – whether to print name of component
- **return\_dict** (*bool*, *optional*) – whether to return description as dictionary in the format {"name": name, "parameters": parameters}

**Returns** prints and returns dictionary

**Return type** None or dict

#### **fit** (*self*, *X*, *y=None*)

Fits the transformer by checking if column names are present in the dataset.

**Parameters**

- **x** (*pd.DataFrame*) – Data to check.
- **y** (*pd.Series*, *optional*) – Targets.

**Returns** *self***fit\_transform** (*self*, *X*, *y=None*)

Fits on X and transforms X

**Parameters**

- **x** (*pd.DataFrame*) – Data to fit and transform
- **y** (*pd.Series*) – Target data

**Returns** Transformed X**Return type** *pd.DataFrame***static load** (*file\_path*)

Loads component at file path

**Parameters** **file\_path** (*str*) – Location to load file**Returns** ComponentBase object**property parameters** (*self*)

Returns the parameters which were used to initialize the component

**save** (*self*, *file\_path*, *pickle\_protocol=cloudpickle.DEFAULT\_PROTOCOL*)

Saves component at file path

**Parameters**

- **file\_path** (*str*) – Location to save file
- **pickle\_protocol** (*int*) – The pickle data stream format.

**Returns** *None***transform** (*self*, *X*, *y=None*)

Transforms data X by selecting columns.

**Parameters**

- **x** (*pd.DataFrame*) – Data to transform.
- **y** (*pd.Series*, *optional*) – Targets.

**Returns** Transformed X.**Return type** *pd.DataFrame*

```
class evalml.pipelines.components.transformers.SimpleImputer (impute_strategy='most_frequent',  

                                                         fill_value=None,  

                                                         random_seed=0,  

                                                         **kwargs)
```

Imputes missing data according to a specified imputation strategy.

**Parameters**

- **impute\_strategy** (*string*) – Impute strategy to use. Valid values include “mean”, “median”, “most\_frequent”, “constant” for numerical data, and “most\_frequent”, “constant” for object data types.

- **fill\_value** (*string*) – When `impute_strategy == “constant”`, `fill_value` is used to replace missing data. Defaults to 0 when imputing numerical data and “missing\_value” for strings or object data types.
- **random\_seed** (*int*) – Seed for the random number generator. Defaults to 0.

#### Attributes

<b>hyper-parameter_ranges</b>	{ “impute_strategy”: [“mean”, “median”, “most_frequent”]}
<b>model_family</b>	ModelFamily.NONE
<b>modifies_features</b>	True
<b>modifies_target</b>	False
<b>name</b>	Simple Imputer

#### Methods

<i>clone</i>	Constructs a new component with the same parameters and random state.
<i>default_parameters</i>	Returns the default parameters for this component.
<i>describe</i>	Describe a component and its parameters
<i>fit</i>	Fits imputer to data. ‘None’ values are converted to np.nan before imputation and are
<i>fit_transform</i>	Fits on X and transforms X
<i>load</i>	Loads component at file path
<i>needs_fitting</i>	Returns boolean determining if component needs fitting before
<i>parameters</i>	Returns the parameters which were used to initialize the component
<i>save</i>	Saves component at file path
<i>transform</i>	Transforms input by imputing missing values. ‘None’ and np.nan values are treated as the same.

#### **clone** (*self*)

Constructs a new component with the same parameters and random state.

**Returns** A new instance of this component with identical parameters and random state.

#### **default\_parameters** (*cls*)

Returns the default parameters for this component.

Our convention is that `Component.default_parameters == Component().parameters`.

**Returns** default parameters for this component.

**Return type** dict

#### **describe** (*self*, *print\_name=False*, *return\_dict=False*)

Describe a component and its parameters

#### Parameters

- **print\_name** (*bool*, *optional*) – whether to print name of component

- **return\_dict** (*bool, optional*) – whether to return description as dictionary in the format {"name": name, "parameters": parameters}

**Returns** prints and returns dictionary

**Return type** None or dict

**fit** (*self, X, y=None*)

**Fits imputer to data.** ‘None’ values are converted to np.nan before imputation and are treated as the same.

**Parameters**

- **X** (*pd.DataFrame or np.ndarray*) – the input training data of shape [n\_samples, n\_features]
- **y** (*pd.Series, optional*) – the target training data of length [n\_samples]

**Returns** self

**fit\_transform** (*self, X, y=None*)

Fits on X and transforms X

**Parameters**

- **X** (*pd.DataFrame*) – Data to fit and transform
- **y** (*pd.Series, optional*) – Target data.

**Returns** Transformed X

**Return type** pd.DataFrame

**static load** (*file\_path*)

Loads component at file path

**Parameters** **file\_path** (*str*) – Location to load file

**Returns** ComponentBase object

**needs\_fitting** (*self*)

Returns boolean determining if component needs fitting before calling predict, predict\_proba, transform, or feature\_importances. This can be overridden to False for components that do not need to be fit or whose fit methods do nothing.

**property parameters** (*self*)

Returns the parameters which were used to initialize the component

**save** (*self, file\_path, pickle\_protocol=cloudpickle.DEFAULT\_PROTOCOL*)

Saves component at file path

**Parameters**

- **file\_path** (*str*) – Location to save file
- **pickle\_protocol** (*int*) – The pickle data stream format.

**Returns** None

**transform** (*self, X, y=None*)

Transforms input by imputing missing values. ‘None’ and np.nan values are treated as the same.

**Parameters**

- **X** (*pd.DataFrame*) – Data to transform

- **y** (*pd.Series, optional*) – Ignored.

**Returns** Transformed X

**Return type** `pd.DataFrame`

```
class evalml.pipelines.components.transformers.SMOTENCOverSampler(sampling_ratio=0.25,  
                                                                k_neighbors_default=5,  
                                                                n_jobs=-  
                                                                1,      ran-  
                                                                dom_seed=0,  
                                                                **kwargs)
```

SMOTENC Oversampler component. Uses SMOTENC to generate synthetic samples. Works on a mix of numerical and categorical columns. Input data must be Woodwork type, and this component is only run during training and not during predict.

#### Parameters

- **sampling\_ratio** (*float*) – This is the goal ratio of the minority to majority class, with range (0, 1]. A value of 0.25 means we want a 1:4 ratio of the minority to majority class after oversampling. We will create the a sampling dictionary using this ratio, with the keys corresponding to the class and the values responding to the number of samples. Defaults to 0.25.
- **k\_neighbors\_default** (*int*) – The number of nearest neighbors used to construct synthetic samples. This is the default value used, but the actual `k_neighbors` value might be smaller if there are less samples. Defaults to 5.
- **n\_jobs** (*int*) – The number of CPU cores to use. Defaults to -1.
- **random\_seed** (*int*) – The seed to use for random sampling. Defaults to 0.

#### Attributes

<b>hyper-parameter_ranges</b>	{}
<b>model_family</b>	ModelFamily.NONE
<b>modifies_features</b>	True
<b>modifies_target</b>	True
<b>name</b>	SMOTENC Oversampler

#### Methods

<code>clone</code>	Constructs a new component with the same parameters and random state.
<code>default_parameters</code>	Returns the default parameters for this component.
<code>describe</code>	Describe a component and its parameters
<code>fit</code>	Fits the sampler to the data.
<code>fit_transform</code>	Fits on X and transforms X
<code>load</code>	Loads component at file path
<code>needs_fitting</code>	Returns boolean determining if component needs fitting before

continues on next page



Table 522 – continued from previous page

<i>parameters</i>	Returns the parameters which were used to initialize the component
<i>save</i>	Saves component at file path
<i>transform</i>	Transforms the input data by sampling the data.

**clone** (*self*)

Constructs a new component with the same parameters and random state.

**Returns** A new instance of this component with identical parameters and random state.

**default\_parameters** (*cls*)

Returns the default parameters for this component.

Our convention is that `Component.default_parameters == Component().parameters`.

**Returns** default parameters for this component.

**Return type** dict

**describe** (*self*, *print\_name=False*, *return\_dict=False*)

Describe a component and its parameters

**Parameters**

- **print\_name** (*bool*, *optional*) – whether to print name of component
- **return\_dict** (*bool*, *optional*) – whether to return description as dictionary in the format {"name": name, "parameters": parameters}

**Returns** prints and returns dictionary

**Return type** None or dict

**fit** (*self*, *X*, *y*)

Fits the sampler to the data.

**Parameters**

- **X** (*pd.DataFrame*) – Input features.
- **y** (*pd.Series*) – Target.

**Returns** self

**fit\_transform** (*self*, *X*, *y*)

Fits on X and transforms X

**Parameters**

- **X** (*pd.DataFrame*) – Data to fit and transform
- **y** (*pd.Series*) – Target data

**Returns** Transformed X

**Return type** pd.DataFrame

**static load** (*file\_path*)

Loads component at file path

**Parameters** **file\_path** (*str*) – Location to load file

**Returns** ComponentBase object

**needs\_fitting** (*self*)

Returns boolean determining if component needs fitting before calling predict, predict\_proba, transform, or feature\_importances. This can be overridden to False for components that do not need to be fit or whose fit methods do nothing.

**property parameters** (*self*)

Returns the parameters which were used to initialize the component

**save** (*self*, *file\_path*, *pickle\_protocol*=*cloudpickle.DEFAULT\_PROTOCOL*)

Saves component at file path

**Parameters**

- **file\_path** (*str*) – Location to save file
- **pickle\_protocol** (*int*) – The pickle data stream format.

**Returns** None

**transform** (*self*, *X*, *y=None*)

Transforms the input data by sampling the data.

**Parameters**

- **X** (*pd.DataFrame*) – Training features.
- **y** (*pd.Series*) – Target.

**Returns** Transformed features and target.

**Return type** *pd.DataFrame*, *pd.Series*

```
class evalml.pipelines.components.transformers.SMOTENOversampler (sampling_ratio=0.25,  
                                                             k_neighbors_default=5,  
                                                             n_jobs=-  
                                                             1, random  
                                                             seed=0,  
                                                             **kwargs)
```

SMOTEN Oversampler component. Uses SMOTEN to generate synthetic samples. Works for purely categorical datasets. This component is only run during training and not during predict.

**Parameters**

- **sampling\_ratio** (*float*) – This is the goal ratio of the minority to majority class, with range (0, 1]. A value of 0.25 means we want a 1:4 ratio of the minority to majority class after oversampling. We will create the a sampling dictionary using this ratio, with the keys corresponding to the class and the values responding to the number of samples. Defaults to 0.25.
- **k\_neighbors\_default** (*int*) – The number of nearest neighbors used to construct synthetic samples. This is the default value used, but the actual *k\_neighbors* value might be smaller if there are less samples. Defaults to 5.
- **n\_jobs** (*int*) – The number of CPU cores to use. Defaults to -1.
- **random\_seed** (*int*) – The seed to use for random sampling. Defaults to 0.

**Attributes**

<b>hyper-parameter_ranges</b>	{}
<b>model_family</b>	ModelFamily.NONE
<b>modifies_features</b>	True
<b>modifies_target</b>	True
<b>name</b>	SMOTEN Oversampler

## Methods

<i>clone</i>	Constructs a new component with the same parameters and random state.
<i>default_parameters</i>	Returns the default parameters for this component.
<i>describe</i>	Describe a component and its parameters
<i>fit</i>	Fits the sampler to the data.
<i>fit_transform</i>	Fits on X and transforms X
<i>load</i>	Loads component at file path
<i>needs_fitting</i>	Returns boolean determining if component needs fitting before
<i>parameters</i>	Returns the parameters which were used to initialize the component
<i>save</i>	Saves component at file path
<i>transform</i>	Transforms the input data by sampling the data.

### **clone** (*self*)

Constructs a new component with the same parameters and random state.

**Returns** A new instance of this component with identical parameters and random state.

### **default\_parameters** (*cls*)

Returns the default parameters for this component.

Our convention is that `Component.default_parameters == Component().parameters`.

**Returns** default parameters for this component.

**Return type** dict

### **describe** (*self*, *print\_name=False*, *return\_dict=False*)

Describe a component and its parameters

#### Parameters

- **print\_name** (*bool*, *optional*) – whether to print name of component
- **return\_dict** (*bool*, *optional*) – whether to return description as dictionary in the format {"name": name, "parameters": parameters}

**Returns** prints and returns dictionary

**Return type** None or dict

### **fit** (*self*, *X*, *y*)

Fits the sampler to the data.

#### Parameters

- **X** (*pd.DataFrame*) – Input features.
- **y** (*pd.Series*) – Target.

**Returns** *self*

**fit\_transform** (*self*, *X*, *y*)  
Fits on X and transforms X

**Parameters**

- **X** (*pd.DataFrame*) – Data to fit and transform
- **y** (*pd.Series*) – Target data

**Returns** Transformed X

**Return type** *pd.DataFrame*

**static load** (*file\_path*)  
Loads component at file path

**Parameters** **file\_path** (*str*) – Location to load file

**Returns** *ComponentBase* object

**needs\_fitting** (*self*)  
Returns boolean determining if component needs fitting before calling predict, predict\_proba, transform, or feature\_importances. This can be overridden to False for components that do not need to be fit or whose fit methods do nothing.

**property parameters** (*self*)  
Returns the parameters which were used to initialize the component

**save** (*self*, *file\_path*, *pickle\_protocol=cloudpickle.DEFAULT\_PROTOCOL*)  
Saves component at file path

**Parameters**

- **file\_path** (*str*) – Location to save file
- **pickle\_protocol** (*int*) – The pickle data stream format.

**Returns** *None*

**transform** (*self*, *X*, *y=None*)  
Transforms the input data by sampling the data.

**Parameters**

- **X** (*pd.DataFrame*) – Training features.
- **y** (*pd.Series*) – Target.

**Returns** Transformed features and target.

**Return type** *pd.DataFrame*, *pd.Series*

**class** evalml.pipelines.components.transformers.**SMOTEOversampler** (*sampling\_ratio=0.25*,  
*k\_neighbors\_default=5*,  
*n\_jobs=-1*,  
*random\_seed=0*,  
*\*\*kwargs*)

SMOTE Oversampler component. Works on numerical datasets only. This component is only run during training and not during predict.

### Parameters

- **sampling\_ratio** (*float*) – This is the goal ratio of the minority to majority class, with range (0, 1]. A value of 0.25 means we want a 1:4 ratio of the minority to majority class after oversampling. We will create the a sampling dictionary using this ratio, with the keys corresponding to the class and the values responding to the number of samples. Defaults to 0.25.
- **k\_neighbors\_default** (*int*) – The number of nearest neighbors used to construct synthetic samples. This is the default value used, but the actual k\_neighbors value might be smaller if there are less samples. Defaults to 5.
- **n\_jobs** (*int*) – The number of CPU cores to use. Defaults to -1.
- **random\_seed** (*int*) – The seed to use for random sampling. Defaults to 0.

### Attributes

<b>hyper-parameter_ranges</b>	{}
<b>model_family</b>	ModelFamily.NONE
<b>modifies_features</b>	True
<b>modifies_target</b>	True
<b>name</b>	SMOTE Oversampler

### Methods

<i>clone</i>	Constructs a new component with the same parameters and random state.
<i>default_parameters</i>	Returns the default parameters for this component.
<i>describe</i>	Describe a component and its parameters
<i>fit</i>	Fits the sampler to the data.
<i>fit_transform</i>	Fits on X and transforms X
<i>load</i>	Loads component at file path
<i>needs_fitting</i>	Returns boolean determining if component needs fitting before
<i>parameters</i>	Returns the parameters which were used to initialize the component
<i>save</i>	Saves component at file path
<i>transform</i>	Transforms the input data by sampling the data.

#### **clone** (*self*)

Constructs a new component with the same parameters and random state.

**Returns** A new instance of this component with identical parameters and random state.

#### **default\_parameters** (*cls*)

Returns the default parameters for this component.

Our convention is that `Component.default_parameters == Component().parameters`.

**Returns** default parameters for this component.

**Return type** dict

**describe** (*self*, *print\_name=False*, *return\_dict=False*)

Describe a component and its parameters

**Parameters**

- **print\_name** (*bool*, *optional*) – whether to print name of component
- **return\_dict** (*bool*, *optional*) – whether to return description as dictionary in the format {"name": name, "parameters": parameters}

**Returns** prints and returns dictionary

**Return type** None or dict

**fit** (*self*, *X*, *y*)

Fits the sampler to the data.

**Parameters**

- **X** (*pd.DataFrame*) – Input features.
- **y** (*pd.Series*) – Target.

**Returns** self

**fit\_transform** (*self*, *X*, *y*)

Fits on X and transforms X

**Parameters**

- **X** (*pd.DataFrame*) – Data to fit and transform
- **y** (*pd.Series*) – Target data

**Returns** Transformed X

**Return type** pd.DataFrame

**static load** (*file\_path*)

Loads component at file path

**Parameters** **file\_path** (*str*) – Location to load file

**Returns** ComponentBase object

**needs\_fitting** (*self*)

Returns boolean determining if component needs fitting before calling predict, predict\_proba, transform, or feature\_importances. This can be overridden to False for components that do not need to be fit or whose fit methods do nothing.

**property parameters** (*self*)

Returns the parameters which were used to initialize the component

**save** (*self*, *file\_path*, *pickle\_protocol=cloudpickle.DEFAULT\_PROTOCOL*)

Saves component at file path

**Parameters**

- **file\_path** (*str*) – Location to save file
- **pickle\_protocol** (*int*) – The pickle data stream format.

**Returns** None

**transform** (*self*, *X*, *y=None*)

Transforms the input data by sampling the data.

**Parameters**

- **x** (*pd.DataFrame*) – Training features.
- **y** (*pd.Series*) – Target.

**Returns** Transformed features and target.

**Return type** *pd.DataFrame*, *pd.Series*

**class** `evalml.pipelines.components.transformers.StandardScaler` (*random\_seed=0*,  
\*\**kwargs*)

A transformer that standardizes input features by removing the mean and scaling to unit variance.

**Parameters** **random\_seed** (*int*) – Seed for the random number generator. Defaults to 0.

**Attributes**

<b>hyper-parameter_ranges</b>	{}
<b>model_family</b>	ModelFamily.NONE
<b>modifies_features</b>	True
<b>modifies_target</b>	False
<b>name</b>	Standard Scaler

**Methods**

<i>clone</i>	Constructs a new component with the same parameters and random state.
<i>default_parameters</i>	Returns the default parameters for this component.
<i>describe</i>	Describe a component and its parameters
<i>fit</i>	Fits component to data
<i>fit_transform</i>	Fits on X and transforms X
<i>load</i>	Loads component at file path
<i>needs_fitting</i>	Returns boolean determining if component needs fitting before
<i>parameters</i>	Returns the parameters which were used to initialize the component
<i>save</i>	Saves component at file path
<i>transform</i>	Transforms data X.

**clone** (*self*)

Constructs a new component with the same parameters and random state.

**Returns** A new instance of this component with identical parameters and random state.

**default\_parameters** (*cls*)

Returns the default parameters for this component.

Our convention is that `Component.default_parameters == Component().parameters`.

**Returns** default parameters for this component.

**Return type** *dict*

**describe** (*self*, *print\_name=False*, *return\_dict=False*)

Describe a component and its parameters

**Parameters**

- **print\_name** (*bool, optional*) – whether to print name of component
- **return\_dict** (*bool, optional*) – whether to return description as dictionary in the format {"name": name, "parameters": parameters}

**Returns** prints and returns dictionary

**Return type** None or dict

**fit** (*self, X, y=None*)

Fits component to data

**Parameters**

- **X** (*list, pd.DataFrame or np.ndarray*) – The input training data of shape [n\_samples, n\_features]
- **y** (*list, pd.Series, np.ndarray, optional*) – The target training data of length [n\_samples]

**Returns** self

**fit\_transform** (*self, X, y=None*)

Fits on X and transforms X

**Parameters**

- **X** (*pd.DataFrame*) – Data to fit and transform
- **y** (*pd.Series*) – Target data

**Returns** Transformed X

**Return type** pd.DataFrame

**static load** (*file\_path*)

Loads component at file path

**Parameters** **file\_path** (*str*) – Location to load file

**Returns** ComponentBase object

**needs\_fitting** (*self*)

Returns boolean determining if component needs fitting before calling predict, predict\_proba, transform, or feature\_importances. This can be overridden to False for components that do not need to be fit or whose fit methods do nothing.

**property parameters** (*self*)

Returns the parameters which were used to initialize the component

**save** (*self, file\_path, pickle\_protocol=cloudpickle.DEFAULT\_PROTOCOL*)

Saves component at file path

**Parameters**

- **file\_path** (*str*) – Location to save file
- **pickle\_protocol** (*int*) – The pickle data stream format.

**Returns** None

**transform** (*self, X, y=None*)

Transforms data X.

**Parameters**



- **x** (*pd.DataFrame*) – Data to transform.
- **y** (*pd.Series*, *optional*) – Target data.

**Returns** Transformed X

**Return type** *pd.DataFrame*

```
class evalml.pipelines.components.transformers.TargetEncoder (cols=None,
                                                             smooth-
                                                             ing=1.0,      han-
                                                             dle_unknown='value',
                                                             han-
                                                             dle_missing='value',
                                                             random_seed=0,
                                                             **kwargs)
```

A transformer that encodes categorical features into target encodings.

#### Parameters

- **cols** (*list*) – Columns to encode. If *None*, all string columns will be encoded, otherwise only the columns provided will be encoded. Defaults to *None*
- **smoothing** (*float*) – The smoothing factor to apply. The larger this value is, the more influence the expected target value has on the resulting target encodings. Must be strictly larger than 0. Defaults to 1.0
- **handle\_unknown** (*string*) – Determines how to handle unknown categories for a feature encountered. Options are ‘value’, ‘error’, and ‘return\_nan’. Defaults to ‘value’, which replaces with the target mean
- **handle\_missing** (*string*) – Determines how to handle missing values encountered during *fit* or *transform*. Options are ‘value’, ‘error’, and ‘return\_nan’. Defaults to ‘value’, which replaces with the target mean
- **random\_seed** (*int*) – Seed for the random number generator. Defaults to 0.

#### Attributes

<b>hyper-parameter_ranges</b>	{}
<b>model_family</b>	ModelFamily.NONE
<b>modifies_features</b>	True
<b>modifies_target</b>	False
<b>name</b>	Target Encoder

#### Methods

<i>clone</i>	Constructs a new component with the same parameters and random state.
<i>default_parameters</i>	Returns the default parameters for this component.
<i>describe</i>	Describe a component and its parameters
<i>fit</i>	Fits component to data
<i>fit_transform</i>	Fits on X and transforms X

continues on next page

Table 526 – continued from previous page

<code>get_feature_names</code>	Return feature names for the input features after fitting.
<code>load</code>	Loads component at file path
<code>needs_fitting</code>	Returns boolean determining if component needs fitting before
<code>parameters</code>	Returns the parameters which were used to initialize the component
<code>save</code>	Saves component at file path
<code>transform</code>	Transforms data X.

**clone** (*self*)

Constructs a new component with the same parameters and random state.

**Returns** A new instance of this component with identical parameters and random state.

**default\_parameters** (*cls*)

Returns the default parameters for this component.

Our convention is that `Component.default_parameters == Component().parameters`.

**Returns** default parameters for this component.

**Return type** dict

**describe** (*self*, *print\_name=False*, *return\_dict=False*)

Describe a component and its parameters

**Parameters**

- **print\_name** (*bool*, *optional*) – whether to print name of component
- **return\_dict** (*bool*, *optional*) – whether to return description as dictionary in the format {"name": name, "parameters": parameters}

**Returns** prints and returns dictionary

**Return type** None or dict

**fit** (*self*, *X*, *y*)

Fits component to data

**Parameters**

- **X** (*list*, *pd.DataFrame* or *np.ndarray*) – The input training data of shape [n\_samples, n\_features]
- **y** (*list*, *pd.Series*, *np.ndarray*, *optional*) – The target training data of length [n\_samples]

**Returns** self

**fit\_transform** (*self*, *X*, *y*)

Fits on X and transforms X

**Parameters**

- **X** (*pd.DataFrame*) – Data to fit and transform
- **y** (*pd.Series*) – Target data

**Returns** Transformed X

**Return type** pd.DataFrame

**get\_feature\_names** (*self*)

Return feature names for the input features after fitting.

**Returns** The feature names after encoding

**Return type** np.array

**static load** (*file\_path*)

Loads component at file path

**Parameters** **file\_path** (*str*) – Location to load file

**Returns** ComponentBase object

**needs\_fitting** (*self*)

Returns boolean determining if component needs fitting before calling predict, predict\_proba, transform, or feature\_importances. This can be overridden to False for components that do not need to be fit or whose fit methods do nothing.

**property parameters** (*self*)

Returns the parameters which were used to initialize the component

**save** (*self*, *file\_path*, *pickle\_protocol=cloudpickle.DEFAULT\_PROTOCOL*)

Saves component at file path

**Parameters**

- **file\_path** (*str*) – Location to save file
- **pickle\_protocol** (*int*) – The pickle data stream format.

**Returns** None

**transform** (*self*, *X*, *y=None*)

Transforms data X.

**Parameters**

- **X** (*pd.DataFrame*) – Data to transform.
- **y** (*pd.Series*, *optional*) – Target data.

**Returns** Transformed X

**Return type** pd.DataFrame

```
class evalml.pipelines.components.transformers.TargetImputer (impute_strategy='most_frequent',  
                                                             fill_value=None,  
                                                             random_seed=0,  
                                                             **kwargs)
```

Imputes missing target data according to a specified imputation strategy.

**Parameters**

- **impute\_strategy** (*string*) – Impute strategy to use. Valid values include “mean”, “median”, “most\_frequent”, “constant” for numerical data, and “most\_frequent”, “constant” for object data types. Defaults to “most\_frequent”.
- **fill\_value** (*string*) – When impute\_strategy == “constant”, fill\_value is used to replace missing data. Defaults to None which uses 0 when imputing numerical data and “missing\_value” for strings or object data types.
- **random\_seed** (*int*) – Seed for the random number generator. Defaults to 0.

### Attributes

<b>hyper-parameter_ranges</b>	{ "impute_strategy": ["mean", "median", "most_frequent"] }
<b>model_family</b>	ModelFamily.NONE
<b>modifies_features</b>	False
<b>modifies_target</b>	True
<b>name</b>	Target Imputer

### Methods

<i>clone</i>	Constructs a new component with the same parameters and random state.
<i>default_parameters</i>	Returns the default parameters for this component.
<i>describe</i>	Describe a component and its parameters
<i>fit</i>	Fits imputer to target data. 'None' values are converted to np.nan before imputation and are
<i>fit_transform</i>	Fits on and transforms the input target data.
<i>load</i>	Loads component at file path
<i>needs_fitting</i>	Returns boolean determining if component needs fitting before
<i>parameters</i>	Returns the parameters which were used to initialize the component
<i>save</i>	Saves component at file path
<i>transform</i>	Transforms input target data by imputing missing values. 'None' and np.nan values are treated as the same.

#### **clone** (*self*)

Constructs a new component with the same parameters and random state.

**Returns** A new instance of this component with identical parameters and random state.

#### **default\_parameters** (*cls*)

Returns the default parameters for this component.

Our convention is that `Component.default_parameters == Component().parameters`.

**Returns** default parameters for this component.

**Return type** dict

#### **describe** (*self*, *print\_name=False*, *return\_dict=False*)

Describe a component and its parameters

##### Parameters

- **print\_name** (*bool*, *optional*) – whether to print name of component
- **return\_dict** (*bool*, *optional*) – whether to return description as dictionary in the format {"name": name, "parameters": parameters}

**Returns** prints and returns dictionary

**Return type** None or dict

**fit** (*self*, *X*, *y*)

Fits imputer to target data. ‘None’ values are converted to np.nan before imputation and are treated as the same.

**Parameters**

- **X** (*pd.DataFrame* or *np.ndarray*) – The input training data of shape [n\_samples, n\_features]. Ignored.
- **y** (*pd.Series*, *optional*) – The target training data of length [n\_samples].

**Returns** *self*

**fit\_transform** (*self*, *X*, *y*)

Fits on and transforms the input target data.

**Parameters**

- **X** (*pd.DataFrame*) – Features. Ignored.
- **y** (*pd.Series*) – Target data to impute.

**Returns** The original X, transformed y

**Return type** (*pd.DataFrame*, *pd.Series*)

**static load** (*file\_path*)

Loads component at file path

**Parameters** **file\_path** (*str*) – Location to load file

**Returns** ComponentBase object

**needs\_fitting** (*self*)

Returns boolean determining if component needs fitting before calling predict, predict\_proba, transform, or feature\_importances. This can be overridden to False for components that do not need to be fit or whose fit methods do nothing.

**property parameters** (*self*)

Returns the parameters which were used to initialize the component

**save** (*self*, *file\_path*, *pickle\_protocol=cloudpickle.DEFAULT\_PROTOCOL*)

Saves component at file path

**Parameters**

- **file\_path** (*str*) – Location to save file
- **pickle\_protocol** (*int*) – The pickle data stream format.

**Returns** None

**transform** (*self*, *X*, *y*)

Transforms input target data by imputing missing values. ‘None’ and np.nan values are treated as the same.

**Parameters**

- **X** (*pd.DataFrame*) – Features. Ignored.
- **y** (*pd.Series*) – Target data to impute.

**Returns** The original X, transformed y

**Return type** (*pd.DataFrame*, *pd.Series*)

```
class evalml.pipelines.components.transformers.TextFeaturizer (random_seed=0,  
                                                         **kwargs)
```

Transformer that can automatically featurize text columns using featuretools' nlp\_primitives.

Since models cannot handle non-numeric data, any text must be broken down into features that provide useful information about that text. This component splits each text column into several informative features: Diversity Score, Mean Characters per Word, Polarity Score, and LSA (Latent Semantic Analysis). Calling transform on this component will replace any text columns in the given dataset with these numeric columns.

**Parameters** `random_seed` (*int*) – Seed for the random number generator. Defaults to 0.

#### Attributes

<b>hyper-parameter_ranges</b>	{}
<b>model_family</b>	ModelFamily.NONE
<b>modifies_features</b>	True
<b>modifies_target</b>	False
<b>name</b>	Text Featurization Component

#### Methods

<code>clone</code>	Constructs a new component with the same parameters and random state.
<code>default_parameters</code>	Returns the default parameters for this component.
<code>describe</code>	Describe a component and its parameters
<code>fit</code>	Fits component to data
<code>fit_transform</code>	Fits on X and transforms X
<code>load</code>	Loads component at file path
<code>needs_fitting</code>	Returns boolean determining if component needs fitting before
<code>parameters</code>	Returns the parameters which were used to initialize the component
<code>save</code>	Saves component at file path
<code>transform</code>	Transforms data X by creating new features using existing text columns

**clone** (*self*)

Constructs a new component with the same parameters and random state.

**Returns** A new instance of this component with identical parameters and random state.

**default\_parameters** (*cls*)

Returns the default parameters for this component.

Our convention is that `Component.default_parameters == Component().parameters`.

**Returns** default parameters for this component.

**Return type** dict

**describe** (*self*, *print\_name=False*, *return\_dict=False*)

Describe a component and its parameters

**Parameters**

- **print\_name** (*bool*, *optional*) – whether to print name of component
- **return\_dict** (*bool*, *optional*) – whether to return description as dictionary in the format {"name": name, "parameters": parameters}

**Returns** prints and returns dictionary

**Return type** None or dict

**fit** (*self*, *X*, *y=None*)

Fits component to data

**Parameters**

- **X** (*pd.DataFrame* or *np.ndarray*) – The input training data of shape [n\_samples, n\_features]
- **y** (*pd.Series*, *np.ndarray*, *optional*) – The target training data of length [n\_samples]

**Returns** self

**fit\_transform** (*self*, *X*, *y=None*)

Fits on X and transforms X

**Parameters**

- **X** (*pd.DataFrame*) – Data to fit and transform
- **y** (*pd.Series*) – Target data

**Returns** Transformed X

**Return type** pd.DataFrame

**static load** (*file\_path*)

Loads component at file path

**Parameters** **file\_path** (*str*) – Location to load file

**Returns** ComponentBase object

**needs\_fitting** (*self*)

Returns boolean determining if component needs fitting before calling predict, predict\_proba, transform, or feature\_importances. This can be overridden to False for components that do not need to be fit or whose fit methods do nothing.

**property parameters** (*self*)

Returns the parameters which were used to initialize the component

**save** (*self*, *file\_path*, *pickle\_protocol=cloudpickle.DEFAULT\_PROTOCOL*)

Saves component at file path

**Parameters**

- **file\_path** (*str*) – Location to save file
- **pickle\_protocol** (*int*) – The pickle data stream format.

**Returns** None

**transform** (*self*, *X*, *y=None*)

Transforms data X by creating new features using existing text columns

**Parameters**

- **X** (*pd.DataFrame*) – The data to transform.

- **y** (*pd.Series, optional*) – Ignored.

**Returns** Transformed X

**Return type** `pd.DataFrame`

```
class evalml.pipelines.components.transformers.Transformer (parameters=None,  
                                                         compo-  
                                                         nent_obj=None,  
                                                         random_seed=0,  
                                                         **kwargs)
```

A component that may or may not need fitting that transforms data. These components are used before an estimator.

To implement a new Transformer, define your own class which is a subclass of Transformer, including a name and a list of acceptable ranges for any parameters to be tuned during the automl search (hyperparameters). Define an `__init__` method which sets up any necessary state and objects. Make sure your `__init__` only uses standard keyword arguments and calls `super().__init__()` with a parameters dict. You may also override the `fit`, `transform`, `fit_transform` and other methods in this class if appropriate.

To see some examples, check out the definitions of any Transformer component.

#### Parameters

- **parameters** (*dict*) – Dictionary of parameters for the component. Defaults to None.
- **component\_obj** (*obj*) – Third-party objects useful in component implementation. Defaults to None.
- **random\_seed** (*int*) – Seed for the random number generator. Defaults to 0.

#### Attributes

<b>model_family</b>	ModelFamily.NONE
<b>modifies_features</b>	True
<b>modifies_target</b>	False

#### Methods

<code>clone</code>	Constructs a new component with the same parameters and random state.
<code>default_parameters</code>	Returns the default parameters for this component.
<code>describe</code>	Describe a component and its parameters
<code>fit</code>	Fits component to data
<code>fit_transform</code>	Fits on X and transforms X
<code>load</code>	Loads component at file path
<code>name</code>	Returns string name of this component
<code>needs_fitting</code>	Returns boolean determining if component needs fitting before
<code>parameters</code>	Returns the parameters which were used to initialize the component
<code>save</code>	Saves component at file path
<code>transform</code>	Transforms data X.

**clone** (*self*)

Constructs a new component with the same parameters and random state.



**Returns** A new instance of this component with identical parameters and random state.

**default\_parameters** (*cls*)

Returns the default parameters for this component.

Our convention is that `Component.default_parameters == Component().parameters`.

**Returns** default parameters for this component.

**Return type** dict

**describe** (*self*, *print\_name=False*, *return\_dict=False*)

Describe a component and its parameters

**Parameters**

- **print\_name** (*bool*, *optional*) – whether to print name of component
- **return\_dict** (*bool*, *optional*) – whether to return description as dictionary in the format {"name": name, "parameters": parameters}

**Returns** prints and returns dictionary

**Return type** None or dict

**fit** (*self*, *X*, *y=None*)

Fits component to data

**Parameters**

- **X** (*list*, *pd.DataFrame* or *np.ndarray*) – The input training data of shape [n\_samples, n\_features]
- **y** (*list*, *pd.Series*, *np.ndarray*, *optional*) – The target training data of length [n\_samples]

**Returns** self

**fit\_transform** (*self*, *X*, *y=None*)

Fits on X and transforms X

**Parameters**

- **X** (*pd.DataFrame*) – Data to fit and transform
- **y** (*pd.Series*) – Target data

**Returns** Transformed X

**Return type** pd.DataFrame

**static load** (*file\_path*)

Loads component at file path

**Parameters** **file\_path** (*str*) – Location to load file

**Returns** ComponentBase object

**property name** (*cls*)

Returns string name of this component

**needs\_fitting** (*self*)

Returns boolean determining if component needs fitting before calling predict, predict\_proba, transform, or feature\_importances. This can be overridden to False for components that do not need to be fit or whose fit methods do nothing.

**property parameters** (*self*)

Returns the parameters which were used to initialize the component

**save** (*self*, *file\_path*, *pickle\_protocol*=*cloudpickle.DEFAULT\_PROTOCOL*)

Saves component at file path

**Parameters**

- **file\_path** (*str*) – Location to save file
- **pickle\_protocol** (*int*) – The pickle data stream format.

**Returns** None

**transform** (*self*, *X*, *y*=None)

Transforms data X.

**Parameters**

- **X** (*pd.DataFrame*) – Data to transform.
- **y** (*pd.Series*, *optional*) – Target data.

**Returns** Transformed X

**Return type** *pd.DataFrame*

```
class evalml.pipelines.components.transformers.Undersampler (sampling_ratio=0.25,  
                                                             sam-  
                                                             pling_ratio_dict=None,  
                                                             min_samples=100,  
                                                             min_percentage=0.1,  
                                                             random_seed=0,  
                                                             **kwargs)
```

Initializes an undersampling transformer to downsample the majority classes in the dataset.

This component is only run during training and not during predict.

**Parameters**

- **sampling\_ratio** (*float*) – The smallest minority:majority ratio that is accepted as ‘balanced’. For instance, a 1:4 ratio would be represented as 0.25, while a 1:1 ratio is 1.0. Must be between 0 and 1, inclusive. Defaults to 0.25.
- **sampling\_ratio\_dict** (*dict*) – A dictionary specifying the desired balanced ratio for each target value. For instance, in a binary case where class 1 is the minority, we could specify: *sampling\_ratio\_dict*={0: 0.5, 1: 1}, which means we would undersample class 0 to have twice the number of samples as class 1 (minority:majority ratio = 0.5), and don’t sample class 1. Overrides *sampling\_ratio* if provided. Defaults to None.
- **min\_samples** (*int*) – The minimum number of samples that we must have for any class, pre or post sampling. If a class must be downsampled, it will not be downsampled past this value. To determine severe imbalance, the minority class must occur less often than this and must have a class ratio below *min\_percentage*. Must be greater than 0. Defaults to 100.
- **min\_percentage** (*float*) – The minimum percentage of the minimum class to total dataset that we tolerate, as long as it is above *min\_samples*. If *min\_percentage* and *min\_samples* are not met, treat this as severely imbalanced, and we will not resample the data. Must be between 0 and 0.5, inclusive. Defaults to 0.1.
- **random\_seed** (*int*) – The seed to use for random sampling. Defaults to 0.

**Attributes**

<b>hyper-parameter_ranges</b>	{}
<b>model_family</b>	ModelFamily.NONE
<b>modifies_features</b>	True
<b>modifies_target</b>	True
<b>name</b>	Undersampler

## Methods

<i>clone</i>	Constructs a new component with the same parameters and random state.
<i>default_parameters</i>	Returns the default parameters for this component.
<i>describe</i>	Describe a component and its parameters
<i>fit</i>	Fits the sampler to the data.
<i>fit_transform</i>	Fits on X and transforms X
<i>load</i>	Loads component at file path
<i>needs_fitting</i>	Returns boolean determining if component needs fitting before
<i>parameters</i>	Returns the parameters which were used to initialize the component
<i>save</i>	Saves component at file path
<i>transform</i>	Transforms the input data by sampling the data.

### **clone** (*self*)

Constructs a new component with the same parameters and random state.

**Returns** A new instance of this component with identical parameters and random state.

### **default\_parameters** (*cls*)

Returns the default parameters for this component.

Our convention is that `Component.default_parameters == Component().parameters`.

**Returns** default parameters for this component.

**Return type** dict

### **describe** (*self*, *print\_name=False*, *return\_dict=False*)

Describe a component and its parameters

#### Parameters

- **print\_name** (*bool*, *optional*) – whether to print name of component
- **return\_dict** (*bool*, *optional*) – whether to return description as dictionary in the format {"name": name, "parameters": parameters}

**Returns** prints and returns dictionary

**Return type** None or dict

### **fit** (*self*, *X*, *y*)

Fits the sampler to the data.

#### Parameters

- **X** (*pd.DataFrame*) – Input features.
- **y** (*pd.Series*) – Target.

**Returns** self

**fit\_transform** (*self*, *X*, *y*)  
Fits on X and transforms X

**Parameters**

- **X** (*pd.DataFrame*) – Data to fit and transform
- **y** (*pd.Series*) – Target data

**Returns** Transformed X

**Return type** *pd.DataFrame*

**static load** (*file\_path*)  
Loads component at file path

**Parameters** **file\_path** (*str*) – Location to load file

**Returns** *ComponentBase* object

**needs\_fitting** (*self*)  
Returns boolean determining if component needs fitting before calling predict, predict\_proba, transform, or feature\_importances. This can be overridden to False for components that do not need to be fit or whose fit methods do nothing.

**property parameters** (*self*)  
Returns the parameters which were used to initialize the component

**save** (*self*, *file\_path*, *pickle\_protocol=cloudpickle.DEFAULT\_PROTOCOL*)  
Saves component at file path

**Parameters**

- **file\_path** (*str*) – Location to save file
- **pickle\_protocol** (*int*) – The pickle data stream format.

**Returns** None

**transform** (*self*, *X*, *y=None*)  
Transforms the input data by sampling the data.

**Parameters**

- **X** (*pd.DataFrame*) – Training features.
- **y** (*pd.Series*) – Target.

**Returns** Transformed features and target.

**Return type** *pd.DataFrame*, *pd.Series*

**class** evalml.pipelines.components.transformers.**URLFeaturizer** (*random\_seed=0*,  
\*\**kwargs*)

Transformer that can automatically extract features from URL.

**Parameters** **random\_seed** (*int*) – Seed for the random number generator. Defaults to 0.

**Attributes**

<b>hyper-parameter_ranges</b>	{}
<b>model_family</b>	ModelFamily.NONE
<b>modifies_features</b>	True
<b>modifies_target</b>	False
<b>name</b>	URL Featurizer

## Methods

<i>clone</i>	Constructs a new component with the same parameters and random state.
<i>default_parameters</i>	Returns the default parameters for this component.
<i>describe</i>	Describe a component and its parameters
<i>fit</i>	Fits component to data
<i>fit_transform</i>	Fits on X and transforms X
<i>load</i>	Loads component at file path
<i>needs_fitting</i>	Returns boolean determining if component needs fitting before
<i>parameters</i>	Returns the parameters which were used to initialize the component
<i>save</i>	Saves component at file path
<i>transform</i>	Transforms data X.

### **clone** (*self*)

Constructs a new component with the same parameters and random state.

**Returns** A new instance of this component with identical parameters and random state.

### **default\_parameters** (*cls*)

Returns the default parameters for this component.

Our convention is that `Component.default_parameters == Component().parameters`.

**Returns** default parameters for this component.

**Return type** dict

### **describe** (*self*, *print\_name=False*, *return\_dict=False*)

Describe a component and its parameters

#### Parameters

- **print\_name** (*bool*, *optional*) – whether to print name of component
- **return\_dict** (*bool*, *optional*) – whether to return description as dictionary in the format {"name": name, "parameters": parameters}

**Returns** prints and returns dictionary

**Return type** None or dict

### **fit** (*self*, *X*, *y=None*)

Fits component to data

#### Parameters

- **X** (*list*, *pd.DataFrame* or *np.ndarray*) – The input training data of shape [n\_samples, n\_features]
- **y** (*list*, *pd.Series*, *np.ndarray*, *optional*) – The target training data of length [n\_samples]

**Returns** *self*

**fit\_transform** (*self*, *X*, *y=None*)

Fits on X and transforms X

**Parameters**

- **X** (*pd.DataFrame*) – Data to fit and transform
- **y** (*pd.Series*) – Target data

**Returns** Transformed X

**Return type** *pd.DataFrame*

**static load** (*file\_path*)

Loads component at file path

**Parameters** **file\_path** (*str*) – Location to load file

**Returns** *ComponentBase* object

**needs\_fitting** (*self*)

Returns boolean determining if component needs fitting before calling predict, predict\_proba, transform, or feature\_importances. This can be overridden to False for components that do not need to be fit or whose fit methods do nothing.

**property parameters** (*self*)

Returns the parameters which were used to initialize the component

**save** (*self*, *file\_path*, *pickle\_protocol=cloudpickle.DEFAULT\_PROTOCOL*)

Saves component at file path

**Parameters**

- **file\_path** (*str*) – Location to save file
- **pickle\_protocol** (*int*) – The pickle data stream format.

**Returns** *None*

**transform** (*self*, *X*, *y=None*)

Transforms data X.

**Parameters**

- **X** (*pd.DataFrame*) – Data to transform.
- **y** (*pd.Series*, *optional*) – Target data.

**Returns** Transformed X

**Return type** *pd.DataFrame*

## Submodules

### component\_base

## Module Contents

## Classes Summary

<i>ComponentBase</i>	Base class for all components.
----------------------	--------------------------------

## Attributes Summary

<i>logger</i>
---------------

## Contents

**class** evalml.pipelines.components.component\_base.**ComponentBase** (*parameters=None, component\_obj=None, random\_seed=0, \*\*kwargs*)

Base class for all components.

### Parameters

- **parameters** (*dict*) – Dictionary of parameters for the component. Defaults to None.
- **component\_obj** (*obj*) – Third-party objects useful in component implementation. Defaults to None.
- **random\_seed** (*int*) – Seed for the random number generator. Defaults to 0.

### Methods

<i>clone</i>	Constructs a new component with the same parameters and random state.
<i>default_parameters</i>	Returns the default parameters for this component.
<i>describe</i>	Describe a component and its parameters
<i>fit</i>	Fits component to data
<i>load</i>	Loads component at file path
<i>model_family</i>	Returns ModelFamily of this component
<i>modifies_features</i>	Returns whether this component modifies (subsets or transforms) the features variable during transform.
<i>modifies_target</i>	Returns whether this component modifies (subsets or transforms) the target variable during transform.
<i>name</i>	Returns string name of this component
<i>needs_fitting</i>	Returns boolean determining if component needs fitting before

continues on next page

Table 534 – continued from previous page

<i>parameters</i>	Returns the parameters which were used to initialize the component
<i>save</i>	Saves component at file path

**clone** (*self*)

Constructs a new component with the same parameters and random state.

**Returns** A new instance of this component with identical parameters and random state.

**default\_parameters** (*cls*)

Returns the default parameters for this component.

Our convention is that `Component.default_parameters == Component().parameters`.

**Returns** default parameters for this component.

**Return type** dict

**describe** (*self*, *print\_name=False*, *return\_dict=False*)

Describe a component and its parameters

**Parameters**

- **print\_name** (*bool*, *optional*) – whether to print name of component
- **return\_dict** (*bool*, *optional*) – whether to return description as dictionary in the format {"name": name, "parameters": parameters}

**Returns** prints and returns dictionary

**Return type** None or dict

**fit** (*self*, *X*, *y=None*)

Fits component to data

**Parameters**

- **X** (*list*, *pd.DataFrame* or *np.ndarray*) – The input training data of shape [n\_samples, n\_features]
- **y** (*list*, *pd.Series*, *np.ndarray*, *optional*) – The target training data of length [n\_samples]

**Returns** self

**static load** (*file\_path*)

Loads component at file path

**Parameters** **file\_path** (*str*) – Location to load file

**Returns** ComponentBase object

**property model\_family** (*cls*)

Returns ModelFamily of this component

**property modifies\_features** (*cls*)

Returns whether this component modifies (subsets or transforms) the features variable during transform. For Estimator objects, this attribute determines if the return value from *predict* or *predict\_proba* should be used as features or targets.

**property modifies\_target** (*cls*)

Returns whether this component modifies (subsets or transforms) the target variable during transform. For Estimator objects, this attribute determines if the return value from *predict* or *predict\_proba* should be used as features or targets.



**property name** (*cls*)

Returns string name of this component

**needs\_fitting** (*self*)

Returns boolean determining if component needs fitting before calling predict, predict\_proba, transform, or feature\_importances. This can be overridden to False for components that do not need to be fit or whose fit methods do nothing.

**property parameters** (*self*)

Returns the parameters which were used to initialize the component

**save** (*self, file\_path, pickle\_protocol=cloudpickle.DEFAULT\_PROTOCOL*)

Saves component at file path

#### Parameters

- **file\_path** (*str*) – Location to save file
- **pickle\_protocol** (*int*) – The pickle data stream format.

**Returns** None

evalml.pipelines.components.component\_base.logger

## component\_base\_meta

### Module Contents

### Classes Summary

---

*ComponentBaseMeta*

Metaclass that overrides creating a new component by wrapping methods with validators and setters

---

### Contents

**class** evalml.pipelines.components.component\_base\_meta.**ComponentBaseMeta**

Metaclass that overrides creating a new component by wrapping methods with validators and setters

#### Attributes

<b>FIT_METHODS</b>	['fit', 'fit_transform']
<b>METHODS_TO_CHECK</b>	['predict', 'predict_proba', 'transform', 'inverse_transform']
<b>PROPERTIES_TO_CHECK</b>	['feature_importance']

#### Methods

<i>check_for_fit</i>	<i>check_for_fit</i> wraps a method that validates if <i>self.is_fitted</i> is <i>True</i> .
<i>register</i>	Register a virtual subclass of an ABC.
<i>set_fit</i>	

---

**classmethod** `check_for_fit` (*cls, method*)

*check\_for\_fit* wraps a method that validates if *self.is\_fitted* is *True*. It raises an exception if *False* and calls and returns the wrapped method if *True*.

**register** (*cls, subclass*)

Register a virtual subclass of an ABC.

Returns the subclass, to allow usage as a class decorator.

**classmethod** `set_fit` (*cls, method*)

## utils

## Module Contents

## Classes Summary

---

<code>WrappedSKClassifier</code>	Scikit-learn classifier wrapper class.
<code>WrappedSKRegressor</code>	Scikit-learn regressor wrapper class.

---

## Functions

---

<code>all_components</code>	
<code>allowed_model_families</code>	List the model types allowed for a particular problem type.
<code>generate_component_code</code>	Creates and returns a string that contains the Python imports and code required for running the EvalML component.
<code>get_estimators</code>	Returns the estimators allowed for a particular problem type.
<code>handle_component_class</code>	Standardizes input from a string name to a Component-Base subclass if necessary.
<code>make_balancing_dictionary</code>	Makes dictionary for oversampler components. Find ratio of each class to the majority.
<code>scikit_learn_wrapped_estimator</code>	

---

## Attributes Summary

---

<code>logger</code>
---------------------

---

## Contents

`evalml.pipelines.components.utils.all_components()`

`evalml.pipelines.components.utils.allowed_model_families(problem_type)`

List the model types allowed for a particular problem type.

**Parameters** *problem\_types* (*ProblemTypes* or *str*) – binary, multiclass, or regression

**Returns** a list of model families

**Return type** list[ModelFamily]

`evalml.pipelines.components.utils.generate_component_code(element)`

Creates and returns a string that contains the Python imports and code required for running the EvalML component.

**Parameters** *element* (*component instance*) – The instance of the component to generate string Python code for

**Returns** String representation of Python code that can be run separately in order to recreate the component instance. Does not include code for custom component implementation.

`evalml.pipelines.components.utils.get_estimators(problem_type,  
model_families=None)`

Returns the estimators allowed for a particular problem type.

Can also optionally filter by a list of model types.

**Parameters**

- **problem\_type** (*ProblemTypes* or *str*) – problem type to filter for
- **model\_families** (*list[ModelFamily]* or *list[str]*) – model families to filter for

**Returns** a list of estimator subclasses

**Return type** list[class]

`evalml.pipelines.components.utils.handle_component_class(component_class)`

Standardizes input from a string name to a ComponentBase subclass if necessary.

If a str is provided, will attempt to look up a ComponentBase class by that name and return a new instance. Otherwise if a ComponentBase subclass is provided, will return that without modification.

**Parameters** *component* (*str*, *ComponentBase*) – input to be standardized

**Returns** ComponentBase

`evalml.pipelines.components.utils.logger`

`evalml.pipelines.components.utils.make_balancing_dictionary(y, sampling_ratio)`

Makes dictionary for oversampler components. Find ratio of each class to the majority. If the ratio is smaller than the *sampling\_ratio*, we want to oversample, otherwise, we don't want to sample at all, and we leave the data as is.

**Parameters**

- **y** (*pd.Series*) – Target data
- **sampling\_ratio** (*float*) – The balanced ratio we want the samples to meet

**Returns** Dictionary where keys are the classes, and the corresponding values are the counts of samples for each class that will satisfy *sampling\_ratio*.

`evalml.pipelines.components.utils.scikit_learn_wrapped_estimator` (*evalml\_obj*)

**class** `evalml.pipelines.components.utils.WrappedSKClassifier` (*pipeline*)  
Scikit-learn classifier wrapper class.

#### Methods

<code>fit</code>	Fits component to data
<code>get_params</code>	Get parameters for this estimator.
<code>predict</code>	Make predictions using selected features.
<code>predict_proba</code>	Make probability estimates for labels.
<code>score</code>	Return the mean accuracy on the given test data and labels.
<code>set_params</code>	Set the parameters of this estimator.

**fit** (*self*, *X*, *y*)  
Fits component to data

#### Parameters

- **X** (*pd.DataFrame* or *np.ndarray*) – the input training data of shape [*n\_samples*, *n\_features*]
- **y** (*pd.Series*, optional) – the target training data of length [*n\_samples*]

**Returns** *self*

**get\_params** (*self*, *deep=True*)  
Get parameters for this estimator.

**Parameters** *deep* (*bool*, *default=True*) – If True, will return the parameters for this estimator and contained subobjects that are estimators.

**Returns** *params* – Parameter names mapped to their values.

**Return type** *dict*

**predict** (*self*, *X*)  
Make predictions using selected features.

**Parameters** **X** (*pd.DataFrame*) – Features

**Returns** Predicted values

**Return type** *np.ndarray*

**predict\_proba** (*self*, *X*)  
Make probability estimates for labels.

**Parameters** **X** (*pd.DataFrame*) – Features

**Returns** Probability estimates

**Return type** *np.ndarray*

**score** (*self*, *X*, *y*, *sample\_weight=None*)  
Return the mean accuracy on the given test data and labels.

In multi-label classification, this is the subset accuracy which is a harsh metric since you require for each sample that each label set be correctly predicted.

#### Parameters

- **X** (*array-like of shape (n\_samples, n\_features)*) – Test samples.

- **y** (array-like of shape (n\_samples,) or (n\_samples, n\_outputs)) – True labels for X.
- **sample\_weight** (array-like of shape (n\_samples,)), default=None) – Sample weights.

**Returns** **score** – Mean accuracy of `self.predict(X)` wrt. y.

**Return type** float

**set\_params** (*self*, **\*\*params**)

Set the parameters of this estimator.

The method works on simple estimators as well as on nested objects (such as `Pipeline`). The latter have parameters of the form `<component>__<parameter>` so that it's possible to update each component of a nested object.

**Parameters** **\*\*params** (*dict*) – Estimator parameters.

**Returns** **self** – Estimator instance.

**Return type** estimator instance

**class** evalml.pipelines.components.utils.**WrappedSKRegressor** (*pipeline*)

Scikit-learn regressor wrapper class.

**Methods**

<code>fit</code>	Fits component to data
<code>get_params</code>	Get parameters for this estimator.
<code>predict</code>	Make predictions using selected features.
<code>score</code>	Return the coefficient of determination $R^2$ of the
<code>set_params</code>	Set the parameters of this estimator.

**fit** (*self*, X, y)

Fits component to data

**Parameters**

- **X** (*pd.DataFrame* or *np.ndarray*) – the input training data of shape [n\_samples, n\_features]
- **y** (*pd.Series*, optional) – the target training data of length [n\_samples]

**Returns** **self**

**get\_params** (*self*, **deep=True**)

Get parameters for this estimator.

**Parameters** **deep** (*bool*, default=True) – If True, will return the parameters for this estimator and contained subobjects that are estimators.

**Returns** **params** – Parameter names mapped to their values.

**Return type** dict

**predict** (*self*, X)

Make predictions using selected features.

**Parameters** **X** (*pd.DataFrame*) – Features

**Returns** Predicted values

**Return type** np.ndarray

**score** (*self*, *X*, *y*, *sample\_weight=None*)

Return the coefficient of determination  $R^2$  of the prediction.

The coefficient  $R^2$  is defined as  $(1 - \frac{u}{v})$ , where  $u$  is the residual sum of squares  $((y_{\text{true}} - y_{\text{pred}}) ** 2).sum()$  and  $v$  is the total sum of squares  $((y_{\text{true}} - y_{\text{true.mean}}()) ** 2).sum()$ . The best possible score is 1.0 and it can be negative (because the model can be arbitrarily worse). A constant model that always predicts the expected value of  $y$ , disregarding the input features, would get a  $R^2$  score of 0.0.

#### Parameters

- **X** (*array-like of shape (n\_samples, n\_features)*) – Test samples. For some estimators this may be a precomputed kernel matrix or a list of generic objects instead with shape  $(n_{\text{samples}}, n_{\text{samples\_fitted}})$ , where  $n_{\text{samples\_fitted}}$  is the number of samples used in the fitting for the estimator.
- **y** (*array-like of shape (n\_samples,) or (n\_samples, n\_outputs)*) – True values for X.
- **sample\_weight** (*array-like of shape (n\_samples,)*, *default=None*) – Sample weights.

**Returns** **score** –  $R^2$  of `self.predict(X)` wrt.  $y$ .

**Return type** float

#### Notes

The  $R^2$  score used when calling `score` on a regressor uses `multioutput='uniform_average'` from version 0.23 to keep consistent with default value of `r2_score()`. This influences the `score` method of all the multioutput regressors (except for `MultiOutputRegressor`).

**set\_params** (*self*, *\*\*params*)

Set the parameters of this estimator.

The method works on simple estimators as well as on nested objects (such as `Pipeline`). The latter have parameters of the form `<component>__<parameter>` so that it's possible to update each component of a nested object.

**Parameters** **\*\*params** (*dict*) – Estimator parameters.

**Returns** **self** – Estimator instance.

**Return type** estimator instance

## Package Contents

### Classes Summary

<code>ARIMAREgressor</code>	Autoregressive Integrated Moving Average Model.
<code>BaselineClassifier</code>	Classifier that predicts using the specified strategy.
<code>BaselineRegressor</code>	Baseline regressor that uses a simple strategy to make predictions.
<code>CatBoostClassifier</code>	CatBoost Classifier, a classifier that uses gradient-boosting on decision trees.

continues on next page

Table 542 – continued from previous page

<i>CatBoostRegressor</i>	CatBoost Regressor, a regressor that uses gradient-boosting on decision trees.
<i>ComponentBase</i>	Base class for all components.
<i>ComponentBaseMeta</i>	Metaclass that overrides creating a new component by wrapping methods with validators and setters
<i>DateTimeFeaturizer</i>	Transformer that can automatically extract features from datetime columns.
<i>DecisionTreeClassifier</i>	Decision Tree Classifier.
<i>DecisionTreeRegressor</i>	Decision Tree Regressor.
<i>DelayedFeatureTransformer</i>	Transformer that delays input features and target variable for time series problems.
<i>DFSTransformer</i>	Featuretools DFS component that generates features for the input features.
<i>DropColumns</i>	Drops specified columns in input data.
<i>DropNullColumns</i>	Transformer to drop features whose percentage of NaN values exceeds a specified threshold.
<i>ElasticNetClassifier</i>	Elastic Net Classifier. Uses Logistic Regression with elasticnet penalty as the base estimator.
<i>ElasticNetRegressor</i>	Elastic Net Regressor.
<i>EmailFeaturizer</i>	Transformer that can automatically extract features from emails.
<i>Estimator</i>	A component that fits and predicts given data.
<i>ExtraTreesClassifier</i>	Extra Trees Classifier.
<i>ExtraTreesRegressor</i>	Extra Trees Regressor.
<i>FeatureSelector</i>	Selects top features based on importance weights.
<i>Imputer</i>	Imputes missing data according to a specified imputation strategy.
<i>KNeighborsClassifier</i>	K-Nearest Neighbors Classifier.
<i>LightGBMClassifier</i>	LightGBM Classifier.
<i>LightGBMRegressor</i>	LightGBM Regressor.
<i>LinearDiscriminantAnalysis</i>	Reduces the number of features by using Linear Discriminant Analysis.
<i>LinearRegressor</i>	Linear Regressor.
<i>LogisticRegressionClassifier</i>	Logistic Regression Classifier.
<i>LogTransformer</i>	Applies a log transformation to the target data.
<i>LSA</i>	Transformer to calculate the Latent Semantic Analysis Values of text input.
<i>OneHotEncoder</i>	A transformer that encodes categorical features in a one-hot numeric array.
<i>PCA</i>	Reduces the number of features by using Principal Component Analysis (PCA).
<i>PerColumnImputer</i>	Imputes missing data according to a specified imputation strategy per column.
<i>PolynomialDetrender</i>	Removes trends from time series by fitting a polynomial to the data.
<i>ProphetRegressor</i>	Prophet is a procedure for forecasting time series data based on an additive model where non-linear trends are fit with yearly, weekly, and daily seasonality, plus holiday effects.
<i>RandomForestClassifier</i>	Random Forest Classifier.
<i>RandomForestRegressor</i>	Random Forest Regressor.

continues on next page

Table 542 – continued from previous page

<i>RFClassifierSelectFromModel</i>	Selects top features based on importance weights using a Random Forest classifier.
<i>RFRegressorSelectFromModel</i>	Selects top features based on importance weights using a Random Forest regressor.
<i>SelectByType</i>	Selects columns by specified Woodwork logical type or semantic tag in input data.
<i>SelectColumns</i>	Selects specified columns in input data.
<i>SimpleImputer</i>	Imputes missing data according to a specified imputation strategy.
<i>SklearnStackedEnsembleClassifier</i>	Scikit-learn Stacked Ensemble Classifier.
<i>SklearnStackedEnsembleRegressor</i>	Scikit-learn Stacked Ensemble Regressor.
<i>SMOTENCOverSampler</i>	SMOTENC Oversampler component. Uses SMOTENC to generate synthetic samples. Works on a mix of numerical and categorical columns.
<i>SMOTENOverSampler</i>	SMOTEN Oversampler component. Uses SMOTEN to generate synthetic samples. Works for purely categorical datasets.
<i>SMOTEOverSampler</i>	SMOTE Oversampler component. Works on numerical datasets only. This component is only run during training and not during predict.
<i>StandardScaler</i>	A transformer that standardizes input features by removing the mean and scaling to unit variance.
<i>SVMClassifier</i>	Support Vector Machine Classifier.
<i>SVMRegressor</i>	Support Vector Machine Regressor.
<i>TargetEncoder</i>	A transformer that encodes categorical features into target encodings.
<i>TargetImputer</i>	Imputes missing target data according to a specified imputation strategy.
<i>TextFeaturizer</i>	Transformer that can automatically featurize text columns using featuretools' nlp_primitives.
<i>TimeSeriesBaselineEstimator</i>	Time series estimator that predicts using the naive forecasting approach.
<i>Transformer</i>	A component that may or may not need fitting that transforms data.
<i>Undersampler</i>	Initializes an undersampling transformer to downsample the majority classes in the dataset.
<i>URLFeaturizer</i>	Transformer that can automatically extract features from URL.
<i>XGBoostClassifier</i>	XGBoost Classifier.
<i>XGBoostRegressor</i>	XGBoost Regressor.

## Contents

**class** evalml.pipelines.components.**ARIMAREgressor** (*date\_index=None*, *trend=None*,  
*start\_p=2*, *d=0*, *start\_q=2*,  
*max\_p=5*, *max\_d=2*, *max\_q=5*,  
*seasonal=True*, *n\_jobs=-1*, *random\_seed=0*, *\*\*kwargs*)

Autoregressive Integrated Moving Average Model. The three parameters (p, d, q) are the AR order, the degree of differencing, and the MA order. More information here: [https://www.statsmodels.org/devel/generated/statsmodels.tsa.arima\\_model.ARIMA.html](https://www.statsmodels.org/devel/generated/statsmodels.tsa.arima_model.ARIMA.html)



Currently ARIMAREgressor isn't supported via conda install. It's recommended that it be installed via PyPI.

### Parameters

- **date\_index** (*str*) – Specifies the name of the column in X that provides the datetime objects. Defaults to None.
- **trend** (*str*) – Controls the deterministic trend. Options are ['n', 'c', 't', 'ct'] where 'c' is a constant term, 't' indicates a linear trend, and 'ct' is both. Can also be an iterable when defining a polynomial, such as [1, 1, 0, 1].
- **start\_p** (*int*) – Minimum Autoregressive order. Defaults to 2.
- **d** (*int*) – Minimum Differencing degree. Defaults to 0.
- **start\_q** (*int*) – Minimum Moving Average order. Defaults to 2.
- **max\_p** (*int*) – Maximum Autoregressive order. Defaults to 5.
- **max\_d** (*int*) – Maximum Differencing degree. Defaults to 2.
- **max\_q** (*int*) – Maximum Moving Average order. Defaults to 5.
- **seasonal** (*boolean*) – Whether to fit a seasonal model to ARIMA. Defaults to True.
- **n\_jobs** (*int or None*) – Non-negative integer describing level of parallelism used for pipelines. Defaults to -1.
- **random\_seed** (*int*) – Seed for the random number generator. Defaults to 0.

### Attributes

<b>hyper-parameter_ranges</b>	{ "start_p": Integer(1, 3), "d": Integer(0, 2), "start_q": Integer(1, 3), "max_p": Integer(3, 10), "max_d": Integer(2, 5), "max_q": Integer(3, 10), "seasonal": [True, False], }
<b>model_family</b>	ModelFamily.ARIMA
<b>modifies_features</b>	True
<b>modifies_target</b>	False
<b>name</b>	ARIMA Regressor
<b>predict_uses_y</b>	False
<b>supported_problem_types</b>	[ProblemTypes.TIME_SERIES_REGRESSION]

### Methods

<i>clone</i>	Constructs a new component with the same parameters and random state.
<i>default_parameters</i>	Returns the default parameters for this component.
<i>describe</i>	Describe a component and its parameters
<i>feature_importance</i>	Returns array of 0's with a length of 1 as feature_importance is not defined for ARIMA regressor.
<i>fit</i>	Fits component to data
<i>load</i>	Loads component at file path
<i>needs_fitting</i>	Returns boolean determining if component needs fitting before

continues on next page

Table 543 – continued from previous page

<i>parameters</i>	Returns the parameters which were used to initialize the component
<i>predict</i>	Make predictions using selected features.
<i>predict_proba</i>	Make probability estimates for labels.
<i>save</i>	Saves component at file path

**clone** (*self*)

Constructs a new component with the same parameters and random state.

**Returns** A new instance of this component with identical parameters and random state.

**default\_parameters** (*cls*)

Returns the default parameters for this component.

Our convention is that `Component.default_parameters == Component().parameters`.

**Returns** default parameters for this component.

**Return type** dict

**describe** (*self*, *print\_name=False*, *return\_dict=False*)

Describe a component and its parameters

**Parameters**

- **print\_name** (*bool*, *optional*) – whether to print name of component
- **return\_dict** (*bool*, *optional*) – whether to return description as dictionary in the format {"name": name, "parameters": parameters}

**Returns** prints and returns dictionary

**Return type** None or dict

**property feature\_importance** (*self*)

Returns array of 0's with a length of 1 as feature\_importance is not defined for ARIMA regressor.

**fit** (*self*, *X*, *y=None*)

Fits component to data

**Parameters**

- **X** (*list*, *pd.DataFrame* or *np.ndarray*) – The input training data of shape [n\_samples, n\_features]
- **y** (*list*, *pd.Series*, *np.ndarray*, *optional*) – The target training data of length [n\_samples]

**Returns** self

**static load** (*file\_path*)

Loads component at file path

**Parameters** **file\_path** (*str*) – Location to load file

**Returns** ComponentBase object

**needs\_fitting** (*self*)

Returns boolean determining if component needs fitting before calling predict, predict\_proba, transform, or feature\_importances. This can be overridden to False for components that do not need to be fit or whose fit methods do nothing.

**property parameters** (*self*)

Returns the parameters which were used to initialize the component

**predict** (*self*, *X*, *y=None*)

Make predictions using selected features.

**Parameters** **X** (*pd.DataFrame*, *np.ndarray*) – Data of shape [n\_samples, n\_features]

**Returns** Predicted values

**Return type** *pd.Series*

**predict\_proba** (*self*, *X*)

Make probability estimates for labels.

**Parameters** **X** (*pd.DataFrame*, *or np.ndarray*) – Features

**Returns** Probability estimates

**Return type** *pd.Series*

**save** (*self*, *file\_path*, *pickle\_protocol=cloudpickle.DEFAULT\_PROTOCOL*)

Saves component at file path

**Parameters**

- **file\_path** (*str*) – Location to save file
- **pickle\_protocol** (*int*) – The pickle data stream format.

**Returns** None

**class** `evalml.pipelines.components.BaselineClassifier` (*strategy='mode'*, *random\_seed=0*, *\*\*kwargs*)

Classifier that predicts using the specified strategy.

This is useful as a simple baseline classifier to compare with other classifiers.

**Parameters**

- **strategy** (*str*) – Method used to predict. Valid options are “mode”, “random” and “random\_weighted”. Defaults to “mode”.
- **random\_seed** (*int*) – Seed for the random number generator. Defaults to 0.

**Attributes**

<b>hyper-parameter_ranges</b>	{}
<b>model_family</b>	ModelFamily.BASELINE
<b>modifies_features</b>	True
<b>modifies_target</b>	False
<b>name</b>	Baseline Classifier
<b>predict_uses_y</b>	False
<b>supported_problem_types</b>	[ProblemTypes.BINARY, ProblemTypes.MULTICLASS]

**Methods**

---

<i>classes_</i>	Returns class labels. Will return None before fitting.
-----------------	--

---

continues on next page

Table 544 – continued from previous page

<code>clone</code>	Constructs a new component with the same parameters and random state.
<code>default_parameters</code>	Returns the default parameters for this component.
<code>describe</code>	Describe a component and its parameters
<code>feature_importance</code>	Returns importance associated with each feature. Since baseline classifiers do not use input features to calculate predictions, returns an array of zeroes.
<code>fit</code>	Fits component to data
<code>load</code>	Loads component at file path
<code>needs_fitting</code>	Returns boolean determining if component needs fitting before
<code>parameters</code>	Returns the parameters which were used to initialize the component
<code>predict</code>	Make predictions using selected features.
<code>predict_proba</code>	Make probability estimates for labels.
<code>save</code>	Saves component at file path

**property classes\_**(*self*)

Returns class labels. Will return None before fitting.

**Returns** Class names

**Return type** list[str] or list(float)

**clone**(*self*)

Constructs a new component with the same parameters and random state.

**Returns** A new instance of this component with identical parameters and random state.

**default\_parameters**(*cls*)

Returns the default parameters for this component.

Our convention is that `Component.default_parameters == Component().parameters`.

**Returns** default parameters for this component.

**Return type** dict

**describe**(*self*, *print\_name=False*, *return\_dict=False*)

Describe a component and its parameters

**Parameters**

- **print\_name** (*bool*, *optional*) – whether to print name of component
- **return\_dict** (*bool*, *optional*) – whether to return description as dictionary in the format {"name": name, "parameters": parameters}

**Returns** prints and returns dictionary

**Return type** None or dict

**property feature\_importance**(*self*)

Returns importance associated with each feature. Since baseline classifiers do not use input features to calculate predictions, returns an array of zeroes.

**Returns** An array of zeroes

**Return type** np.ndarray (float)

**fit** (*self*, *X*, *y=None*)

Fits component to data

**Parameters**

- **X** (*list*, *pd.DataFrame* or *np.ndarray*) – The input training data of shape [n\_samples, n\_features]
- **y** (*list*, *pd.Series*, *np.ndarray*, *optional*) – The target training data of length [n\_samples]

**Returns** *self*

**static load** (*file\_path*)

Loads component at file path

**Parameters** **file\_path** (*str*) – Location to load file

**Returns** ComponentBase object

**needs\_fitting** (*self*)

Returns boolean determining if component needs fitting before calling predict, predict\_proba, transform, or feature\_importances. This can be overridden to False for components that do not need to be fit or whose fit methods do nothing.

**property parameters** (*self*)

Returns the parameters which were used to initialize the component

**predict** (*self*, *X*)

Make predictions using selected features.

**Parameters** **X** (*pd.DataFrame*, *np.ndarray*) – Data of shape [n\_samples, n\_features]

**Returns** Predicted values

**Return type** *pd.Series*

**predict\_proba** (*self*, *X*)

Make probability estimates for labels.

**Parameters** **X** (*pd.DataFrame*, or *np.ndarray*) – Features

**Returns** Probability estimates

**Return type** *pd.Series*

**save** (*self*, *file\_path*, *pickle\_protocol=cloudpickle.DEFAULT\_PROTOCOL*)

Saves component at file path

**Parameters**

- **file\_path** (*str*) – Location to save file
- **pickle\_protocol** (*int*) – The pickle data stream format.

**Returns** None

**class** evalml.pipelines.components.**BaselineRegressor** (*strategy='mean'*, *random\_seed=0*, *\*\*kwargs*)

Baseline regressor that uses a simple strategy to make predictions. This is useful as a simple baseline regressor to compare with other regressors.

**Parameters**

- **strategy** (*str*) – Method used to predict. Valid options are “mean”, “median”. Defaults to “mean”.

- **random\_seed** (*int*) – Seed for the random number generator. Defaults to 0.

#### Attributes

<b>hyper-parameter_ranges</b>	{}
<b>model_family</b>	ModelFamily.BASELINE
<b>modifies_features</b>	True
<b>modifies_target</b>	False
<b>name</b>	Baseline Regressor
<b>pre-dict_uses_y</b>	False
<b>supported_problem_types</b>	[ ProblemTypes.REGRESSION, ProblemTypes.TIME_SERIES_REGRESSION,]

#### Methods

<i>clone</i>	Constructs a new component with the same parameters and random state.
<i>default_parameters</i>	Returns the default parameters for this component.
<i>describe</i>	Describe a component and its parameters
<i>feature_importance</i>	Returns importance associated with each feature. Since baseline regressors do not use input features to calculate predictions, returns an array of zeroes.
<i>fit</i>	Fits component to data
<i>load</i>	Loads component at file path
<i>needs_fitting</i>	Returns boolean determining if component needs fitting before
<i>parameters</i>	Returns the parameters which were used to initialize the component
<i>predict</i>	Make predictions using selected features.
<i>predict_proba</i>	Make probability estimates for labels.
<i>save</i>	Saves component at file path

#### **clone** (*self*)

Constructs a new component with the same parameters and random state.

**Returns** A new instance of this component with identical parameters and random state.

#### **default\_parameters** (*cls*)

Returns the default parameters for this component.

Our convention is that `Component.default_parameters == Component().parameters`.

**Returns** default parameters for this component.

**Return type** dict

#### **describe** (*self*, *print\_name=False*, *return\_dict=False*)

Describe a component and its parameters

#### Parameters

- **print\_name** (*bool*, *optional*) – whether to print name of component

- **return\_dict** (*bool, optional*) – whether to return description as dictionary in the format {"name": name, "parameters": parameters}

**Returns** prints and returns dictionary

**Return type** None or dict

**property feature\_importance** (*self*)

Returns importance associated with each feature. Since baseline regressors do not use input features to calculate predictions, returns an array of zeroes.

**Returns** An array of zeroes

**Return type** np.ndarray (float)

**fit** (*self, X, y=None*)

Fits component to data

**Parameters**

- **X** (*list, pd.DataFrame or np.ndarray*) – The input training data of shape [n\_samples, n\_features]
- **y** (*list, pd.Series, np.ndarray, optional*) – The target training data of length [n\_samples]

**Returns** self

**static load** (*file\_path*)

Loads component at file path

**Parameters** **file\_path** (*str*) – Location to load file

**Returns** ComponentBase object

**needs\_fitting** (*self*)

Returns boolean determining if component needs fitting before calling predict, predict\_proba, transform, or feature\_importances. This can be overridden to False for components that do not need to be fit or whose fit methods do nothing.

**property parameters** (*self*)

Returns the parameters which were used to initialize the component

**predict** (*self, X*)

Make predictions using selected features.

**Parameters** **X** (*pd.DataFrame, np.ndarray*) – Data of shape [n\_samples, n\_features]

**Returns** Predicted values

**Return type** pd.Series

**predict\_proba** (*self, X*)

Make probability estimates for labels.

**Parameters** **X** (*pd.DataFrame, or np.ndarray*) – Features

**Returns** Probability estimates

**Return type** pd.Series

**save** (*self, file\_path, pickle\_protocol=cloudpickle.DEFAULT\_PROTOCOL*)

Saves component at file path

**Parameters**

- **file\_path** (*str*) – Location to save file

- **pickle\_protocol** (*int*) – The pickle data stream format.

**Returns** None

```
class evalml.pipelines.components.CatBoostClassifier (n_estimators=10, eta=0.03,  
                                                    max_depth=6, bootstrap_type=None, silent=True,  
                                                    allow_writing_files=False,  
                                                    random_seed=0, n_jobs=-1,  
                                                    **kwargs)
```

CatBoost Classifier, a classifier that uses gradient-boosting on decision trees. CatBoost is an open-source library and natively supports categorical features.

For more information, check out <https://catboost.ai/>

### Parameters

- **n\_estimators** (*float*) – The maximum number of trees to build. Defaults to 10.
- **eta** (*float*) – The learning rate. Defaults to 0.03.
- **max\_depth** (*int*) – The maximum tree depth for base learners. Defaults to 6.
- **bootstrap\_type** (*string*) – Defines the method for sampling the weights of objects. Available methods are ‘Bayesian’, ‘Bernoulli’, ‘MVS’. Defaults to None.
- **silent** (*boolean*) – Whether to use the “silent” logging mode. Defaults to True.
- **allow\_writing\_files** (*boolean*) – Whether to allow writing snapshot files while training. Defaults to False.
- **n\_jobs** (*int or None*) – Number of jobs to run in parallel. -1 uses all processes. Defaults to -1.
- **random\_seed** (*int*) – Seed for the random number generator. Defaults to 0.

### Attributes

<b>hyper-parameter_ranges</b>	{ “n_estimators”: Integer(4, 100), “eta”: Real(0.000001, 1), “max_depth”: Integer(4, 10), }
<b>model_family</b>	ModelFamily.CATBOOST
<b>modifies_features</b>	True
<b>modifies_target</b>	False
<b>name</b>	CatBoost Classifier
<b>pre-dict_uses_y</b>	False
<b>supported_problem_types</b>	[ ProblemTypes.BINARY, ProblemTypes.MULTICLASS, ProblemTypes.TIME_SERIES_BINARY, ProblemTypes.TIME_SERIES_MULTICLASS, ]

### Methods

<i>clone</i>	Constructs a new component with the same parameters and random state.
<i>default_parameters</i>	Returns the default parameters for this component.
<i>describe</i>	Describe a component and its parameters

continues on next page



Table 546 – continued from previous page

<code>feature_importance</code>	Returns importance associated with each feature.
<code>fit</code>	Fits component to data
<code>load</code>	Loads component at file path
<code>needs_fitting</code>	Returns boolean determining if component needs fitting before
<code>parameters</code>	Returns the parameters which were used to initialize the component
<code>predict</code>	Make predictions using selected features.
<code>predict_proba</code>	Make probability estimates for labels.
<code>save</code>	Saves component at file path

**clone** (*self*)

Constructs a new component with the same parameters and random state.

**Returns** A new instance of this component with identical parameters and random state.

**default\_parameters** (*cls*)

Returns the default parameters for this component.

Our convention is that `Component.default_parameters == Component().parameters`.

**Returns** default parameters for this component.

**Return type** dict

**describe** (*self*, *print\_name=False*, *return\_dict=False*)

Describe a component and its parameters

**Parameters**

- **print\_name** (*bool*, *optional*) – whether to print name of component
- **return\_dict** (*bool*, *optional*) – whether to return description as dictionary in the format {"name": name, "parameters": parameters}

**Returns** prints and returns dictionary

**Return type** None or dict

**property feature\_importance** (*self*)

Returns importance associated with each feature.

**Returns** Importance associated with each feature

**Return type** np.ndarray

**fit** (*self*, *X*, *y=None*)

Fits component to data

**Parameters**

- **X** (*list*, *pd.DataFrame* or *np.ndarray*) – The input training data of shape [n\_samples, n\_features]
- **y** (*list*, *pd.Series*, *np.ndarray*, *optional*) – The target training data of length [n\_samples]

**Returns** self

**static load** (*file\_path*)

Loads component at file path

**Parameters** **file\_path** (*str*) – Location to load file

**Returns** ComponentBase object

**needs\_fitting** (*self*)

Returns boolean determining if component needs fitting before calling predict, predict\_proba, transform, or feature\_importances. This can be overridden to False for components that do not need to be fit or whose fit methods do nothing.

**property parameters** (*self*)

Returns the parameters which were used to initialize the component

**predict** (*self*, *X*)

Make predictions using selected features.

**Parameters** *X* (*pd.DataFrame*, *np.ndarray*) – Data of shape [n\_samples, n\_features]

**Returns** Predicted values

**Return type** *pd.Series*

**predict\_proba** (*self*, *X*)

Make probability estimates for labels.

**Parameters** *X* (*pd.DataFrame*, or *np.ndarray*) – Features

**Returns** Probability estimates

**Return type** *pd.Series*

**save** (*self*, *file\_path*, *pickle\_protocol=cloudpickle.DEFAULT\_PROTOCOL*)

Saves component at file path

**Parameters**

- **file\_path** (*str*) – Location to save file
- **pickle\_protocol** (*int*) – The pickle data stream format.

**Returns** None

```
class evalml.pipelines.components.CatBoostRegressor (n_estimators=10, eta=0.03,
                                                    max_depth=6, bootstrap_type=None, silent=False,
                                                    allow_writing_files=False,
                                                    random_seed=0, n_jobs=-1,
                                                    **kwargs)
```

CatBoost Regressor, a regressor that uses gradient-boosting on decision trees. CatBoost is an open-source library and natively supports categorical features.

For more information, check out <https://catboost.ai/>

**Parameters**

- **n\_estimators** (*float*) – The maximum number of trees to build. Defaults to 10.
- **eta** (*float*) – The learning rate. Defaults to 0.03.
- **max\_depth** (*int*) – The maximum tree depth for base learners. Defaults to 6.
- **bootstrap\_type** (*string*) – Defines the method for sampling the weights of objects. Available methods are ‘Bayesian’, ‘Bernoulli’, ‘MVS’. Defaults to None.
- **silent** (*boolean*) – Whether to use the “silent” logging mode. Defaults to True.
- **allow\_writing\_files** (*boolean*) – Whether to allow writing snapshot files while training. Defaults to False.

- **n\_jobs** (*int* or *None*) – Number of jobs to run in parallel. -1 uses all processes. Defaults to -1.
- **random\_seed** (*int*) – Seed for the random number generator. Defaults to 0.

#### Attributes

<b>hyper-parameter_ranges</b>	{ “n_estimators”: Integer(4, 100), “eta”: Real(0.000001, 1), “max_depth”: Integer(4, 10), }
<b>model_family</b>	ModelFamily.CATBOOST
<b>modifies_features</b>	True
<b>modifies_target</b>	False
<b>name</b>	CatBoost Regressor
<b>predict_uses_y</b>	False
<b>supported_problem_types</b>	[ ProblemTypes.REGRESSION, ProblemTypes.TIME_SERIES_REGRESSION, ]

#### Methods

<i>clone</i>	Constructs a new component with the same parameters and random state.
<i>default_parameters</i>	Returns the default parameters for this component.
<i>describe</i>	Describe a component and its parameters
<i>feature_importance</i>	Returns importance associated with each feature.
<i>fit</i>	Fits component to data
<i>load</i>	Loads component at file path
<i>needs_fitting</i>	Returns boolean determining if component needs fitting before
<i>parameters</i>	Returns the parameters which were used to initialize the component
<i>predict</i>	Make predictions using selected features.
<i>predict_proba</i>	Make probability estimates for labels.
<i>save</i>	Saves component at file path

#### **clone** (*self*)

Constructs a new component with the same parameters and random state.

**Returns** A new instance of this component with identical parameters and random state.

#### **default\_parameters** (*cls*)

Returns the default parameters for this component.

Our convention is that `Component.default_parameters == Component().parameters`.

**Returns** default parameters for this component.

**Return type** dict

#### **describe** (*self*, *print\_name=False*, *return\_dict=False*)

Describe a component and its parameters

**Parameters**

- **print\_name** (*bool, optional*) – whether to print name of component
- **return\_dict** (*bool, optional*) – whether to return description as dictionary in the format {"name": name, "parameters": parameters}

**Returns** prints and returns dictionary

**Return type** None or dict

**property feature\_importance** (*self*)

Returns importance associated with each feature.

**Returns** Importance associated with each feature

**Return type** np.ndarray

**fit** (*self, X, y=None*)

Fits component to data

**Parameters**

- **X** (*list, pd.DataFrame or np.ndarray*) – The input training data of shape [n\_samples, n\_features]
- **y** (*list, pd.Series, np.ndarray, optional*) – The target training data of length [n\_samples]

**Returns** self

**static load** (*file\_path*)

Loads component at file path

**Parameters** **file\_path** (*str*) – Location to load file

**Returns** ComponentBase object

**needs\_fitting** (*self*)

Returns boolean determining if component needs fitting before calling predict, predict\_proba, transform, or feature\_importances. This can be overridden to False for components that do not need to be fit or whose fit methods do nothing.

**property parameters** (*self*)

Returns the parameters which were used to initialize the component

**predict** (*self, X*)

Make predictions using selected features.

**Parameters** **X** (*pd.DataFrame, np.ndarray*) – Data of shape [n\_samples, n\_features]

**Returns** Predicted values

**Return type** pd.Series

**predict\_proba** (*self, X*)

Make probability estimates for labels.

**Parameters** **X** (*pd.DataFrame, or np.ndarray*) – Features

**Returns** Probability estimates

**Return type** pd.Series

**save** (*self, file\_path, pickle\_protocol=cloudpickle.DEFAULT\_PROTOCOL*)

Saves component at file path

**Parameters**

- **file\_path** (*str*) – Location to save file
- **pickle\_protocol** (*int*) – The pickle data stream format.

**Returns** None

```
class evalml.pipelines.components.ComponentBase (parameters=None, component_obj=None, random_seed=0, **kwargs)
```

Base class for all components.

#### Parameters

- **parameters** (*dict*) – Dictionary of parameters for the component. Defaults to None.
- **component\_obj** (*obj*) – Third-party objects useful in component implementation. Defaults to None.
- **random\_seed** (*int*) – Seed for the random number generator. Defaults to 0.

#### Methods

<i>clone</i>	Constructs a new component with the same parameters and random state.
<i>default_parameters</i>	Returns the default parameters for this component.
<i>describe</i>	Describe a component and its parameters
<i>fit</i>	Fits component to data
<i>load</i>	Loads component at file path
<i>model_family</i>	Returns ModelFamily of this component
<i>modifies_features</i>	Returns whether this component modifies (subsets or transforms) the features variable during transform.
<i>modifies_target</i>	Returns whether this component modifies (subsets or transforms) the target variable during transform.
<i>name</i>	Returns string name of this component
<i>needs_fitting</i>	Returns boolean determining if component needs fitting before
<i>parameters</i>	Returns the parameters which were used to initialize the component
<i>save</i>	Saves component at file path

**clone** (*self*)

Constructs a new component with the same parameters and random state.

**Returns** A new instance of this component with identical parameters and random state.

**default\_parameters** (*cls*)

Returns the default parameters for this component.

Our convention is that `Component.default_parameters == Component().parameters`.

**Returns** default parameters for this component.

**Return type** dict

**describe** (*self*, *print\_name=False*, *return\_dict=False*)

Describe a component and its parameters

#### Parameters

- **print\_name** (*bool*, *optional*) – whether to print name of component

- **return\_dict** (*bool, optional*) – whether to return description as dictionary in the format {"name": name, "parameters": parameters}

**Returns** prints and returns dictionary

**Return type** None or dict

**fit** (*self, X, y=None*)

Fits component to data

**Parameters**

- **X** (*list, pd.DataFrame or np.ndarray*) – The input training data of shape [n\_samples, n\_features]
- **y** (*list, pd.Series, np.ndarray, optional*) – The target training data of length [n\_samples]

**Returns** self

**static load** (*file\_path*)

Loads component at file path

**Parameters** **file\_path** (*str*) – Location to load file

**Returns** ComponentBase object

**property model\_family** (*cls*)

Returns ModelFamily of this component

**property modifies\_features** (*cls*)

Returns whether this component modifies (subsets or transforms) the features variable during transform. For Estimator objects, this attribute determines if the return value from *predict* or *predict\_proba* should be used as features or targets.

**property modifies\_target** (*cls*)

Returns whether this component modifies (subsets or transforms) the target variable during transform. For Estimator objects, this attribute determines if the return value from *predict* or *predict\_proba* should be used as features or targets.

**property name** (*cls*)

Returns string name of this component

**needs\_fitting** (*self*)

Returns boolean determining if component needs fitting before calling *predict*, *predict\_proba*, *transform*, or *feature\_importances*. This can be overridden to False for components that do not need to be fit or whose fit methods do nothing.

**property parameters** (*self*)

Returns the parameters which were used to initialize the component

**save** (*self, file\_path, pickle\_protocol=cloudpickle.DEFAULT\_PROTOCOL*)

Saves component at file path

**Parameters**

- **file\_path** (*str*) – Location to save file
- **pickle\_protocol** (*int*) – The pickle data stream format.

**Returns** None

**class** evalml.pipelines.components.ComponentBaseMeta

Metaclass that overrides creating a new component by wrapping methods with validators and setters

### Attributes

<b>FIT_METHODS</b>	['fit', 'fit_transform']
<b>METHODS_TO_CHECK</b>	['predict', 'predict_proba', 'transform', 'inverse_transform']
<b>PROPERTIES_TO_CHECK</b>	['feature_importance']

### Methods

<code>check_for_fit</code>	<code>check_for_fit</code> wraps a method that validates if <code>self.is_fitted</code> is <code>True</code> .
<code>register</code>	Register a virtual subclass of an ABC.
<code>set_fit</code>	

**classmethod** `check_for_fit` (*cls*, *method*)

`check_for_fit` wraps a method that validates if `self.is_fitted` is `True`. It raises an exception if `False` and calls and returns the wrapped method if `True`.

**register** (*cls*, *subclass*)

Register a virtual subclass of an ABC.

Returns the subclass, to allow usage as a class decorator.

**classmethod** `set_fit` (*cls*, *method*)

**class** `evalml.pipelines.components.DateTimeFeaturizer` (*features\_to\_extract=None*, *encode\_as\_categories=False*, *date\_index=None*, *random\_seed=0*, *\*\*kwargs*)

Transformer that can automatically extract features from datetime columns.

#### Parameters

- **features\_to\_extract** (*list*) – List of features to extract. Valid options include “year”, “month”, “day\_of\_week”, “hour”. Defaults to `None`.
- **encode\_as\_categories** (*bool*) – Whether day-of-week and month features should be encoded as pandas “category” dtype. This allows OneHotEncoders to encode these features. Defaults to `False`.
- **date\_index** (*str*) – Name of the column containing the datetime information used to order the data. Ignored.
- **random\_seed** (*int*) – Seed for the random number generator. Defaults to 0.

### Attributes

<b>hyper-parameter_ranges</b>	{}
<b>model_family</b>	ModelFamily.NONE
<b>modifies_features</b>	True
<b>modifies_target</b>	False
<b>name</b>	DateTime Featurization Component

## Methods

<i>clone</i>	Constructs a new component with the same parameters and random state.
<i>default_parameters</i>	Returns the default parameters for this component.
<i>describe</i>	Describe a component and its parameters
<i>fit</i>	Fits component to data
<i>fit_transform</i>	Fits on X and transforms X
<i>get_feature_names</i>	Gets the categories of each datetime feature.
<i>load</i>	Loads component at file path
<i>needs_fitting</i>	Returns boolean determining if component needs fitting before
<i>parameters</i>	Returns the parameters which were used to initialize the component
<i>save</i>	Saves component at file path
<i>transform</i>	Transforms data X by creating new features using existing DateTime columns, and then dropping those DateTime columns

### **clone** (*self*)

Constructs a new component with the same parameters and random state.

**Returns** A new instance of this component with identical parameters and random state.

### **default\_parameters** (*cls*)

Returns the default parameters for this component.

Our convention is that `Component.default_parameters == Component().parameters`.

**Returns** default parameters for this component.

**Return type** dict

### **describe** (*self*, *print\_name=False*, *return\_dict=False*)

Describe a component and its parameters

#### Parameters

- **print\_name** (*bool*, *optional*) – whether to print name of component
- **return\_dict** (*bool*, *optional*) – whether to return description as dictionary in the format {"name": name, "parameters": parameters}

**Returns** prints and returns dictionary

**Return type** None or dict



**fit** (*self*, *X*, *y=None*)

Fits component to data

**Parameters**

- **X** (*list*, *pd.DataFrame* or *np.ndarray*) – The input training data of shape [n\_samples, n\_features]
- **y** (*list*, *pd.Series*, *np.ndarray*, *optional*) – The target training data of length [n\_samples]

**Returns** *self*

**fit\_transform** (*self*, *X*, *y=None*)

Fits on X and transforms X

**Parameters**

- **X** (*pd.DataFrame*) – Data to fit and transform
- **y** (*pd.Series*) – Target data

**Returns** Transformed X

**Return type** *pd.DataFrame*

**get\_feature\_names** (*self*)

Gets the categories of each datetime feature.

**Returns** Dictionary, where each key-value pair is a column name and a dictionary mapping the unique feature values to their integer encoding.

**static load** (*file\_path*)

Loads component at file path

**Parameters** **file\_path** (*str*) – Location to load file

**Returns** *ComponentBase* object

**needs\_fitting** (*self*)

Returns boolean determining if component needs fitting before calling predict, predict\_proba, transform, or feature\_importances. This can be overridden to False for components that do not need to be fit or whose fit methods do nothing.

**property parameters** (*self*)

Returns the parameters which were used to initialize the component

**save** (*self*, *file\_path*, *pickle\_protocol=cloudpickle.DEFAULT\_PROTOCOL*)

Saves component at file path

**Parameters**

- **file\_path** (*str*) – Location to save file
- **pickle\_protocol** (*int*) – The pickle data stream format.

**Returns** *None*

**transform** (*self*, *X*, *y=None*)

Transforms data X by creating new features using existing DateTime columns, and then dropping those DateTime columns

**Parameters**

- **X** (*pd.DataFrame*) – Data to transform
- **y** (*pd.Series*, *optional*) – Ignored.

**Returns** Transformed X

**Return type** `pd.DataFrame`

```
class evalml.pipelines.components.DecisionTreeClassifier (criterion='gini',  
                                                         max_features='auto',  
                                                         max_depth=6,  
                                                         min_samples_split=2,  
                                                         min_weight_fraction_leaf=0.0,  
                                                         random_seed=0,  
                                                         **kwargs)
```

Decision Tree Classifier.

#### Parameters

- **criterion** (`{ "gini", "entropy" }`) – The function to measure the quality of a split. Supported criteria are “gini” for the Gini impurity and “entropy” for the information gain. Defaults to “gini”.
- **max\_features** (*int, float or { "auto", "sqrt", "log2" }*) – The number of features to consider when looking for the best split:
  - If `int`, then consider `max_features` features at each split.
  - If `float`, then `max_features` is a fraction and `int(max_features * n_features)` features are considered at each split.
  - If “auto”, then `max_features=sqrt(n_features)`.
  - If “sqrt”, then `max_features=sqrt(n_features)`.
  - If “log2”, then `max_features=log2(n_features)`.
  - If `None`, then `max_features = n_features`.

The search for a split does not stop until at least one valid partition of the node samples is found, even if it requires to effectively inspect more than `max_features` features. Defaults to “auto”.

- **max\_depth** (*int*) – The maximum depth of the tree. Defaults to 6.
- **min\_samples\_split** (*int or float*) – The minimum number of samples required to split an internal node:
  - If `int`, then consider `min_samples_split` as the minimum number.
  - If `float`, then `min_samples_split` is a fraction and `ceil(min_samples_split * n_samples)` are the minimum number of samples for each split.

Defaults to 2.

- **min\_weight\_fraction\_leaf** (*float*) – The minimum weighted fraction of the sum total of weights (of all the input samples) required to be at a leaf node. Defaults to 0.0.
- **random\_seed** (*int*) – Seed for the random number generator. Defaults to 0.

#### Attributes

<b>hyper-parameter_ranges</b>	{ "criterion": ["gini", "entropy"], "max_features": ["auto", "sqrt", "log2"], "max_depth": Integer(4, 10), }
<b>model_family</b>	ModelFamily.DECISION_TREE
<b>modifies_features</b>	True
<b>modifies_target</b>	False
<b>name</b>	Decision Tree Classifier
<b>pre-dict_uses_y</b>	False
<b>supported_problem_types</b>	[ ProblemTypes.BINARY, ProblemTypes.MULTICLASS, ProblemTypes.TIME_SERIES_BINARY, ProblemTypes.TIME_SERIES_MULTICLASS, ]

## Methods

<i>clone</i>	Constructs a new component with the same parameters and random state.
<i>default_parameters</i>	Returns the default parameters for this component.
<i>describe</i>	Describe a component and its parameters
<i>feature_importance</i>	Returns importance associated with each feature.
<i>fit</i>	Fits component to data
<i>load</i>	Loads component at file path
<i>needs_fitting</i>	Returns boolean determining if component needs fitting before
<i>parameters</i>	Returns the parameters which were used to initialize the component
<i>predict</i>	Make predictions using selected features.
<i>predict_proba</i>	Make probability estimates for labels.
<i>save</i>	Saves component at file path

### **clone** (*self*)

Constructs a new component with the same parameters and random state.

**Returns** A new instance of this component with identical parameters and random state.

### **default\_parameters** (*cls*)

Returns the default parameters for this component.

Our convention is that `Component.default_parameters == Component().parameters`.

**Returns** default parameters for this component.

**Return type** dict

### **describe** (*self*, *print\_name=False*, *return\_dict=False*)

Describe a component and its parameters

#### Parameters

- **print\_name** (*bool*, *optional*) – whether to print name of component
- **return\_dict** (*bool*, *optional*) – whether to return description as dictionary in the format {"name": name, "parameters": parameters}

**Returns** prints and returns dictionary

**Return type** None or dict

**property feature\_importance** (*self*)

Returns importance associated with each feature.

**Returns** Importance associated with each feature

**Return type** np.ndarray

**fit** (*self*, *X*, *y=None*)

Fits component to data

**Parameters**

- **X** (*list*, *pd.DataFrame* or *np.ndarray*) – The input training data of shape [n\_samples, n\_features]
- **y** (*list*, *pd.Series*, *np.ndarray*, *optional*) – The target training data of length [n\_samples]

**Returns** self

**static load** (*file\_path*)

Loads component at file path

**Parameters** **file\_path** (*str*) – Location to load file

**Returns** ComponentBase object

**needs\_fitting** (*self*)

Returns boolean determining if component needs fitting before calling predict, predict\_proba, transform, or feature\_importances. This can be overridden to False for components that do not need to be fit or whose fit methods do nothing.

**property parameters** (*self*)

Returns the parameters which were used to initialize the component

**predict** (*self*, *X*)

Make predictions using selected features.

**Parameters** **X** (*pd.DataFrame*, *np.ndarray*) – Data of shape [n\_samples, n\_features]

**Returns** Predicted values

**Return type** pd.Series

**predict\_proba** (*self*, *X*)

Make probability estimates for labels.

**Parameters** **X** (*pd.DataFrame*, or *np.ndarray*) – Features

**Returns** Probability estimates

**Return type** pd.Series

**save** (*self*, *file\_path*, *pickle\_protocol=cloudpickle.DEFAULT\_PROTOCOL*)

Saves component at file path

**Parameters**

- **file\_path** (*str*) – Location to save file
- **pickle\_protocol** (*int*) – The pickle data stream format.

**Returns** None

```
class evalml.pipelines.components.DecisionTreeRegressor (criterion='mse',
                                                    max_features='auto',
                                                    max_depth=6,
                                                    min_samples_split=2,
                                                    min_weight_fraction_leaf=0.0,
                                                    random_seed=0,
                                                    **kwargs)
```

Decision Tree Regressor.

### Parameters

- **criterion** (*{ "mse", "friedman\_mse", "mae", "poisson" }*) – The function to measure the quality of a split. Supported criteria are:
  - “mse” for the mean squared error, which is equal to variance reduction as feature selection criterion and minimizes the L2 loss using the mean of each terminal node
  - “friedman\_mse”, which uses mean squared error with Friedman’s improvement score for potential splits
  - “mae” for the mean absolute error, which minimizes the L1 loss using the median of each terminal node,
  - “poisson” which uses reduction in Poisson deviance to find splits.
- **max\_features** (*int, float or { "auto", "sqrt", "log2" }*) – The number of features to consider when looking for the best split:
  - If int, then consider max\_features features at each split.
  - If float, then max\_features is a fraction and  $\text{int}(\text{max\_features} * \text{n\_features})$  features are considered at each split.
  - If “auto”, then  $\text{max\_features} = \sqrt{\text{n\_features}}$ .
  - If “sqrt”, then  $\text{max\_features} = \sqrt{\text{n\_features}}$ .
  - If “log2”, then  $\text{max\_features} = \log_2(\text{n\_features})$ .
  - If None, then  $\text{max\_features} = \text{n\_features}$ .

The search for a split does not stop until at least one valid partition of the node samples is found, even if it requires to effectively inspect more than max\_features features.

- **max\_depth** (*int*) – The maximum depth of the tree. Defaults to 6.
- **min\_samples\_split** (*int or float*) – The minimum number of samples required to split an internal node:
  - If int, then consider min\_samples\_split as the minimum number.
  - If float, then min\_samples\_split is a fraction and  $\text{ceil}(\text{min\_samples\_split} * \text{n\_samples})$  are the minimum number of samples for each split.

Defaults to 2.

- **min\_weight\_fraction\_leaf** (*float*) – The minimum weighted fraction of the sum total of weights (of all the input samples) required to be at a leaf node. Defaults to 0.0.
- **random\_seed** (*int*) – Seed for the random number generator. Defaults to 0.

### Attributes

<b>hyper-parameter_ranges</b>	{ "criterion": ["mse", "friedman_mse", "mae"], "max_features": ["auto", "sqrt", "log2"], "max_depth": Integer(4, 10), }
<b>model_family</b>	ModelFamily.DECISION_TREE
<b>modifies_features</b>	True
<b>modifies_target</b>	False
<b>name</b>	Decision Tree Regressor
<b>pre-dict_uses_y</b>	False
<b>supported_problem_types</b>	[ ProblemTypes.REGRESSION, ProblemTypes.TIME_SERIES_REGRESSION, ]

## Methods

<i>clone</i>	Constructs a new component with the same parameters and random state.
<i>default_parameters</i>	Returns the default parameters for this component.
<i>describe</i>	Describe a component and its parameters
<i>feature_importance</i>	Returns importance associated with each feature.
<i>fit</i>	Fits component to data
<i>load</i>	Loads component at file path
<i>needs_fitting</i>	Returns boolean determining if component needs fitting before
<i>parameters</i>	Returns the parameters which were used to initialize the component
<i>predict</i>	Make predictions using selected features.
<i>predict_proba</i>	Make probability estimates for labels.
<i>save</i>	Saves component at file path

### **clone** (*self*)

Constructs a new component with the same parameters and random state.

**Returns** A new instance of this component with identical parameters and random state.

### **default\_parameters** (*cls*)

Returns the default parameters for this component.

Our convention is that `Component.default_parameters == Component().parameters`.

**Returns** default parameters for this component.

**Return type** dict

### **describe** (*self*, *print\_name=False*, *return\_dict=False*)

Describe a component and its parameters

#### Parameters

- **print\_name** (*bool*, *optional*) – whether to print name of component
- **return\_dict** (*bool*, *optional*) – whether to return description as dictionary in the format {"name": name, "parameters": parameters}

**Returns** prints and returns dictionary

**Return type** None or dict

**property feature\_importance** (*self*)

Returns importance associated with each feature.

**Returns** Importance associated with each feature

**Return type** np.ndarray

**fit** (*self*, *X*, *y=None*)

Fits component to data

**Parameters**

- **X** (*list*, *pd.DataFrame* or *np.ndarray*) – The input training data of shape [n\_samples, n\_features]
- **y** (*list*, *pd.Series*, *np.ndarray*, *optional*) – The target training data of length [n\_samples]

**Returns** self

**static load** (*file\_path*)

Loads component at file path

**Parameters** **file\_path** (*str*) – Location to load file

**Returns** ComponentBase object

**needs\_fitting** (*self*)

Returns boolean determining if component needs fitting before calling predict, predict\_proba, transform, or feature\_importances. This can be overridden to False for components that do not need to be fit or whose fit methods do nothing.

**property parameters** (*self*)

Returns the parameters which were used to initialize the component

**predict** (*self*, *X*)

Make predictions using selected features.

**Parameters** **X** (*pd.DataFrame*, *np.ndarray*) – Data of shape [n\_samples, n\_features]

**Returns** Predicted values

**Return type** pd.Series

**predict\_proba** (*self*, *X*)

Make probability estimates for labels.

**Parameters** **X** (*pd.DataFrame*, or *np.ndarray*) – Features

**Returns** Probability estimates

**Return type** pd.Series

**save** (*self*, *file\_path*, *pickle\_protocol=cloudpickle.DEFAULT\_PROTOCOL*)

Saves component at file path

**Parameters**

- **file\_path** (*str*) – Location to save file
- **pickle\_protocol** (*int*) – The pickle data stream format.

**Returns** None

```
class evalml.pipelines.components.DelayedFeatureTransformer(date_index=None,
                                                            max_delay=2, de-
                                                            lay_features=True,
                                                            delay_target=True,
                                                            gap=1, ran-
                                                            dom_seed=0,
                                                            **kwargs)
```

Transformer that delays input features and target variable for time series problems.

#### Parameters

- **date\_index** (*str*) – Name of the column containing the datetime information used to order the data. Ignored.
- **max\_delay** (*int*) – Maximum number of time units to delay each feature. Defaults to 2.
- **delay\_features** (*bool*) – Whether to delay the input features. Defaults to True.
- **delay\_target** (*bool*) – Whether to delay the target. Defaults to True.
- **gap** (*int*) – The number of time units between when the features are collected and when the target is collected. For example, if you are predicting the next time step’s target, gap=1. This is only needed because when gap=0, we need to be sure to start the lagging of the target variable at 1. Defaults to 1.
- **random\_seed** (*int*) – Seed for the random number generator. This transformer performs the same regardless of the random seed provided.

#### Attributes

<b>hyper-parameter_ranges</b>	{}
<b>model_family</b>	ModelFamily.NONE
<b>modifies_features</b>	True
<b>modifies_target</b>	False
<b>name</b>	Delayed Feature Transformer
<b>needs_fitting</b>	False

#### Methods

<i>clone</i>	Constructs a new component with the same parameters and random state.
<i>default_parameters</i>	Returns the default parameters for this component.
<i>describe</i>	Describe a component and its parameters
<i>fit</i>	Fits the DelayFeatureTransformer.
<i>fit_transform</i>	Fits on X and transforms X
<i>load</i>	Loads component at file path
<i>parameters</i>	Returns the parameters which were used to initialize the component
<i>save</i>	Saves component at file path
<i>transform</i>	Computes the delayed features for all features in X and y.

**clone** (*self*)



Constructs a new component with the same parameters and random state.

**Returns** A new instance of this component with identical parameters and random state.

**default\_parameters** (*cls*)

Returns the default parameters for this component.

Our convention is that `Component.default_parameters == Component().parameters`.

**Returns** default parameters for this component.

**Return type** dict

**describe** (*self*, *print\_name=False*, *return\_dict=False*)

Describe a component and its parameters

**Parameters**

- **print\_name** (*bool*, *optional*) – whether to print name of component
- **return\_dict** (*bool*, *optional*) – whether to return description as dictionary in the format {"name": name, "parameters": parameters}

**Returns** prints and returns dictionary

**Return type** None or dict

**fit** (*self*, *X*, *y=None*)

Fits the DelayFeatureTransformer.

**Parameters**

- **X** (*pd.DataFrame* or *np.ndarray*) – The input training data of shape [n\_samples, n\_features]
- **y** (*pd.Series*, *optional*) – The target training data of length [n\_samples]

**Returns** self

**fit\_transform** (*self*, *X*, *y*)

Fits on X and transforms X

**Parameters**

- **X** (*pd.DataFrame*) – Data to fit and transform
- **y** (*pd.Series*) – Target data

**Returns** Transformed X

**Return type** pd.DataFrame

**static load** (*file\_path*)

Loads component at file path

**Parameters** **file\_path** (*str*) – Location to load file

**Returns** ComponentBase object

**property parameters** (*self*)

Returns the parameters which were used to initialize the component

**save** (*self*, *file\_path*, *pickle\_protocol=cloudpickle.DEFAULT\_PROTOCOL*)

Saves component at file path

**Parameters**

- **file\_path** (*str*) – Location to save file

- **pickle\_protocol** (*int*) – The pickle data stream format.

**Returns** None

**transform** (*self*, *X*, *y=None*)

Computes the delayed features for all features in *X* and *y*.

For each feature in *X*, it will add a column to the output dataframe for each delay in the (inclusive) range [1, *max\_delay*]. The values of each delayed feature are simply the original feature shifted forward in time by the delay amount. For example, a delay of 3 units means that the feature value at row *n* will be taken from the *n*-3rd row of that feature

If *y* is not None, it will also compute the delayed values for the target variable.

**Parameters**

- **X** (*pd.DataFrame* or *None*) – Data to transform. None is expected when only the target variable is being used.
- **y** (*pd.Series*, or *None*) – Target.

**Returns** Transformed *X*.

**Return type** *pd.DataFrame*

**class** evalml.pipelines.components.**DFSTransformer** (*index='index'*, *random\_seed=0*,  
\*\**kwargs*)

Featuretools DFS component that generates features for the input features.

**Parameters**

- **index** (*string*) – The name of the column that contains the indices. If no column with this name exists, then `featuretools.EntitySet()` creates a column with this name to serve as the index column. Defaults to 'index'.
- **random\_seed** (*int*) – Seed for the random number generator. Defaults to 0.

**Attributes**

<b>hyper-param-eter_ranges</b>	{}
<b>model_family</b>	ModelFamily.NONE
<b>modi-fies_features</b>	True
<b>modi-fies_target</b>	False
<b>name</b>	DFS Transformer

**Methods**

<i>clone</i>	Constructs a new component with the same parameters and random state.
<i>default_parameters</i>	Returns the default parameters for this component.
<i>describe</i>	Describe a component and its parameters
<i>fit</i>	Fits the DFSTransformer Transformer component.
<i>fit_transform</i>	Fits on <i>X</i> and transforms <i>X</i>
<i>load</i>	Loads component at file path

continues on next page

Table 554 – continued from previous page

<i>needs_fitting</i>	Returns boolean determining if component needs fitting before
<i>parameters</i>	Returns the parameters which were used to initialize the component
<i>save</i>	Saves component at file path
<i>transform</i>	Computes the feature matrix for the input X using featuretools’ dfs algorithm.

**clone** (*self*)

Constructs a new component with the same parameters and random state.

**Returns** A new instance of this component with identical parameters and random state.

**default\_parameters** (*cls*)

Returns the default parameters for this component.

Our convention is that `Component.default_parameters == Component().parameters`.

**Returns** default parameters for this component.

**Return type** dict

**describe** (*self*, *print\_name=False*, *return\_dict=False*)

Describe a component and its parameters

**Parameters**

- **print\_name** (*bool*, *optional*) – whether to print name of component
- **return\_dict** (*bool*, *optional*) – whether to return description as dictionary in the format {“name”: name, “parameters”: parameters}

**Returns** prints and returns dictionary

**Return type** None or dict

**fit** (*self*, *X*, *y=None*)

Fits the DFSTransformer Transformer component.

**Parameters**

- **X** (*pd.DataFrame*, *np.array*) – The input data to transform, of shape [n\_samples, n\_features]
- **y** (*pd.Series*, *np.ndarray*, *optional*) – The target training data of length [n\_samples]

**Returns** self

**fit\_transform** (*self*, *X*, *y=None*)

Fits on X and transforms X

**Parameters**

- **X** (*pd.DataFrame*) – Data to fit and transform
- **y** (*pd.Series*) – Target data

**Returns** Transformed X

**Return type** pd.DataFrame

**static load** (*file\_path*)

Loads component at file path

**Parameters** `file_path` (*str*) – Location to load file

**Returns** ComponentBase object

**needs\_fitting**(*self*)

Returns boolean determining if component needs fitting before calling predict, predict\_proba, transform, or feature\_importances. This can be overridden to False for components that do not need to be fit or whose fit methods do nothing.

**property parameters** (*self*)

Returns the parameters which were used to initialize the component

```
save (self, file_path, pickle_protocol=cloudpickle.DEFAULT_PROTOCOL)
```

Saves component at file path

## Parameters

- **file\_path** (*str*) – Location to save file
- **pickle\_protocol** (*int*) – The pickle data stream format.

**Returns** None

```
transform (self, X, y=None)
```

Computes the feature matrix for the input X using featuretools' dfs algorithm.

## Parameters

- **x** (*pd.DataFrame* or *np.ndarray*) – The input training data to transform. Has shape [n\_samples, n\_features]
- **y** (*pd.Series*, optional) – Ignored.

Returns Feature matrix

**Return type** `pd.DataFrame`

```
class evalml.pipelines.components.DropColumns(columns=None, random_seed=0,  
                                              **kwargs)
```

Drops specified columns in input data.

## Parameters

- **columns** (*list (string)*) – List of column names, used to determine which columns to drop.
- **random\_seed** (*int*) – Seed for the random number generator. Defaults to 0.

## Attributes

<b>hyper-parameter_ranges</b>	{}
<b>model_family</b>	ModelFamily.NONE
<b>modifies_features</b>	True
<b>modifies_target</b>	False
<b>name</b>	Drop Columns Transformer
<b>needs_fitting</b>	False

## Methods

<code>clone</code>	Constructs a new component with the same parameters and random state.
<code>default_parameters</code>	Returns the default parameters for this component.
<code>describe</code>	Describe a component and its parameters
<code>fit</code>	Fits the transformer by checking if column names are present in the dataset.
<code>fit_transform</code>	Fits on X and transforms X
<code>load</code>	Loads component at file path
<code>parameters</code>	Returns the parameters which were used to initialize the component
<code>save</code>	Saves component at file path
<code>transform</code>	Transforms data X by dropping columns.

**clone** (*self*)

Constructs a new component with the same parameters and random state.

**Returns** A new instance of this component with identical parameters and random state.

**default\_parameters** (*cls*)

Returns the default parameters for this component.

Our convention is that `Component.default_parameters == Component().parameters`.

**Returns** default parameters for this component.

**Return type** dict

**describe** (*self*, *print\_name=False*, *return\_dict=False*)

Describe a component and its parameters

**Parameters**

- **print\_name** (*bool*, *optional*) – whether to print name of component
- **return\_dict** (*bool*, *optional*) – whether to return description as dictionary in the format {"name": name, "parameters": parameters}

**Returns** prints and returns dictionary

**Return type** None or dict

**fit** (*self*, *X*, *y=None*)

Fits the transformer by checking if column names are present in the dataset.

**Parameters**

- **X** (*pd.DataFrame*) – Data to check.
- **y** (*pd.Series*, *optional*) – Targets.

**Returns** self

**fit\_transform** (*self*, *X*, *y=None*)

Fits on X and transforms X

**Parameters**

- **X** (*pd.DataFrame*) – Data to fit and transform
- **y** (*pd.Series*) – Target data

**Returns** Transformed X

**Return type** `pd.DataFrame`

**static load** (*file\_path*)

Loads component at file path

**Parameters** `file_path` (*str*) – Location to load file

**Returns** `ComponentBase` object

**property parameters** (*self*)

Returns the parameters which were used to initialize the component

**save** (*self*, *file\_path*, *pickle\_protocol*=`cloudpickle.DEFAULT_PROTOCOL`)

Saves component at file path

**Parameters**

- `file_path` (*str*) – Location to save file
- `pickle_protocol` (*int*) – The pickle data stream format.

**Returns** `None`

**transform** (*self*, *X*, *y*=`None`)

Transforms data *X* by dropping columns.

**Parameters**

- `X` (`pd.DataFrame`) – Data to transform.
- `y` (`pd.Series`, *optional*) – Targets.

**Returns** Transformed *X*.

**Return type** `pd.DataFrame`

**class** `evalml.pipelines.components.DropNullColumns` (*pct\_null\_threshold*=1.0, *random\_seed*=0, *\*\*kwargs*) *ran-*

Transformer to drop features whose percentage of NaN values exceeds a specified threshold.

**Parameters**

- `pct_null_threshold` (*float*) – The percentage of NaN values in an input feature to drop. Must be a value between [0, 1] inclusive. If equal to 0.0, will drop columns with any null values. If equal to 1.0, will drop columns with all null values. Defaults to 0.95.
- `random_seed` (*int*) – Seed for the random number generator. Defaults to 0.

**Attributes**

<b>hyper-parameter_ranges</b>	{}
<b>model_family</b>	<code>ModelFamily.NONE</code>
<b>modifies_features</b>	<code>True</code>
<b>modifies_target</b>	<code>False</code>
<b>name</b>	Drop Null Columns Transformer

**Methods**

<code>clone</code>	Constructs a new component with the same parameters and random state.
<code>default_parameters</code>	Returns the default parameters for this component.
<code>describe</code>	Describe a component and its parameters
<code>fit</code>	Fits component to data
<code>fit_transform</code>	Fits on X and transforms X
<code>load</code>	Loads component at file path
<code>needs_fitting</code>	Returns boolean determining if component needs fitting before
<code>parameters</code>	Returns the parameters which were used to initialize the component
<code>save</code>	Saves component at file path
<code>transform</code>	Transforms data X by dropping columns that exceed the threshold of null values.

**clone** (*self*)

Constructs a new component with the same parameters and random state.

**Returns** A new instance of this component with identical parameters and random state.

**default\_parameters** (*cls*)

Returns the default parameters for this component.

Our convention is that `Component.default_parameters == Component().parameters`.

**Returns** default parameters for this component.

**Return type** dict

**describe** (*self*, *print\_name=False*, *return\_dict=False*)

Describe a component and its parameters

**Parameters**

- **print\_name** (*bool*, *optional*) – whether to print name of component
- **return\_dict** (*bool*, *optional*) – whether to return description as dictionary in the format {"name": name, "parameters": parameters}

**Returns** prints and returns dictionary

**Return type** None or dict

**fit** (*self*, *X*, *y=None*)

Fits component to data

**Parameters**

- **X** (*list*, *pd.DataFrame* or *np.ndarray*) – The input training data of shape [n\_samples, n\_features]
- **y** (*list*, *pd.Series*, *np.ndarray*, *optional*) – The target training data of length [n\_samples]

**Returns** self

**fit\_transform** (*self*, *X*, *y=None*)

Fits on X and transforms X

**Parameters**

- **X** (*pd.DataFrame*) – Data to fit and transform

- **y** (*pd.Series*) – Target data

**Returns** Transformed X

**Return type** *pd.DataFrame*

**static load** (*file\_path*)

Loads component at file path

**Parameters** **file\_path** (*str*) – Location to load file

**Returns** *ComponentBase* object

**needs\_fitting** (*self*)

Returns boolean determining if component needs fitting before calling predict, predict\_proba, transform, or feature\_importances. This can be overridden to False for components that do not need to be fit or whose fit methods do nothing.

**property parameters** (*self*)

Returns the parameters which were used to initialize the component

**save** (*self, file\_path, pickle\_protocol=cloudpickle.DEFAULT\_PROTOCOL*)

Saves component at file path

**Parameters**

- **file\_path** (*str*) – Location to save file
- **pickle\_protocol** (*int*) – The pickle data stream format.

**Returns** None

**transform** (*self, X, y=None*)

Transforms data X by dropping columns that exceed the threshold of null values.

**Parameters**

- **X** (*pd.DataFrame*) – Data to transform
- **y** (*pd.Series, optional*) – Ignored.

**Returns** Transformed X

**Return type** *pd.DataFrame*

```
class evalml.pipelines.components.ElasticNetClassifier (penalty='elasticnet',  
                                                    C=1.0, l1_ratio=0.15,  
                                                    multi_class='auto',  
                                                    solver='saga', n_jobs=- 1,  
                                                    random_seed=0, **kwargs)
```

Elastic Net Classifier. Uses Logistic Regression with elasticnet penalty as the base estimator.

**Parameters**

- **penalty** (*{ "l1", "l2", "elasticnet", "none" }*) – The norm used in penalization. Defaults to “elasticnet”.
- **C** (*float*) – Inverse of regularization strength. Must be a positive float. Defaults to 1.0.
- **l1\_ratio** (*float*) – The mixing parameter, with  $0 \leq \text{l1\_ratio} \leq 1$ . Only used if *penalty='elasticnet'*. Setting *l1\_ratio=0* is equivalent to using *penalty='l2'*, while setting *l1\_ratio=1* is equivalent to using *penalty='l1'*. For  $0 < \text{l1\_ratio} < 1$ , the penalty is a combination of L1 and L2. Defaults to 0.15.



- **multi\_class** (`{"auto", "ovr", "multinomial"}`) – If the option chosen is “ovr”, then a binary problem is fit for each label. For “multinomial” the loss minimised is the multinomial loss fit across the entire probability distribution, even when the data is binary. “multinomial” is unavailable when solver=“liblinear”. “auto” selects “ovr” if the data is binary, or if solver=“liblinear”, and otherwise selects “multinomial”. Defaults to “auto”.
- **solver** (`{"newton-cg", "lbfgs", "liblinear", "sag", "saga"}`) – Algorithm to use in the optimization problem. For small datasets, “liblinear” is a good choice, whereas “sag” and “saga” are faster for large ones. For multiclass problems, only “newton-cg”, “sag”, “saga” and “lbfgs” handle multinomial loss; “liblinear” is limited to one-versus-rest schemes.
  - “newton-cg”, “lbfgs”, “sag” and “saga” handle L2 or no penalty
  - “liblinear” and “saga” also handle L1 penalty
  - “saga” also supports “elasticnet” penalty
  - “liblinear” does not support setting penalty=’none’
 Defaults to “saga”.
- **n\_jobs** (*int*) – Number of parallel threads used to run xgboost. Note that creating thread contention will significantly slow down the algorithm. Defaults to -1.
- **random\_seed** (*int*) – Seed for the random number generator. Defaults to 0.

#### Attributes

<b>hyper-parameter_ranges</b>	{ “C”: Real(0.01, 10), “l1_ratio”: Real(0, 1)}
<b>model_family</b>	ModelFamily.LINEAR_MODEL
<b>modifies_features</b>	True
<b>modifies_target</b>	False
<b>name</b>	Elastic Net Classifier
<b>predict Uses y</b>	False
<b>supported_problem_types</b>	[ ProblemTypes.BINARY, ProblemTypes.MULTICLASS, ProblemTypes.TIME_SERIES_BINARY, ProblemTypes.TIME_SERIES_MULTICLASS,]

#### Methods

<i>clone</i>	Constructs a new component with the same parameters and random state.
<i>default_parameters</i>	Returns the default parameters for this component.
<i>describe</i>	Describe a component and its parameters
<i>feature_importance</i>	Returns importance associated with each feature.
<i>fit</i>	Fits component to data
<i>load</i>	Loads component at file path
<i>needs_fitting</i>	Returns boolean determining if component needs fitting before
<i>parameters</i>	Returns the parameters which were used to initialize the component

continues on next page

Table 557 – continued from previous page

<code>predict</code>	Make predictions using selected features.
<code>predict_proba</code>	Make probability estimates for labels.
<code>save</code>	Saves component at file path

**clone** (*self*)

Constructs a new component with the same parameters and random state.

**Returns** A new instance of this component with identical parameters and random state.

**default\_parameters** (*cls*)

Returns the default parameters for this component.

Our convention is that `Component.default_parameters == Component().parameters`.

**Returns** default parameters for this component.

**Return type** dict

**describe** (*self*, *print\_name=False*, *return\_dict=False*)

Describe a component and its parameters

**Parameters**

- **print\_name** (*bool*, *optional*) – whether to print name of component
- **return\_dict** (*bool*, *optional*) – whether to return description as dictionary in the format {"name": name, "parameters": parameters}

**Returns** prints and returns dictionary

**Return type** None or dict

**property feature\_importance** (*self*)

Returns importance associated with each feature.

**Returns** Importance associated with each feature

**Return type** np.ndarray

**fit** (*self*, *X*, *y*)

Fits component to data

**Parameters**

- **X** (*list*, *pd.DataFrame* or *np.ndarray*) – The input training data of shape [n\_samples, n\_features]
- **y** (*list*, *pd.Series*, *np.ndarray*, *optional*) – The target training data of length [n\_samples]

**Returns** self

**static load** (*file\_path*)

Loads component at file path

**Parameters** **file\_path** (*str*) – Location to load file

**Returns** ComponentBase object

**needs\_fitting** (*self*)

Returns boolean determining if component needs fitting before calling predict, predict\_proba, transform, or feature\_importances. This can be overridden to False for components that do not need to be fit or whose fit methods do nothing.

**property parameters** (*self*)

Returns the parameters which were used to initialize the component

**predict** (*self*, *X*)

Make predictions using selected features.

**Parameters** *X* (*pd.DataFrame*, *np.ndarray*) – Data of shape [n\_samples, n\_features]

**Returns** Predicted values

**Return type** *pd.Series*

**predict\_proba** (*self*, *X*)

Make probability estimates for labels.

**Parameters** *X* (*pd.DataFrame*, *or np.ndarray*) – Features

**Returns** Probability estimates

**Return type** *pd.Series*

**save** (*self*, *file\_path*, *pickle\_protocol=cloudpickle.DEFAULT\_PROTOCOL*)

Saves component at file path

**Parameters**

- **file\_path** (*str*) – Location to save file
- **pickle\_protocol** (*int*) – The pickle data stream format.

**Returns** None

**class** evalml.pipelines.components.**ElasticNetRegressor** (*alpha=0.0001*, *l1\_ratio=0.15*,  
*max\_iter=1000*, *normalize=False*, *random\_seed=0*,  
*\*\*kwargs*)

Elastic Net Regressor.

**Parameters**

- **alpha** (*float*) – Constant that multiplies the penalty terms. Defaults to 0.0001.
- **l1\_ratio** (*float*) – The mixing parameter, with  $0 \leq \text{l1\_ratio} \leq 1$ . Only used if `penalty='elasticnet'`. Setting `l1_ratio=0` is equivalent to using `penalty='l2'`, while setting `l1_ratio=1` is equivalent to using `penalty='l1'`. For  $0 < \text{l1\_ratio} < 1$ , the penalty is a combination of L1 and L2. Defaults to 0.15.
- **max\_iter** (*int*) – The maximum number of iterations. Defaults to 1000.
- **normalize** (*boolean*) – If True, the regressors will be normalized before regression by subtracting the mean and dividing by the l2-norm. Defaults to False.
- **random\_seed** (*int*) – Seed for the random number generator. Defaults to 0.

**Attributes**

<b>hyper-parameter_ranges</b>	{ "alpha": Real(0, 1), "l1_ratio": Real(0, 1), }
<b>model_family</b>	ModelFamily.LINEAR_MODEL
<b>modifies_features</b>	True
<b>modifies_target</b>	False
<b>name</b>	Elastic Net Regressor
<b>pre-dict_uses_y</b>	False
<b>supported_problem_types</b>	[ ProblemTypes.REGRESSION, ProblemTypes.TIME_SERIES_REGRESSION, ]

## Methods

<i>clone</i>	Constructs a new component with the same parameters and random state.
<i>default_parameters</i>	Returns the default parameters for this component.
<i>describe</i>	Describe a component and its parameters
<i>feature_importance</i>	Returns importance associated with each feature.
<i>fit</i>	Fits component to data
<i>load</i>	Loads component at file path
<i>needs_fitting</i>	Returns boolean determining if component needs fitting before
<i>parameters</i>	Returns the parameters which were used to initialize the component
<i>predict</i>	Make predictions using selected features.
<i>predict_proba</i>	Make probability estimates for labels.
<i>save</i>	Saves component at file path

### **clone** (*self*)

Constructs a new component with the same parameters and random state.

**Returns** A new instance of this component with identical parameters and random state.

### **default\_parameters** (*cls*)

Returns the default parameters for this component.

Our convention is that `Component.default_parameters == Component().parameters`.

**Returns** default parameters for this component.

**Return type** dict

### **describe** (*self*, *print\_name=False*, *return\_dict=False*)

Describe a component and its parameters

#### Parameters

- **print\_name** (*bool*, *optional*) – whether to print name of component
- **return\_dict** (*bool*, *optional*) – whether to return description as dictionary in the format {"name": name, "parameters": parameters}

**Returns** prints and returns dictionary

**Return type** None or dict

**property feature\_importance** (*self*)

Returns importance associated with each feature.

**Returns** Importance associated with each feature

**Return type** np.ndarray

**fit** (*self*, *X*, *y=None*)

Fits component to data

**Parameters**

- **X** (*list*, *pd.DataFrame* or *np.ndarray*) – The input training data of shape [n\_samples, n\_features]
- **y** (*list*, *pd.Series*, *np.ndarray*, *optional*) – The target training data of length [n\_samples]

**Returns** self

**static load** (*file\_path*)

Loads component at file path

**Parameters** **file\_path** (*str*) – Location to load file

**Returns** ComponentBase object

**needs\_fitting** (*self*)

Returns boolean determining if component needs fitting before calling predict, predict\_proba, transform, or feature\_importances. This can be overridden to False for components that do not need to be fit or whose fit methods do nothing.

**property parameters** (*self*)

Returns the parameters which were used to initialize the component

**predict** (*self*, *X*)

Make predictions using selected features.

**Parameters** **X** (*pd.DataFrame*, *np.ndarray*) – Data of shape [n\_samples, n\_features]

**Returns** Predicted values

**Return type** pd.Series

**predict\_proba** (*self*, *X*)

Make probability estimates for labels.

**Parameters** **X** (*pd.DataFrame*, or *np.ndarray*) – Features

**Returns** Probability estimates

**Return type** pd.Series

**save** (*self*, *file\_path*, *pickle\_protocol=cloudpickle.DEFAULT\_PROTOCOL*)

Saves component at file path

**Parameters**

- **file\_path** (*str*) – Location to save file
- **pickle\_protocol** (*int*) – The pickle data stream format.

**Returns** None

**class** evalml.pipelines.components.**EmailFeaturizer** (*random\_seed=0, \*\*kwargs*)  
Transformer that can automatically extract features from emails.

**Parameters** **random\_seed** (*int*) – Seed for the random number generator. Defaults to 0.

#### Attributes

<b>hyper- parame- ter_ranges</b>	{}
<b>model_family</b>	ModelFamily.NONE
<b>modi- fies_features</b>	True
<b>modi- fies_target</b>	False
<b>name</b>	Email Featurizer

#### Methods

<i>clone</i>	Constructs a new component with the same parameters and random state.
<i>default_parameters</i>	Returns the default parameters for this component.
<i>describe</i>	Describe a component and its parameters
<i>fit</i>	Fits component to data
<i>fit_transform</i>	Fits on X and transforms X
<i>load</i>	Loads component at file path
<i>needs_fitting</i>	Returns boolean determining if component needs fitting before
<i>parameters</i>	Returns the parameters which were used to initialize the component
<i>save</i>	Saves component at file path
<i>transform</i>	Transforms data X.

**clone** (*self*)

Constructs a new component with the same parameters and random state.

**Returns** A new instance of this component with identical parameters and random state.

**default\_parameters** (*cls*)

Returns the default parameters for this component.

Our convention is that `Component.default_parameters == Component().parameters`.

**Returns** default parameters for this component.

**Return type** dict

**describe** (*self, print\_name=False, return\_dict=False*)

Describe a component and its parameters

**Parameters**

- **print\_name** (*bool, optional*) – whether to print name of component
- **return\_dict** (*bool, optional*) – whether to return description as dictionary in the format {"name": name, "parameters": parameters}

**Returns** prints and returns dictionary

**Return type** None or dict

**fit** (*self*, *X*, *y=None*)

Fits component to data

**Parameters**

- **X** (*list*, *pd.DataFrame* or *np.ndarray*) – The input training data of shape [n\_samples, n\_features]
- **y** (*list*, *pd.Series*, *np.ndarray*, *optional*) – The target training data of length [n\_samples]

**Returns** *self*

**fit\_transform** (*self*, *X*, *y=None*)

Fits on X and transforms X

**Parameters**

- **X** (*pd.DataFrame*) – Data to fit and transform
- **y** (*pd.Series*) – Target data

**Returns** Transformed X

**Return type** *pd.DataFrame*

**static load** (*file\_path*)

Loads component at file path

**Parameters** **file\_path** (*str*) – Location to load file

**Returns** ComponentBase object

**needs\_fitting** (*self*)

Returns boolean determining if component needs fitting before calling predict, predict\_proba, transform, or feature\_importances. This can be overridden to False for components that do not need to be fit or whose fit methods do nothing.

**property parameters** (*self*)

Returns the parameters which were used to initialize the component

**save** (*self*, *file\_path*, *pickle\_protocol=cloudpickle.DEFAULT\_PROTOCOL*)

Saves component at file path

**Parameters**

- **file\_path** (*str*) – Location to save file
- **pickle\_protocol** (*int*) – The pickle data stream format.

**Returns** None

**transform** (*self*, *X*, *y=None*)

Transforms data X.

**Parameters**

- **X** (*pd.DataFrame*) – Data to transform.
- **y** (*pd.Series*, *optional*) – Target data.

**Returns** Transformed X

**Return type** *pd.DataFrame*

```
class evalml.pipelines.components.Estimator (parameters=None, component_obj=None,  
                                              random_seed=0, **kwargs)
```

A component that fits and predicts given data.

To implement a new Estimator, define your own class which is a subclass of Estimator, including a name and a list of acceptable ranges for any parameters to be tuned during the automl search (hyperparameters). Define an `__init__` method which sets up any necessary state and objects. Make sure your `__init__` only uses standard keyword arguments and calls `super().__init__()` with a parameters dict. You may also override the `fit`, `transform`, `fit_transform` and other methods in this class if appropriate.

To see some examples, check out the definitions of any Estimator component.

### Parameters

- **parameters** (*dict*) – Dictionary of parameters for the component. Defaults to None.
- **component\_obj** (*obj*) – Third-party objects useful in component implementation. Defaults to None.
- **random\_seed** (*int*) – Seed for the random number generator. Defaults to 0.

### Attributes

<b>model_family</b>	ModelFamily.NONE
<b>modifies_features</b>	True
<b>modifies_target</b>	False
<b>pre-dict_uses_y</b>	False

### Methods

<i>clone</i>	Constructs a new component with the same parameters and random state.
<i>default_parameters</i>	Returns the default parameters for this component.
<i>describe</i>	Describe a component and its parameters
<i>feature_importance</i>	Returns importance associated with each feature.
<i>fit</i>	Fits component to data
<i>load</i>	Loads component at file path
<i>name</i>	Returns string name of this component
<i>needs_fitting</i>	Returns boolean determining if component needs fitting before
<i>parameters</i>	Returns the parameters which were used to initialize the component
<i>predict</i>	Make predictions using selected features.
<i>predict_proba</i>	Make probability estimates for labels.
<i>save</i>	Saves component at file path
<i>supported_problem_types</i>	Problem types this estimator supports

**clone** (*self*)

Constructs a new component with the same parameters and random state.

**Returns** A new instance of this component with identical parameters and random state.

**default\_parameters** (*cls*)

Returns the default parameters for this component.



Our convention is that `Component.default_parameters == Component().parameters`.

**Returns** default parameters for this component.

**Return type** dict

**describe** (*self*, *print\_name=False*, *return\_dict=False*)

Describe a component and its parameters

**Parameters**

- **print\_name** (*bool*, *optional*) – whether to print name of component
- **return\_dict** (*bool*, *optional*) – whether to return description as dictionary in the format {"name": name, "parameters": parameters}

**Returns** prints and returns dictionary

**Return type** None or dict

**property feature\_importance** (*self*)

Returns importance associated with each feature.

**Returns** Importance associated with each feature

**Return type** np.ndarray

**fit** (*self*, *X*, *y=None*)

Fits component to data

**Parameters**

- **X** (*list*, *pd.DataFrame* or *np.ndarray*) – The input training data of shape [n\_samples, n\_features]
- **y** (*list*, *pd.Series*, *np.ndarray*, *optional*) – The target training data of length [n\_samples]

**Returns** self

**static load** (*file\_path*)

Loads component at file path

**Parameters** **file\_path** (*str*) – Location to load file

**Returns** ComponentBase object

**property name** (*cls*)

Returns string name of this component

**needs\_fitting** (*self*)

Returns boolean determining if component needs fitting before calling predict, predict\_proba, transform, or feature\_importances. This can be overridden to False for components that do not need to be fit or whose fit methods do nothing.

**property parameters** (*self*)

Returns the parameters which were used to initialize the component

**predict** (*self*, *X*)

Make predictions using selected features.

**Parameters** **X** (*pd.DataFrame*, *np.ndarray*) – Data of shape [n\_samples, n\_features]

**Returns** Predicted values

**Return type** pd.Series

**predict\_proba** (*self*, *X*)

Make probability estimates for labels.

**Parameters** *X* (*pd.DataFrame*, or *np.ndarray*) – Features

**Returns** Probability estimates

**Return type** *pd.Series*

**save** (*self*, *file\_path*, *pickle\_protocol=cloudpickle.DEFAULT\_PROTOCOL*)

Saves component at file path

**Parameters**

- **file\_path** (*str*) – Location to save file
- **pickle\_protocol** (*int*) – The pickle data stream format.

**Returns** None

**property supported\_problem\_types** (*cls*)

Problem types this estimator supports

```
class evalml.pipelines.components.ExtraTreesClassifier (n_estimators=100,  
                                                    max_features='auto',  
                                                    max_depth=6,  
                                                    min_samples_split=2,  
                                                    min_weight_fraction_leaf=0.0,  
                                                    n_jobs=-1, ran-  
                                                    dom_seed=0, **kwargs)
```

Extra Trees Classifier.

**Parameters**

- **n\_estimators** (*float*) – The number of trees in the forest. Defaults to 100.
- **max\_features** (*int*, *float* or {"auto", "sqrt", "log2"}) – The number of features to consider when looking for the best split:
  - If *int*, then consider *max\_features* features at each split.
  - If *float*, then *max\_features* is a fraction and *int(max\_features \* n\_features)* features are considered at each split.
  - If “auto”, then *max\_features=sqrt(n\_features)*.
  - If “sqrt”, then *max\_features=sqrt(n\_features)*.
  - If “log2”, then *max\_features=log2(n\_features)*.
  - If *None*, then *max\_features = n\_features*.

The search for a split does not stop until at least one valid partition of the node samples is found, even if it requires to effectively inspect more than *max\_features* features. Defaults to “auto”.

- **max\_depth** (*int*) – The maximum depth of the tree. Defaults to 6.
- **min\_samples\_split** (*int* or *float*) – The minimum number of samples required to split an internal node:
  - If *int*, then consider *min\_samples\_split* as the minimum number.
  - If *float*, then *min\_samples\_split* is a fraction and *ceil(min\_samples\_split \* n\_samples)* are the minimum number of samples for each split.
- **to 2.** (*Defaults*) –

- **min\_weight\_fraction\_leaf** (*float*) – The minimum weighted fraction of the sum total of weights (of all the input samples) required to be at a leaf node. Defaults to 0.0.
- **n\_jobs** (*int or None*) – Number of jobs to run in parallel. -1 uses all processes. Defaults to -1.
- **random\_seed** (*int*) – Seed for the random number generator. Defaults to 0.

#### Attributes

<b>hyper-parameter_ranges</b>	{ “n_estimators”: Integer(10, 1000), “max_features”: [“auto”, “sqrt”, “log2”], “max_depth”: Integer(4, 10),}
<b>model_family</b>	ModelFamily.EXTRA_TREES
<b>modifies_features</b>	True
<b>modifies_target</b>	False
<b>name</b>	Extra Trees Classifier
<b>predict_uses_y</b>	False
<b>supported_problem_types</b>	[ ProblemTypes.BINARY, ProblemTypes.MULTICLASS, ProblemTypes.TIME_SERIES_BINARY, ProblemTypes.TIME_SERIES_MULTICLASS,]

#### Methods

<i>clone</i>	Constructs a new component with the same parameters and random state.
<i>default_parameters</i>	Returns the default parameters for this component.
<i>describe</i>	Describe a component and its parameters
<i>feature_importance</i>	Returns importance associated with each feature.
<i>fit</i>	Fits component to data
<i>load</i>	Loads component at file path
<i>needs_fitting</i>	Returns boolean determining if component needs fitting before
<i>parameters</i>	Returns the parameters which were used to initialize the component
<i>predict</i>	Make predictions using selected features.
<i>predict_proba</i>	Make probability estimates for labels.
<i>save</i>	Saves component at file path

#### **clone** (*self*)

Constructs a new component with the same parameters and random state.

**Returns** A new instance of this component with identical parameters and random state.

#### **default\_parameters** (*cls*)

Returns the default parameters for this component.

Our convention is that `Component.default_parameters == Component().parameters`.

**Returns** default parameters for this component.

**Return type** dict

#### **describe** (*self*, *print\_name=False*, *return\_dict=False*)

Describe a component and its parameters

**Parameters**

- **print\_name** (*bool, optional*) – whether to print name of component
- **return\_dict** (*bool, optional*) – whether to return description as dictionary in the format {"name": name, "parameters": parameters}

**Returns** prints and returns dictionary

**Return type** None or dict

**property feature\_importance** (*self*)

Returns importance associated with each feature.

**Returns** Importance associated with each feature

**Return type** np.ndarray

**fit** (*self, X, y=None*)

Fits component to data

**Parameters**

- **X** (*list, pd.DataFrame or np.ndarray*) – The input training data of shape [n\_samples, n\_features]
- **y** (*list, pd.Series, np.ndarray, optional*) – The target training data of length [n\_samples]

**Returns** self

**static load** (*file\_path*)

Loads component at file path

**Parameters** **file\_path** (*str*) – Location to load file

**Returns** ComponentBase object

**needs\_fitting** (*self*)

Returns boolean determining if component needs fitting before calling predict, predict\_proba, transform, or feature\_importances. This can be overridden to False for components that do not need to be fit or whose fit methods do nothing.

**property parameters** (*self*)

Returns the parameters which were used to initialize the component

**predict** (*self, X*)

Make predictions using selected features.

**Parameters** **X** (*pd.DataFrame, np.ndarray*) – Data of shape [n\_samples, n\_features]

**Returns** Predicted values

**Return type** pd.Series

**predict\_proba** (*self, X*)

Make probability estimates for labels.

**Parameters** **X** (*pd.DataFrame, or np.ndarray*) – Features

**Returns** Probability estimates

**Return type** pd.Series

**save** (*self, file\_path, pickle\_protocol=cloudpickle.DEFAULT\_PROTOCOL*)

Saves component at file path

**Parameters**

- **file\_path** (*str*) – Location to save file
- **pickle\_protocol** (*int*) – The pickle data stream format.

**Returns** None

```
class evalml.pipelines.components.ExtraTreesRegressor (n_estimators=100,
                                                    max_features='auto',
                                                    max_depth=6,
                                                    min_samples_split=2,
                                                    min_weight_fraction_leaf=0.0,
                                                    n_jobs=-1, random_seed=0,
                                                    **kwargs)
```

Extra Trees Regressor.

**Parameters**

- **n\_estimators** (*float*) – The number of trees in the forest. Defaults to 100.
- **max\_features** (*int, float or {"auto", "sqrt", "log2"}*) – The number of features to consider when looking for the best split:
  - If *int*, then consider *max\_features* features at each split.
  - If *float*, then *max\_features* is a fraction and  $\text{int}(\text{max\_features} * \text{n\_features})$  features are considered at each split.
  - If “auto”, then  $\text{max\_features} = \sqrt{\text{n\_features}}$ .
  - If “sqrt”, then  $\text{max\_features} = \sqrt{\text{n\_features}}$ .
  - If “log2”, then  $\text{max\_features} = \log_2(\text{n\_features})$ .
  - If *None*, then  $\text{max\_features} = \text{n\_features}$ .

The search for a split does not stop until at least one valid partition of the node samples is found, even if it requires to effectively inspect more than *max\_features* features. Defaults to “auto”.

- **max\_depth** (*int*) – The maximum depth of the tree. Defaults to 6.
- **min\_samples\_split** (*int or float*) – The minimum number of samples required to split an internal node:
  - If *int*, then consider *min\_samples\_split* as the minimum number.
  - If *float*, then *min\_samples\_split* is a fraction and  $\text{ceil}(\text{min\_samples\_split} * \text{n\_samples})$  are the minimum number of samples for each split.
- **to 2.** (*Defaults*) –
- **min\_weight\_fraction\_leaf** (*float*) – The minimum weighted fraction of the sum total of weights (of all the input samples) required to be at a leaf node. Defaults to 0.0.
- **n\_jobs** (*int or None*) – Number of jobs to run in parallel. -1 uses all processes. Defaults to -1.
- **random\_seed** (*int*) – Seed for the random number generator. Defaults to 0.

**Attributes**

<b>hyper-parameter_ranges</b>	{ “n_estimators”: Integer(10, 1000), “max_features”: [“auto”, “sqrt”, “log2”], “max_depth”: Integer(4, 10),}
<b>model_family</b>	ModelFamily.EXTRA_TREES
<b>modifies_features</b>	True
<b>modifies_target</b>	False
<b>name</b>	Extra Trees Regressor
<b>pre-dict_uses_y</b>	False
<b>supported_problem_types</b>	[ ProblemTypes.REGRESSION, ProblemTypes.TIME_SERIES_REGRESSION,]

## Methods

<i>clone</i>	Constructs a new component with the same parameters and random state.
<i>default_parameters</i>	Returns the default parameters for this component.
<i>describe</i>	Describe a component and its parameters
<i>feature_importance</i>	Returns importance associated with each feature.
<i>fit</i>	Fits component to data
<i>load</i>	Loads component at file path
<i>needs_fitting</i>	Returns boolean determining if component needs fitting before
<i>parameters</i>	Returns the parameters which were used to initialize the component
<i>predict</i>	Make predictions using selected features.
<i>predict_proba</i>	Make probability estimates for labels.
<i>save</i>	Saves component at file path

### **clone** (*self*)

Constructs a new component with the same parameters and random state.

**Returns** A new instance of this component with identical parameters and random state.

### **default\_parameters** (*cls*)

Returns the default parameters for this component.

Our convention is that `Component.default_parameters == Component().parameters`.

**Returns** default parameters for this component.

**Return type** dict

### **describe** (*self*, *print\_name=False*, *return\_dict=False*)

Describe a component and its parameters

#### Parameters

- **print\_name** (*bool*, *optional*) – whether to print name of component
- **return\_dict** (*bool*, *optional*) – whether to return description as dictionary in the format {“name”: name, “parameters”: parameters}

**Returns** prints and returns dictionary

**Return type** None or dict

**property feature\_importance** (*self*)

Returns importance associated with each feature.

**Returns** Importance associated with each feature

**Return type** np.ndarray

**fit** (*self*, *X*, *y=None*)

Fits component to data

**Parameters**

- **X** (*list*, *pd.DataFrame* or *np.ndarray*) – The input training data of shape [n\_samples, n\_features]
- **y** (*list*, *pd.Series*, *np.ndarray*, *optional*) – The target training data of length [n\_samples]

**Returns** self

**static load** (*file\_path*)

Loads component at file path

**Parameters** **file\_path** (*str*) – Location to load file

**Returns** ComponentBase object

**needs\_fitting** (*self*)

Returns boolean determining if component needs fitting before calling predict, predict\_proba, transform, or feature\_importances. This can be overridden to False for components that do not need to be fit or whose fit methods do nothing.

**property parameters** (*self*)

Returns the parameters which were used to initialize the component

**predict** (*self*, *X*)

Make predictions using selected features.

**Parameters** **X** (*pd.DataFrame*, *np.ndarray*) – Data of shape [n\_samples, n\_features]

**Returns** Predicted values

**Return type** pd.Series

**predict\_proba** (*self*, *X*)

Make probability estimates for labels.

**Parameters** **X** (*pd.DataFrame*, or *np.ndarray*) – Features

**Returns** Probability estimates

**Return type** pd.Series

**save** (*self*, *file\_path*, *pickle\_protocol=cloudpickle.DEFAULT\_PROTOCOL*)

Saves component at file path

**Parameters**

- **file\_path** (*str*) – Location to save file
- **pickle\_protocol** (*int*) – The pickle data stream format.

**Returns** None

```
class evalml.pipelines.components.FeatureSelector (parameters=None, component_obj=None, random_seed=0, **kwargs)
```

Selects top features based on importance weights.

#### Parameters

- **parameters** (*dict*) – Dictionary of parameters for the component. Defaults to None.
- **component\_obj** (*obj*) – Third-party objects useful in component implementation. Defaults to None.
- **random\_seed** (*int*) – Seed for the random number generator. Defaults to 0.

#### Attributes

<b>model_family</b>	ModelFamily.NONE
<b>modifies_features</b>	True
<b>modifies_target</b>	False

#### Methods

<i>clone</i>	Constructs a new component with the same parameters and random state.
<i>default_parameters</i>	Returns the default parameters for this component.
<i>describe</i>	Describe a component and its parameters
<i>fit</i>	Fits component to data
<i>fit_transform</i>	Fits on X and transforms X
<i>get_names</i>	Get names of selected features.
<i>load</i>	Loads component at file path
<i>name</i>	Returns string name of this component
<i>needs_fitting</i>	Returns boolean determining if component needs fitting before
<i>parameters</i>	Returns the parameters which were used to initialize the component
<i>save</i>	Saves component at file path
<i>transform</i>	Transforms input data by selecting features. If the component_obj does not have a transform method, will raise an MethodPropertyNotFoundError exception.

#### **clone** (*self*)

Constructs a new component with the same parameters and random state.

**Returns** A new instance of this component with identical parameters and random state.

#### **default\_parameters** (*cls*)

Returns the default parameters for this component.

Our convention is that `Component.default_parameters == Component().parameters`.

**Returns** default parameters for this component.

**Return type** dict

#### **describe** (*self*, *print\_name=False*, *return\_dict=False*)



Describe a component and its parameters

#### Parameters

- **print\_name** (*bool, optional*) – whether to print name of component
- **return\_dict** (*bool, optional*) – whether to return description as dictionary in the format {"name": name, "parameters": parameters}

**Returns** prints and returns dictionary

**Return type** None or dict

**fit** (*self, X, y=None*)

Fits component to data

#### Parameters

- **X** (*list, pd.DataFrame or np.ndarray*) – The input training data of shape [n\_samples, n\_features]
- **y** (*list, pd.Series, np.ndarray, optional*) – The target training data of length [n\_samples]

**Returns** self

**fit\_transform** (*self, X, y=None*)

Fits on X and transforms X

#### Parameters

- **X** (*pd.DataFrame*) – Data to fit and transform
- **y** (*pd.Series*) – Target data

**Returns** Transformed X

**Return type** pd.DataFrame

**get\_names** (*self*)

Get names of selected features.

**Returns** List of the names of features selected

**Return type** list[str]

**static load** (*file\_path*)

Loads component at file path

**Parameters** **file\_path** (*str*) – Location to load file

**Returns** ComponentBase object

**property name** (*cls*)

Returns string name of this component

**needs\_fitting** (*self*)

Returns boolean determining if component needs fitting before calling predict, predict\_proba, transform, or feature\_importances. This can be overridden to False for components that do not need to be fit or whose fit methods do nothing.

**property parameters** (*self*)

Returns the parameters which were used to initialize the component

**save** (*self, file\_path, pickle\_protocol=cloudpickle.DEFAULT\_PROTOCOL*)

Saves component at file path

**Parameters**

- **file\_path** (*str*) – Location to save file
- **pickle\_protocol** (*int*) – The pickle data stream format.

**Returns** None

**transform** (*self*, *X*, *y=None*)

Transforms input data by selecting features. If the component\_obj does not have a transform method, will raise an `MethodPropertyNotFoundError` exception.

**Parameters**

- **X** (*pd.DataFrame*) – Data to transform.
- **y** (*pd.Series*, *optional*) – Target data. Ignored.

**Returns** Transformed X

**Return type** `pd.DataFrame`

```
class evalml.pipelines.components.Imputer (categorical_impute_strategy='most_frequent',  
                                             categorical_fill_value=None,           nu-  
                                             numeric_impute_strategy='mean',         nu-  
                                             numeric_fill_value=None,       random_seed=0,  
                                             **kwargs)
```

Imputes missing data according to a specified imputation strategy.

**Parameters**

- **categorical\_impute\_strategy** (*string*) – Impute strategy to use for string, object, boolean, categorical dtypes. Valid values include “most\_frequent” and “constant”.
- **numeric\_impute\_strategy** (*string*) – Impute strategy to use for numeric columns. Valid values include “mean”, “median”, “most\_frequent”, and “constant”.
- **categorical\_fill\_value** (*string*) – When `categorical_impute_strategy == “constant”`, `fill_value` is used to replace missing data. The default value of `None` will fill with the string “missing\_value”.
- **numeric\_fill\_value** (*int*, *float*) – When `numeric_impute_strategy == “constant”`, `fill_value` is used to replace missing data. The default value of `None` will fill with 0.
- **random\_seed** (*int*) – Seed for the random number generator. Defaults to 0.

**Attributes**

<b>hyper-parameter_ranges</b>	{ “categorical_impute_strategy”: [“most_frequent”], “numeric_impute_strategy”: [“mean”, “median”, “most_frequent”], }
<b>model_family</b>	ModelFamily.NONE
<b>modifies_features</b>	True
<b>modifies_target</b>	False
<b>name</b>	Imputer

**Methods**

<code>clone</code>	Constructs a new component with the same parameters and random state.
<code>default_parameters</code>	Returns the default parameters for this component.
<code>describe</code>	Describe a component and its parameters
<code>fit</code>	Fits imputer to data. ‘None’ values are converted to np.nan before imputation and are
<code>fit_transform</code>	Fits on X and transforms X
<code>load</code>	Loads component at file path
<code>needs_fitting</code>	Returns boolean determining if component needs fitting before
<code>parameters</code>	Returns the parameters which were used to initialize the component
<code>save</code>	Saves component at file path
<code>transform</code>	Transforms data X by imputing missing values. ‘None’ values are converted to np.nan before imputation and are

**clone** (*self*)

Constructs a new component with the same parameters and random state.

**Returns** A new instance of this component with identical parameters and random state.

**default\_parameters** (*cls*)

Returns the default parameters for this component.

Our convention is that `Component.default_parameters == Component().parameters`.

**Returns** default parameters for this component.

**Return type** dict

**describe** (*self*, *print\_name=False*, *return\_dict=False*)

Describe a component and its parameters

**Parameters**

- **print\_name** (*bool*, *optional*) – whether to print name of component
- **return\_dict** (*bool*, *optional*) – whether to return description as dictionary in the format {"name": name, "parameters": parameters}

**Returns** prints and returns dictionary

**Return type** None or dict

**fit** (*self*, *X*, *y=None*)

**Fits imputer to data. ‘None’ values are converted to np.nan before imputation and are** treated as the same.

**Parameters**

- **X** (*pd.DataFrame*, *np.ndarray*) – The input training data of shape [n\_samples, n\_features]
- **y** (*pd.Series*, *optional*) – The target training data of length [n\_samples]

**Returns** self

**fit\_transform** (*self*, *X*, *y=None*)

Fits on X and transforms X

**Parameters**

- **X** (*pd.DataFrame*) – Data to fit and transform
- **y** (*pd.Series*) – Target data

**Returns** Transformed X**Return type** *pd.DataFrame***static load** (*file\_path*)

Loads component at file path

**Parameters** **file\_path** (*str*) – Location to load file**Returns** *ComponentBase* object**needs\_fitting** (*self*)

Returns boolean determining if component needs fitting before calling `predict`, `predict_proba`, `transform`, or `feature_importances`. This can be overridden to `False` for components that do not need to be fit or whose fit methods do nothing.

**property parameters** (*self*)

Returns the parameters which were used to initialize the component

**save** (*self*, *file\_path*, *pickle\_protocol=cloudpickle.DEFAULT\_PROTOCOL*)

Saves component at file path

**Parameters**

- **file\_path** (*str*) – Location to save file
- **pickle\_protocol** (*int*) – The pickle data stream format.

**Returns** `None`**transform** (*self*, *X*, *y=None*)

Transforms data **X** by imputing missing values. ‘None’ values are converted to `np.nan` before imputation and are treated as the same.

**Parameters**

- **X** (*pd.DataFrame*) – Data to transform
- **y** (*pd.Series*, *optional*) – Ignored.

**Returns** Transformed X**Return type** *pd.DataFrame*

```
class evalml.pipelines.components.KNeighborsClassifier (n_neighbors=5,  
                                                    weights='uniform', algo-  
                                                    rithm='auto', leaf_size=30,  
                                                    p=2, random_seed=0,  
                                                    **kwargs)
```

K-Nearest Neighbors Classifier.

**Parameters**

- **n\_neighbors** (*int*) – Number of neighbors to use by default. Defaults to 5.
- **weights** (*{ 'uniform', 'distance' } or callable*) – Weight function used in prediction. Can be:
  - ‘uniform’ : uniform weights. All points in each neighborhood are weighted equally.

- ‘distance’ : weight points by the inverse of their distance. in this case, closer neighbors of a query point will have a greater influence than neighbors which are further away.
- [callable] : a user-defined function which accepts an array of distances, and returns an array of the same shape containing the weights.

Defaults to “uniform”.

- **algorithm** ({‘auto’, ‘ball\_tree’, ‘kd\_tree’, ‘brute’}) – Algorithm used to compute the nearest neighbors:
  - ‘ball\_tree’ will use BallTree
  - ‘kd\_tree’ will use KDTree
  - ‘brute’ will use a brute-force search.

‘auto’ will attempt to decide the most appropriate algorithm based on the values passed to fit method. Defaults to “auto”. Note: fitting on sparse input will override the setting of this parameter, using brute force.
- **leaf\_size** (*int*) – Leaf size passed to BallTree or KDTree. This can affect the speed of the construction and query, as well as the memory required to store the tree. The optimal value depends on the nature of the problem. Defaults to 30.
- **p** (*int*) – Power parameter for the Minkowski metric. When  $p = 1$ , this is equivalent to using manhattan\_distance (l1), and euclidean\_distance (l2) for  $p = 2$ . For arbitrary  $p$ , minkowski\_distance (l\_p) is used. Defaults to 2.
- **random\_seed** (*int*) – Seed for the random number generator. Defaults to 0.

#### Attributes

<b>hyper-parameter_ranges</b>	{ “n_neighbors”: Integer(2, 12), “weights”: [“uniform”, “distance”], “algorithm”: [“auto”, “ball_tree”, “kd_tree”, “brute”], “leaf_size”: Integer(10, 30), “p”: Integer(1, 5),}
<b>model_family</b>	ModelFamily.K_NEIGHBORS
<b>modifies_features</b>	True
<b>modifies_target</b>	False
<b>name</b>	KNN Classifier
<b>predict_uses_y</b>	False
<b>supported_problem_types</b>	[ ProblemTypes.BINARY, ProblemTypes.MULTICLASS, ProblemTypes.TIME_SERIES_BINARY, ProblemTypes.TIME_SERIES_MULTICLASS,]

#### Methods

<i>clone</i>	Constructs a new component with the same parameters and random state.
<i>default_parameters</i>	Returns the default parameters for this component.
<i>describe</i>	Describe a component and its parameters
<i>feature_importance</i>	Returns array of 0’s matching the input number of features as feature_importance is
<i>fit</i>	Fits component to data
<i>load</i>	Loads component at file path

continues on next page

Table 565 – continued from previous page

<code>needs_fitting</code>	Returns boolean determining if component needs fitting before
<code>parameters</code>	Returns the parameters which were used to initialize the component
<code>predict</code>	Make predictions using selected features.
<code>predict_proba</code>	Make probability estimates for labels.
<code>save</code>	Saves component at file path

**clone** (*self*)

Constructs a new component with the same parameters and random state.

**Returns** A new instance of this component with identical parameters and random state.

**default\_parameters** (*cls*)

Returns the default parameters for this component.

Our convention is that `Component.default_parameters == Component().parameters`.

**Returns** default parameters for this component.

**Return type** dict

**describe** (*self*, *print\_name=False*, *return\_dict=False*)

Describe a component and its parameters

**Parameters**

- **print\_name** (*bool*, *optional*) – whether to print name of component
- **return\_dict** (*bool*, *optional*) – whether to return description as dictionary in the format {"name": name, "parameters": parameters}

**Returns** prints and returns dictionary

**Return type** None or dict

**property feature\_importance** (*self*)

Returns array of 0's matching the input number of features as `feature_importance` is not defined for KNN classifiers.

**fit** (*self*, *X*, *y=None*)

Fits component to data

**Parameters**

- **X** (*list*, *pd.DataFrame* or *np.ndarray*) – The input training data of shape [n\_samples, n\_features]
- **y** (*list*, *pd.Series*, *np.ndarray*, *optional*) – The target training data of length [n\_samples]

**Returns** self

**static load** (*file\_path*)

Loads component at file path

**Parameters** **file\_path** (*str*) – Location to load file

**Returns** ComponentBase object

**needs\_fitting** (*self*)

Returns boolean determining if component needs fitting before calling `predict`, `predict_proba`, `transform`,

or `feature_importances`. This can be overridden to `False` for components that do not need to be fit or whose fit methods do nothing.

**property** `parameters` (*self*)

Returns the parameters which were used to initialize the component

**predict** (*self*, *X*)

Make predictions using selected features.

**Parameters** *X* (*pd.DataFrame*, *np.ndarray*) – Data of shape [n\_samples, n\_features]

**Returns** Predicted values

**Return type** *pd.Series*

**predict\_proba** (*self*, *X*)

Make probability estimates for labels.

**Parameters** *X* (*pd.DataFrame*, or *np.ndarray*) – Features

**Returns** Probability estimates

**Return type** *pd.Series*

**save** (*self*, *file\_path*, *pickle\_protocol=cloudpickle.DEFAULT\_PROTOCOL*)

Saves component at file path

**Parameters**

- **file\_path** (*str*) – Location to save file
- **pickle\_protocol** (*int*) – The pickle data stream format.

**Returns** `None`

```
class evalml.pipelines.components.LightGBMClassifier (boosting_type='gbdt',
                                                    learning_rate=0.1,
                                                    n_estimators=100,
                                                    max_depth=0, num_leaves=31,
                                                    min_child_samples=20, bag-
                                                    ging_fraction=0.9, bag-
                                                    ging_freq=0, n_jobs=-1,
                                                    random_seed=0, **kwargs)
```

LightGBM Classifier.

**Parameters**

- **boosting\_type** (*string*) – Type of boosting to use. Defaults to “gbdt”. - ‘gbdt’ uses traditional Gradient Boosting Decision Tree - “dart”, uses Dropouts meet Multiple Additive Regression Trees - “goss”, uses Gradient-based One-Side Sampling - “rf”, uses Random Forest
- **learning\_rate** (*float*) – Boosting learning rate. Defaults to 0.1.
- **n\_estimators** (*int*) – Number of boosted trees to fit. Defaults to 100.
- **max\_depth** (*int*) – Maximum tree depth for base learners, <=0 means no limit. Defaults to 0.
- **num\_leaves** (*int*) – Maximum tree leaves for base learners. Defaults to 31.
- **min\_child\_samples** (*int*) – Minimum number of data needed in a child (leaf). Defaults to 20.

- **bagging\_fraction** (*float*) – LightGBM will randomly select a subset of features on each iteration (tree) without resampling if this is smaller than 1.0. For example, if set to 0.8, LightGBM will select 80% of features before training each tree. This can be used to speed up training and deal with overfitting. Defaults to 0.9.
- **bagging\_freq** (*int*) – Frequency for bagging. 0 means bagging is disabled. k means perform bagging at every k iteration. Every k-th iteration, LightGBM will randomly select `bagging_fraction * 100 %` of the data to use for the next k iterations. Defaults to 0.
- **n\_jobs** (*int or None*) – Number of threads to run in parallel. -1 uses all threads. Defaults to -1.
- **random\_seed** (*int*) – Seed for the random number generator. Defaults to 0.

#### Attributes

<b>hyper-parameter_ranges</b>	{ "learning_rate": Real(0.000001, 1), "boosting_type": ["gbdt", "dart", "goss", "rf"], "n_estimators": Integer(10, 100), "max_depth": Integer(0, 10), "num_leaves": Integer(2, 100), "min_child_samples": Integer(1, 100), "bagging_fraction": Real(0.000001, 1), "bagging_freq": Integer(0, 1), }
<b>model_family</b>	ModelFamily.LIGHTGBM
<b>modifies_features</b>	True
<b>modifies_target</b>	False
<b>name</b>	LightGBM Classifier
<b>predict_uses_y</b>	False
<b>SEED_MAX</b>	SEED_BOUNDS.max_bound
<b>SEED_MIN</b>	0
<b>supported_problem_types</b>	[ ProblemTypes.BINARY, ProblemTypes.MULTICLASS, ProblemTypes.TIME_SERIES_BINARY, ProblemTypes.TIME_SERIES_MULTICLASS, ]

#### Methods

<i>clone</i>	Constructs a new component with the same parameters and random state.
<i>default_parameters</i>	Returns the default parameters for this component.
<i>describe</i>	Describe a component and its parameters
<i>feature_importance</i>	Returns importance associated with each feature.
<i>fit</i>	Fits component to data
<i>load</i>	Loads component at file path
<i>needs_fitting</i>	Returns boolean determining if component needs fitting before
<i>parameters</i>	Returns the parameters which were used to initialize the component
<i>predict</i>	Make predictions using selected features.
<i>predict_proba</i>	Make probability estimates for labels.
<i>save</i>	Saves component at file path

#### **clone** (*self*)

Constructs a new component with the same parameters and random state.

**Returns** A new instance of this component with identical parameters and random state.



**default\_parameters** (*cls*)

Returns the default parameters for this component.

Our convention is that `Component.default_parameters == Component().parameters`.

**Returns** default parameters for this component.

**Return type** dict

**describe** (*self*, *print\_name=False*, *return\_dict=False*)

Describe a component and its parameters

**Parameters**

- **print\_name** (*bool*, *optional*) – whether to print name of component
- **return\_dict** (*bool*, *optional*) – whether to return description as dictionary in the format {"name": name, "parameters": parameters}

**Returns** prints and returns dictionary

**Return type** None or dict

**property feature\_importance** (*self*)

Returns importance associated with each feature.

**Returns** Importance associated with each feature

**Return type** np.ndarray

**fit** (*self*, *X*, *y=None*)

Fits component to data

**Parameters**

- **X** (*list*, *pd.DataFrame* or *np.ndarray*) – The input training data of shape [n\_samples, n\_features]
- **y** (*list*, *pd.Series*, *np.ndarray*, *optional*) – The target training data of length [n\_samples]

**Returns** self

**static load** (*file\_path*)

Loads component at file path

**Parameters** **file\_path** (*str*) – Location to load file

**Returns** ComponentBase object

**needs\_fitting** (*self*)

Returns boolean determining if component needs fitting before calling predict, predict\_proba, transform, or feature\_importances. This can be overridden to False for components that do not need to be fit or whose fit methods do nothing.

**property parameters** (*self*)

Returns the parameters which were used to initialize the component

**predict** (*self*, *X*)

Make predictions using selected features.

**Parameters** **X** (*pd.DataFrame*, *np.ndarray*) – Data of shape [n\_samples, n\_features]

**Returns** Predicted values

**Return type** pd.Series

**predict\_proba** (*self*, *X*)

Make probability estimates for labels.

**Parameters** *X* (*pd.DataFrame*, or *np.ndarray*) – Features

**Returns** Probability estimates

**Return type** *pd.Series*

**save** (*self*, *file\_path*, *pickle\_protocol=cloudpickle.DEFAULT\_PROTOCOL*)

Saves component at file path

**Parameters**

- **file\_path** (*str*) – Location to save file
- **pickle\_protocol** (*int*) – The pickle data stream format.

**Returns** None

```
class evalml.pipelines.components.LightGBMRegressor (boosting_type='gbdt', learning_rate=0.1, n_estimators=20, max_depth=0, num_leaves=31, min_child_samples=20, bagging_fraction=0.9, bagging_freq=0, n_jobs=-1, random_seed=0, **kwargs)
```

LightGBM Regressor.

**Parameters**

- **boosting\_type** (*string*) – Type of boosting to use. Defaults to “gbdt”. - ‘gbdt’ uses traditional Gradient Boosting Decision Tree - “dart”, uses Dropouts meet Multiple Additive Regression Trees - “goss”, uses Gradient-based One-Side Sampling - “rf”, uses Random Forest
- **learning\_rate** (*float*) – Boosting learning rate. Defaults to 0.1.
- **n\_estimators** (*int*) – Number of boosted trees to fit. Defaults to 100.
- **max\_depth** (*int*) – Maximum tree depth for base learners, <=0 means no limit. Defaults to 0.
- **num\_leaves** (*int*) – Maximum tree leaves for base learners. Defaults to 31.
- **min\_child\_samples** (*int*) – Minimum number of data needed in a child (leaf). Defaults to 20.
- **bagging\_fraction** (*float*) – LightGBM will randomly select a subset of features on each iteration (tree) without resampling if this is smaller than 1.0. For example, if set to 0.8, LightGBM will select 80% of features before training each tree. This can be used to speed up training and deal with overfitting. Defaults to 0.9.
- **bagging\_freq** (*int*) – Frequency for bagging. 0 means bagging is disabled. k means perform bagging at every k iteration. Every k-th iteration, LightGBM will randomly select bagging\_fraction \* 100 % of the data to use for the next k iterations. Defaults to 0.
- **n\_jobs** (*int* or *None*) – Number of threads to run in parallel. -1 uses all threads. Defaults to -1.
- **random\_seed** (*int*) – Seed for the random number generator. Defaults to 0.

**Attributes**

<b>hyper-parameter_ranges</b>	{ "learning_rate": Real(0.000001, 1), "boosting_type": ["gbdt", "dart", "goss", "rf"], "n_estimators": Integer(10, 100), "max_depth": Integer(0, 10), "num_leaves": Integer(2, 100), "min_child_samples": Integer(1, 100), "bagging_fraction": Real(0.000001, 1), "bagging_freq": Integer(0, 1),}
<b>model_family</b>	ModelFamily.LIGHTGBM
<b>modifies_features</b>	True
<b>modifies_target</b>	False
<b>name</b>	LightGBM Regressor
<b>pre-dict_uses_y</b>	False
<b>SEED_MAX</b>	SEED_BOUNDS.max_bound
<b>SEED_MIN</b>	0
<b>supported_problem_types</b>	[ProblemTypes.REGRESSION]

## Methods

<i>clone</i>	Constructs a new component with the same parameters and random state.
<i>default_parameters</i>	Returns the default parameters for this component.
<i>describe</i>	Describe a component and its parameters
<i>feature_importance</i>	Returns importance associated with each feature.
<i>fit</i>	Fits component to data
<i>load</i>	Loads component at file path
<i>needs_fitting</i>	Returns boolean determining if component needs fitting before
<i>parameters</i>	Returns the parameters which were used to initialize the component
<i>predict</i>	Make predictions using selected features.
<i>predict_proba</i>	Make probability estimates for labels.
<i>save</i>	Saves component at file path

### **clone** (*self*)

Constructs a new component with the same parameters and random state.

**Returns** A new instance of this component with identical parameters and random state.

### **default\_parameters** (*cls*)

Returns the default parameters for this component.

Our convention is that `Component.default_parameters == Component().parameters`.

**Returns** default parameters for this component.

**Return type** dict

### **describe** (*self*, *print\_name=False*, *return\_dict=False*)

Describe a component and its parameters

#### Parameters

- **print\_name** (*bool*, *optional*) – whether to print name of component

- **return\_dict** (*bool, optional*) – whether to return description as dictionary in the format {"name": name, "parameters": parameters}

**Returns** prints and returns dictionary

**Return type** None or dict

**property feature\_importance** (*self*)

Returns importance associated with each feature.

**Returns** Importance associated with each feature

**Return type** np.ndarray

**fit** (*self, X, y=None*)

Fits component to data

**Parameters**

- **X** (*list, pd.DataFrame or np.ndarray*) – The input training data of shape [n\_samples, n\_features]
- **y** (*list, pd.Series, np.ndarray, optional*) – The target training data of length [n\_samples]

**Returns** self

**static load** (*file\_path*)

Loads component at file path

**Parameters** **file\_path** (*str*) – Location to load file

**Returns** ComponentBase object

**needs\_fitting** (*self*)

Returns boolean determining if component needs fitting before calling predict, predict\_proba, transform, or feature\_importances. This can be overridden to False for components that do not need to be fit or whose fit methods do nothing.

**property parameters** (*self*)

Returns the parameters which were used to initialize the component

**predict** (*self, X*)

Make predictions using selected features.

**Parameters** **X** (*pd.DataFrame, np.ndarray*) – Data of shape [n\_samples, n\_features]

**Returns** Predicted values

**Return type** pd.Series

**predict\_proba** (*self, X*)

Make probability estimates for labels.

**Parameters** **X** (*pd.DataFrame, or np.ndarray*) – Features

**Returns** Probability estimates

**Return type** pd.Series

**save** (*self, file\_path, pickle\_protocol=cloudpickle.DEFAULT\_PROTOCOL*)

Saves component at file path

**Parameters**

- **file\_path** (*str*) – Location to save file

- **pickle\_protocol** (*int*) – The pickle data stream format.

**Returns** None

**class** evalml.pipelines.components.**LinearDiscriminantAnalysis** (*n\_components=None*,  
*random\_seed=0*,  
*\*\*kwargs*)

Reduces the number of features by using Linear Discriminant Analysis.

#### Parameters

- **n\_components** (*int*) – The number of features to maintain after computation. Defaults to None.
- **random\_seed** (*int*) – Seed for the random number generator. Defaults to 0.

#### Attributes

<b>hyper-parameter_ranges</b>	{}
<b>model_family</b>	ModelFamily.NONE
<b>modifies_features</b>	True
<b>modifies_target</b>	False
<b>name</b>	Linear Discriminant Analysis Transformer

#### Methods

<i>clone</i>	Constructs a new component with the same parameters and random state.
<i>default_parameters</i>	Returns the default parameters for this component.
<i>describe</i>	Describe a component and its parameters
<i>fit</i>	Fits component to data
<i>fit_transform</i>	Fits on X and transforms X
<i>load</i>	Loads component at file path
<i>needs_fitting</i>	Returns boolean determining if component needs fitting before
<i>parameters</i>	Returns the parameters which were used to initialize the component
<i>save</i>	Saves component at file path
<i>transform</i>	Transforms data X.

**clone** (*self*)

Constructs a new component with the same parameters and random state.

**Returns** A new instance of this component with identical parameters and random state.

**default\_parameters** (*cls*)

Returns the default parameters for this component.

Our convention is that `Component.default_parameters == Component().parameters`.

**Returns** default parameters for this component.

**Return type** dict

**describe** (*self*, *print\_name=False*, *return\_dict=False*)

Describe a component and its parameters

**Parameters**

- **print\_name** (*bool*, *optional*) – whether to print name of component
- **return\_dict** (*bool*, *optional*) – whether to return description as dictionary in the format {"name": name, "parameters": parameters}

**Returns** prints and returns dictionary

**Return type** None or dict

**fit** (*self*, *X*, *y*)

Fits component to data

**Parameters**

- **X** (*list*, *pd.DataFrame* or *np.ndarray*) – The input training data of shape [n\_samples, n\_features]
- **y** (*list*, *pd.Series*, *np.ndarray*, *optional*) – The target training data of length [n\_samples]

**Returns** self

**fit\_transform** (*self*, *X*, *y=None*)

Fits on X and transforms X

**Parameters**

- **X** (*pd.DataFrame*) – Data to fit and transform
- **y** (*pd.Series*) – Target data

**Returns** Transformed X

**Return type** pd.DataFrame

**static load** (*file\_path*)

Loads component at file path

**Parameters** **file\_path** (*str*) – Location to load file

**Returns** ComponentBase object

**needs\_fitting** (*self*)

Returns boolean determining if component needs fitting before calling predict, predict\_proba, transform, or feature\_importances. This can be overridden to False for components that do not need to be fit or whose fit methods do nothing.

**property parameters** (*self*)

Returns the parameters which were used to initialize the component

**save** (*self*, *file\_path*, *pickle\_protocol=cloudpickle.DEFAULT\_PROTOCOL*)

Saves component at file path

**Parameters**

- **file\_path** (*str*) – Location to save file
- **pickle\_protocol** (*int*) – The pickle data stream format.

**Returns** None

**transform** (*self*, *X*, *y=None*)

Transforms data *X*.

#### Parameters

- **x** (*pd.DataFrame*) – Data to transform.
- **y** (*pd.Series*, *optional*) – Target data.

**Returns** Transformed *X*

**Return type** *pd.DataFrame*

**class** evalml.pipelines.components.**LinearRegressor** (*fit\_intercept=True*, *normalize=False*, *n\_jobs=-1*, *random\_seed=0*, *\*\*kwargs*)

Linear Regressor.

#### Parameters

- **fit\_intercept** (*boolean*) – Whether to calculate the intercept for this model. If set to *False*, no intercept will be used in calculations (i.e. data is expected to be centered). Defaults to *True*.
- **normalize** (*boolean*) – If *True*, the regressors will be normalized before regression by subtracting the mean and dividing by the l2-norm. This parameter is ignored when *fit\_intercept* is set to *False*. Defaults to *False*.
- **n\_jobs** (*int or None*) – Number of jobs to run in parallel. -1 uses all threads. Defaults to -1.
- **random\_seed** (*int*) – Seed for the random number generator. Defaults to 0.

#### Attributes

hyper-parameter_ranges	{ “fit_intercept”: [True, False], “normalize”: [True, False]}
model_family	ModelFamily.LINEAR_MODEL
modifies_features	True
modifies_target	False
name	Linear Regressor
pre-dict_uses_y	False
supported_problem_types	[ ProblemTypes.REGRESSION, ProblemTypes.TIME_SERIES_REGRESSION,]

#### Methods

<i>clone</i>	Constructs a new component with the same parameters and random state.
<i>default_parameters</i>	Returns the default parameters for this component.
<i>describe</i>	Describe a component and its parameters
<i>feature_importance</i>	Returns importance associated with each feature.
<i>fit</i>	Fits component to data
<i>load</i>	Loads component at file path

continues on next page

Table 569 – continued from previous page

<code>needs_fitting</code>	Returns boolean determining if component needs fitting before
<code>parameters</code>	Returns the parameters which were used to initialize the component
<code>predict</code>	Make predictions using selected features.
<code>predict_proba</code>	Make probability estimates for labels.
<code>save</code>	Saves component at file path

**clone** (*self*)

Constructs a new component with the same parameters and random state.

**Returns** A new instance of this component with identical parameters and random state.

**default\_parameters** (*cls*)

Returns the default parameters for this component.

Our convention is that `Component.default_parameters == Component().parameters`.

**Returns** default parameters for this component.

**Return type** dict

**describe** (*self*, *print\_name=False*, *return\_dict=False*)

Describe a component and its parameters

**Parameters**

- **print\_name** (*bool*, *optional*) – whether to print name of component
- **return\_dict** (*bool*, *optional*) – whether to return description as dictionary in the format {"name": name, "parameters": parameters}

**Returns** prints and returns dictionary

**Return type** None or dict

**property feature\_importance** (*self*)

Returns importance associated with each feature.

**Returns** Importance associated with each feature

**Return type** np.ndarray

**fit** (*self*, *X*, *y=None*)

Fits component to data

**Parameters**

- **X** (*list*, *pd.DataFrame* or *np.ndarray*) – The input training data of shape [n\_samples, n\_features]
- **y** (*list*, *pd.Series*, *np.ndarray*, *optional*) – The target training data of length [n\_samples]

**Returns** self

**static load** (*file\_path*)

Loads component at file path

**Parameters** **file\_path** (*str*) – Location to load file

**Returns** ComponentBase object



**needs\_fitting** (*self*)

Returns boolean determining if component needs fitting before calling predict, predict\_proba, transform, or feature\_importances. This can be overridden to False for components that do not need to be fit or whose fit methods do nothing.

**property parameters** (*self*)

Returns the parameters which were used to initialize the component

**predict** (*self*, *X*)

Make predictions using selected features.

**Parameters** *X* (*pd.DataFrame*, *np.ndarray*) – Data of shape [n\_samples, n\_features]

**Returns** Predicted values

**Return type** *pd.Series*

**predict\_proba** (*self*, *X*)

Make probability estimates for labels.

**Parameters** *X* (*pd.DataFrame*, or *np.ndarray*) – Features

**Returns** Probability estimates

**Return type** *pd.Series*

**save** (*self*, *file\_path*, *pickle\_protocol=cloudpickle.DEFAULT\_PROTOCOL*)

Saves component at file path

**Parameters**

- **file\_path** (*str*) – Location to save file
- **pickle\_protocol** (*int*) – The pickle data stream format.

**Returns** None

```
class evalml.pipelines.components.LogisticRegressionClassifier (penalty='l2',
                                                                C=1.0,
                                                                multi_class='auto',
                                                                solver='lbfgs',
                                                                n_jobs=- 1, ran-
                                                                dom_seed=0,
                                                                **kwargs)
```

Logistic Regression Classifier.

**Parameters**

- **penalty** (*{ "l1", "l2", "elasticnet", "none" }*) – The norm used in penalization. Defaults to “l2”.
- **C** (*float*) – Inverse of regularization strength. Must be a positive float. Defaults to 1.0.
- **multi\_class** (*{ "auto", "ovr", "multinomial" }*) – If the option chosen is “ovr”, then a binary problem is fit for each label. For “multinomial” the loss minimised is the multinomial loss fit across the entire probability distribution, even when the data is binary. “multinomial” is unavailable when solver=“liblinear”. “auto” selects “ovr” if the data is binary, or if solver=“liblinear”, and otherwise selects “multinomial”. Defaults to “auto”.
- **solver** (*{ "newton-cg", "lbfgs", "liblinear", "sag", "saga" }*) – Algorithm to use in the optimization problem. For small datasets, “liblinear” is a good choice, whereas “sag” and “saga” are faster for large ones. For multiclass problems, only “newton-cg”, “sag”, “saga” and “lbfgs” handle multinomial loss; “liblinear” is limited to one-versus-rest schemes.

- “newton-cg”, “lbfgs”, “sag” and “saga” handle L2 or no penalty
- “liblinear” and “saga” also handle L1 penalty
- “saga” also supports “elasticnet” penalty
- “liblinear” does not support setting `penalty='none'`

Defaults to “lbfgs”.

- **n\_jobs** (*int*) – Number of parallel threads used to run xgboost. Note that creating thread contention will significantly slow down the algorithm. Defaults to -1.
- **random\_seed** (*int*) – Seed for the random number generator. Defaults to 0.

#### Attributes

<b>hyper-parameter_ranges</b>	{ “penalty”: [“l2”], “C”: Real(0.01, 10), }
<b>model_family</b>	ModelFamily.LINEAR_MODEL
<b>modifies_features</b>	True
<b>modifies_target</b>	False
<b>name</b>	Logistic Regression Classifier
<b>predict_uses_y</b>	False
<b>supported_problem_types</b>	[ ProblemTypes.BINARY, ProblemTypes.MULTICLASS, ProblemTypes.TIME_SERIES_BINARY, ProblemTypes.TIME_SERIES_MULTICLASS, ]

#### Methods

<i>clone</i>	Constructs a new component with the same parameters and random state.
<i>default_parameters</i>	Returns the default parameters for this component.
<i>describe</i>	Describe a component and its parameters
<i>feature_importance</i>	Returns importance associated with each feature.
<i>fit</i>	Fits component to data
<i>load</i>	Loads component at file path
<i>needs_fitting</i>	Returns boolean determining if component needs fitting before
<i>parameters</i>	Returns the parameters which were used to initialize the component
<i>predict</i>	Make predictions using selected features.
<i>predict_proba</i>	Make probability estimates for labels.
<i>save</i>	Saves component at file path

#### **clone** (*self*)

Constructs a new component with the same parameters and random state.

**Returns** A new instance of this component with identical parameters and random state.

#### **default\_parameters** (*cls*)

Returns the default parameters for this component.

Our convention is that `Component.default_parameters == Component().parameters`.

**Returns** default parameters for this component.

**Return type** dict

**describe** (*self*, *print\_name=False*, *return\_dict=False*)

Describe a component and its parameters

**Parameters**

- **print\_name** (*bool*, *optional*) – whether to print name of component
- **return\_dict** (*bool*, *optional*) – whether to return description as dictionary in the format {"name": name, "parameters": parameters}

**Returns** prints and returns dictionary

**Return type** None or dict

**property feature\_importance** (*self*)

Returns importance associated with each feature.

**Returns** Importance associated with each feature

**Return type** np.ndarray

**fit** (*self*, *X*, *y=None*)

Fits component to data

**Parameters**

- **X** (*list*, *pd.DataFrame* or *np.ndarray*) – The input training data of shape [n\_samples, n\_features]
- **y** (*list*, *pd.Series*, *np.ndarray*, *optional*) – The target training data of length [n\_samples]

**Returns** self

**static load** (*file\_path*)

Loads component at file path

**Parameters** **file\_path** (*str*) – Location to load file

**Returns** ComponentBase object

**needs\_fitting** (*self*)

Returns boolean determining if component needs fitting before calling predict, predict\_proba, transform, or feature\_importances. This can be overridden to False for components that do not need to be fit or whose fit methods do nothing.

**property parameters** (*self*)

Returns the parameters which were used to initialize the component

**predict** (*self*, *X*)

Make predictions using selected features.

**Parameters** **X** (*pd.DataFrame*, *np.ndarray*) – Data of shape [n\_samples, n\_features]

**Returns** Predicted values

**Return type** pd.Series

**predict\_proba** (*self*, *X*)

Make probability estimates for labels.

**Parameters** **X** (*pd.DataFrame*, or *np.ndarray*) – Features

**Returns** Probability estimates

**Return type** `pd.Series`

**save** (*self*, *file\_path*, *pickle\_protocol=cloudpickle.DEFAULT\_PROTOCOL*)

Saves component at file path

**Parameters**

- **file\_path** (*str*) – Location to save file
- **pickle\_protocol** (*int*) – The pickle data stream format.

**Returns** `None`

**class** `evalml.pipelines.components.LogTransformer` (*random\_seed=0*)

Applies a log transformation to the target data.

**Attributes**

<b>hyper-parameter_ranges</b>	{}
<b>model_family</b>	ModelFamily.NONE
<b>modifies_features</b>	False
<b>modifies_target</b>	True
<b>name</b>	Log Transformer

**Methods**

<i>clone</i>	Constructs a new component with the same parameters and random state.
<i>default_parameters</i>	Returns the default parameters for this component.
<i>describe</i>	Describe a component and its parameters
<i>fit</i>	Fits the LogTransformer.
<i>fit_transform</i>	Log transforms the target variable.
<i>inverse_transform</i>	Inverts the transformation done by the transform method.
<i>load</i>	Loads component at file path
<i>needs_fitting</i>	Returns boolean determining if component needs fitting before
<i>parameters</i>	Returns the parameters which were used to initialize the component
<i>save</i>	Saves component at file path
<i>transform</i>	Log transforms the target variable.

**clone** (*self*)

Constructs a new component with the same parameters and random state.

**Returns** A new instance of this component with identical parameters and random state.

**default\_parameters** (*cls*)

Returns the default parameters for this component.

Our convention is that `Component.default_parameters == Component().parameters`.

**Returns** default parameters for this component.

**Return type** dict

**describe** (*self*, *print\_name=False*, *return\_dict=False*)

Describe a component and its parameters

**Parameters**

- **print\_name** (*bool*, *optional*) – whether to print name of component
- **return\_dict** (*bool*, *optional*) – whether to return description as dictionary in the format {"name": name, "parameters": parameters}

**Returns** prints and returns dictionary

**Return type** None or dict

**fit** (*self*, *X*, *y=None*)

Fits the LogTransformer.

**Parameters**

- **X** (*pd.DataFrame* or *np.ndarray*) – Ignored.
- **y** (*pd.Series*, *optional*) – Ignored.

**Returns** self

**fit\_transform** (*self*, *X*, *y=None*)

Log transforms the target variable.

**Parameters**

- **X** (*pd.DataFrame*, *optional*) – Ignored.
- **y** (*pd.Series*) – Target variable to log transform.

**Returns**

**The input features are returned without modification. The target** variable y is log transformed.

**Return type** tuple of *pd.DataFrame*, *pd.Series*

**inverse\_transform** (*self*, *y*)

Inverts the transformation done by the transform method.

**Arguments:** y (*pd.Series*): Target transformed by this component.

**Returns** Target without the transformation.

**Return type** *pd.Series*∅

**static load** (*file\_path*)

Loads component at file path

**Parameters** **file\_path** (*str*) – Location to load file

**Returns** ComponentBase object

**needs\_fitting** (*self*)

Returns boolean determining if component needs fitting before calling predict, predict\_proba, transform, or feature\_importances. This can be overridden to False for components that do not need to be fit or whose fit methods do nothing.

**property parameters** (*self*)

Returns the parameters which were used to initialize the component

**save** (*self*, *file\_path*, *pickle\_protocol*=*cloudpickle.DEFAULT\_PROTOCOL*)

Saves component at file path

**Parameters**

- **file\_path** (*str*) – Location to save file
- **pickle\_protocol** (*int*) – The pickle data stream format.

**Returns** None

**transform** (*self*, *X*, *y*=*None*)

Log transforms the target variable.

**Parameters**

- **x** (*pd.DataFrame*, *optional*) – Ignored.
- **y** (*pd.Series*) – Target data to log transform.

**Returns**

**The input features are returned without modification. The target** variable *y* is log transformed.

**Return type** tuple of *pd.DataFrame*, *pd.Series*

**class** evalml.pipelines.components.**LSA** (*random\_seed*=0, *\*\*kwargs*)

Transformer to calculate the Latent Semantic Analysis Values of text input.

**Parameters** **random\_seed** (*int*) – Seed for the random number generator. Defaults to 0.

**Attributes**

<b>hyper-parameter_ranges</b>	{}
<b>model_family</b>	ModelFamily.NONE
<b>modifies_features</b>	True
<b>modifies_target</b>	False
<b>name</b>	LSA Transformer

**Methods**

<i>clone</i>	Constructs a new component with the same parameters and random state.
<i>default_parameters</i>	Returns the default parameters for this component.
<i>describe</i>	Describe a component and its parameters
<i>fit</i>	Fits component to data
<i>fit_transform</i>	Fits on <i>X</i> and transforms <i>X</i>
<i>load</i>	Loads component at file path
<i>needs_fitting</i>	Returns boolean determining if component needs fitting before

continues on next page

Table 572 – continued from previous page

<i>parameters</i>	Returns the parameters which were used to initialize the component
<i>save</i>	Saves component at file path
<i>transform</i>	Transforms data X by applying the LSA pipeline.

**clone** (*self*)

Constructs a new component with the same parameters and random state.

**Returns** A new instance of this component with identical parameters and random state.

**default\_parameters** (*cls*)

Returns the default parameters for this component.

Our convention is that `Component.default_parameters == Component().parameters`.

**Returns** default parameters for this component.

**Return type** dict

**describe** (*self*, *print\_name=False*, *return\_dict=False*)

Describe a component and its parameters

**Parameters**

- **print\_name** (*bool*, *optional*) – whether to print name of component
- **return\_dict** (*bool*, *optional*) – whether to return description as dictionary in the format {"name": name, "parameters": parameters}

**Returns** prints and returns dictionary

**Return type** None or dict

**fit** (*self*, *X*, *y=None*)

Fits component to data

**Parameters**

- **X** (*list*, *pd.DataFrame* or *np.ndarray*) – The input training data of shape [n\_samples, n\_features]
- **y** (*list*, *pd.Series*, *np.ndarray*, *optional*) – The target training data of length [n\_samples]

**Returns** self

**fit\_transform** (*self*, *X*, *y=None*)

Fits on X and transforms X

**Parameters**

- **X** (*pd.DataFrame*) – Data to fit and transform
- **y** (*pd.Series*) – Target data

**Returns** Transformed X

**Return type** pd.DataFrame

**static load** (*file\_path*)

Loads component at file path

**Parameters** **file\_path** (*str*) – Location to load file

**Returns** ComponentBase object

**needs\_fitting** (*self*)

Returns boolean determining if component needs fitting before calling predict, predict\_proba, transform, or feature\_importances. This can be overridden to False for components that do not need to be fit or whose fit methods do nothing.

**property parameters** (*self*)

Returns the parameters which were used to initialize the component

**save** (*self*, *file\_path*, *pickle\_protocol*=cloudpickle.DEFAULT\_PROTOCOL)

Saves component at file path

#### Parameters

- **file\_path** (*str*) – Location to save file
- **pickle\_protocol** (*int*) – The pickle data stream format.

**Returns** None

**transform** (*self*, *X*, *y*=None)

Transforms data X by applying the LSA pipeline.

#### Parameters

- **X** (*pd.DataFrame*) – The data to transform.
- **y** (*pd.Series*, *optional*) – Ignored.

**Returns**

**Transformed X. The original column is removed and replaced with two columns of the format** *LSA(original\_column\_name)[feature\_number]*, where *feature\_number* is 0 or 1.

**Return type** *pd.DataFrame*

```
class evalml.pipelines.components.OneHotEncoder (top_n=10, features_to_encode=None,
                                                categories=None, drop='if_binary',
                                                handle_unknown='ignore', handle_missing='error',
                                                random_seed=0,
                                                **kwargs)
```

A transformer that encodes categorical features in a one-hot numeric array.

#### Parameters

- **top\_n** (*int*) – Number of categories per column to encode. If None, all categories will be encoded. Otherwise, the *n* most frequent will be encoded and all others will be dropped. Defaults to 10.
- **features\_to\_encode** (*list[str]*) – List of columns to encode. All other columns will remain untouched. If None, all appropriate columns will be encoded. Defaults to None.
- **categories** (*list*) – A two dimensional list of categories, where *categories[i]* is a list of the categories for the column at index *i*. This can also be None, or “auto” if *top\_n* is not None. Defaults to None.
- **drop** (*string*, *list*) – Method (“first” or “if\_binary”) to use to drop one category per feature. Can also be a list specifying which categories to drop for each feature. Defaults to ‘if\_binary’.
- **handle\_unknown** (*string*) – Whether to ignore or error for unknown categories for a feature encountered during *fit* or *transform*. If either *top\_n* or *categories* is used to limit the number of categories per column, this must be “ignore”. Defaults to “ignore”.



- **handle\_missing** (*string*) – Options for how to handle missing (NaN) values encountered during *fit* or *transform*. If this is set to “as\_category” and NaN values are within the *n* most frequent, “nan” values will be encoded as their own column. If this is set to “error”, any missing values encountered will raise an error. Defaults to “error”.
- **random\_seed** (*int*) – Seed for the random number generator. Defaults to 0.

#### Attributes

<b>hyper-parameter_ranges</b>	{}
<b>model_family</b>	ModelFamily.NONE
<b>modifies_features</b>	True
<b>modifies_target</b>	False
<b>name</b>	One Hot Encoder

#### Methods

<i>categories</i>	Returns a list of the unique categories to be encoded for the particular feature, in order.
<i>clone</i>	Constructs a new component with the same parameters and random state.
<i>default_parameters</i>	Returns the default parameters for this component.
<i>describe</i>	Describe a component and its parameters
<i>fit</i>	Fits component to data
<i>fit_transform</i>	Fits on X and transforms X
<i>get_feature_names</i>	Return feature names for the categorical features after fitting.
<i>load</i>	Loads component at file path
<i>needs_fitting</i>	Returns boolean determining if component needs fitting before
<i>parameters</i>	Returns the parameters which were used to initialize the component
<i>save</i>	Saves component at file path
<i>transform</i>	One-hot encode the input data.

#### **categories** (*self*, *feature\_name*)

Returns a list of the unique categories to be encoded for the particular feature, in order.

**Parameters** **feature\_name** (*str*) – the name of any feature provided to one-hot encoder during fit

**Returns** the unique categories, in the same dtype as they were provided during fit

**Return type** np.ndarray

#### **clone** (*self*)

Constructs a new component with the same parameters and random state.

**Returns** A new instance of this component with identical parameters and random state.

#### **default\_parameters** (*cls*)

Returns the default parameters for this component.

Our convention is that `Component.default_parameters == Component().parameters`.

**Returns** default parameters for this component.

**Return type** dict

**describe** (*self*, *print\_name=False*, *return\_dict=False*)

Describe a component and its parameters

**Parameters**

- **print\_name** (*bool*, *optional*) – whether to print name of component
- **return\_dict** (*bool*, *optional*) – whether to return description as dictionary in the format {"name": name, "parameters": parameters}

**Returns** prints and returns dictionary

**Return type** None or dict

**fit** (*self*, *X*, *y=None*)

Fits component to data

**Parameters**

- **X** (*list*, *pd.DataFrame* or *np.ndarray*) – The input training data of shape [n\_samples, n\_features]
- **y** (*list*, *pd.Series*, *np.ndarray*, *optional*) – The target training data of length [n\_samples]

**Returns** self

**fit\_transform** (*self*, *X*, *y=None*)

Fits on X and transforms X

**Parameters**

- **X** (*pd.DataFrame*) – Data to fit and transform
- **y** (*pd.Series*) – Target data

**Returns** Transformed X

**Return type** pd.DataFrame

**get\_feature\_names** (*self*)

Return feature names for the categorical features after fitting.

Feature names are formatted as {column name}\_{category name}. In the event of a duplicate name, an integer will be added at the end of the feature name to distinguish it.

For example, consider a dataframe with a column called “A” and category “x\_y” and another column called “A\_x” with “y”. In this example, the feature names would be “A\_x\_y” and “A\_x\_y\_1”.

**Returns** The feature names after encoding, provided in the same order as input\_features.

**Return type** np.ndarray

**static load** (*file\_path*)

Loads component at file path

**Parameters** **file\_path** (*str*) – Location to load file

**Returns** ComponentBase object

**needs\_fitting** (*self*)

Returns boolean determining if component needs fitting before calling predict, predict\_proba, transform, or feature\_importances. This can be overridden to False for components that do not need to be fit or whose fit methods do nothing.

**property parameters** (*self*)

Returns the parameters which were used to initialize the component

**save** (*self*, *file\_path*, *pickle\_protocol=cloudpickle.DEFAULT\_PROTOCOL*)

Saves component at file path

**Parameters**

- **file\_path** (*str*) – Location to save file
- **pickle\_protocol** (*int*) – The pickle data stream format.

**Returns** None

**transform** (*self*, *X*, *y=None*)

One-hot encode the input data.

**Parameters**

- **X** (*pd.DataFrame*) – Features to one-hot encode.
- **y** (*pd.Series*) – Ignored.

**Returns** Transformed data, where each categorical feature has been encoded into numerical columns using one-hot encoding.

**Return type** *pd.DataFrame*

**class** evalml.pipelines.components.**PCA** (*variance=0.95*, *n\_components=None*, *random\_seed=0*, *\*\*kwargs*)

Reduces the number of features by using Principal Component Analysis (PCA).

**Parameters**

- **variance** (*float*) – The percentage of the original data variance that should be preserved when reducing the number of features. Defaults to 0.95.
- **n\_components** (*int*) – The number of features to maintain after computing SVD. Defaults to None, but will override variance variable if set.
- **random\_seed** (*int*) – Seed for the random number generator. Defaults to 0.

**Attributes**

<b>hyper-parameter_ranges</b>	Real(0.25, 1)}:type: {"variance"}
<b>model_family</b>	ModelFamily.NONE
<b>modifies_features</b>	True
<b>modifies_target</b>	False
<b>name</b>	PCA Transformer

**Methods**

<code>clone</code>	Constructs a new component with the same parameters and random state.
<code>default_parameters</code>	Returns the default parameters for this component.
<code>describe</code>	Describe a component and its parameters
<code>fit</code>	Fits component to data
<code>fit_transform</code>	Fits on X and transforms X
<code>load</code>	Loads component at file path
<code>needs_fitting</code>	Returns boolean determining if component needs fitting before
<code>parameters</code>	Returns the parameters which were used to initialize the component
<code>save</code>	Saves component at file path
<code>transform</code>	Transforms data X.

**clone** (*self*)

Constructs a new component with the same parameters and random state.

**Returns** A new instance of this component with identical parameters and random state.

**default\_parameters** (*cls*)

Returns the default parameters for this component.

Our convention is that `Component.default_parameters == Component().parameters`.

**Returns** default parameters for this component.

**Return type** dict

**describe** (*self*, *print\_name=False*, *return\_dict=False*)

Describe a component and its parameters

**Parameters**

- **print\_name** (*bool*, *optional*) – whether to print name of component
- **return\_dict** (*bool*, *optional*) – whether to return description as dictionary in the format {"name": name, "parameters": parameters}

**Returns** prints and returns dictionary

**Return type** None or dict

**fit** (*self*, *X*, *y=None*)

Fits component to data

**Parameters**

- **X** (*list*, *pd.DataFrame* or *np.ndarray*) – The input training data of shape [n\_samples, n\_features]
- **y** (*list*, *pd.Series*, *np.ndarray*, *optional*) – The target training data of length [n\_samples]

**Returns** self

**fit\_transform** (*self*, *X*, *y=None*)

Fits on X and transforms X

**Parameters**

- **X** (*pd.DataFrame*) – Data to fit and transform

- **y** (*pd.Series*) – Target data

**Returns** Transformed X

**Return type** *pd.DataFrame*

**static load** (*file\_path*)

Loads component at file path

**Parameters** **file\_path** (*str*) – Location to load file

**Returns** *ComponentBase* object

**needs\_fitting** (*self*)

Returns boolean determining if component needs fitting before calling predict, predict\_proba, transform, or feature\_importances. This can be overridden to False for components that do not need to be fit or whose fit methods do nothing.

**property parameters** (*self*)

Returns the parameters which were used to initialize the component

**save** (*self, file\_path, pickle\_protocol=cloudpickle.DEFAULT\_PROTOCOL*)

Saves component at file path

**Parameters**

- **file\_path** (*str*) – Location to save file
- **pickle\_protocol** (*int*) – The pickle data stream format.

**Returns** None

**transform** (*self, X, y=None*)

Transforms data X.

**Parameters**

- **X** (*pd.DataFrame*) – Data to transform.
- **y** (*pd.Series, optional*) – Target data.

**Returns** Transformed X

**Return type** *pd.DataFrame*

**class** evalml.pipelines.components.**PerColumnImputer** (*impute\_strategies=None, default\_impute\_strategy='most\_frequent', random\_seed=0, \*\*kwargs*)

Imputes missing data according to a specified imputation strategy per column.

**Parameters**

- **impute\_strategies** (*dict*) – Column and {"impute\_strategy": strategy, "fill\_value": value} pairings. Valid values for impute strategy include "mean", "median", "most\_frequent", "constant" for numerical data, and "most\_frequent", "constant" for object data types. Defaults to None, which uses "most\_frequent" for all columns. When impute\_strategy == "constant", fill\_value is used to replace missing data. When None, uses 0 when imputing numerical data and "missing\_value" for strings or object data types.
- **default\_impute\_strategy** (*str*) – Impute strategy to fall back on when none is provided for a certain column. Valid values include "mean", "median", "most\_frequent", "constant" for numerical data, and "most\_frequent", "constant" for object data types. Defaults to "most\_frequent".
- **random\_seed** (*int*) – Seed for the random number generator. Defaults to 0.

### Attributes

<b>hyper-parameter_ranges</b>	{}
<b>model_family</b>	ModelFamily.NONE
<b>modifies_features</b>	True
<b>modifies_target</b>	False
<b>name</b>	Per Column Imputer

### Methods

<i>clone</i>	Constructs a new component with the same parameters and random state.
<i>default_parameters</i>	Returns the default parameters for this component.
<i>describe</i>	Describe a component and its parameters
<i>fit</i>	Fits imputers on input data
<i>fit_transform</i>	Fits on X and transforms X
<i>load</i>	Loads component at file path
<i>needs_fitting</i>	Returns boolean determining if component needs fitting before
<i>parameters</i>	Returns the parameters which were used to initialize the component
<i>save</i>	Saves component at file path
<i>transform</i>	Transforms input data by imputing missing values.

#### **clone** (*self*)

Constructs a new component with the same parameters and random state.

**Returns** A new instance of this component with identical parameters and random state.

#### **default\_parameters** (*cls*)

Returns the default parameters for this component.

Our convention is that `Component.default_parameters == Component().parameters`.

**Returns** default parameters for this component.

**Return type** dict

#### **describe** (*self*, *print\_name=False*, *return\_dict=False*)

Describe a component and its parameters

##### Parameters

- **print\_name** (*bool*, *optional*) – whether to print name of component
- **return\_dict** (*bool*, *optional*) – whether to return description as dictionary in the format {"name": name, "parameters": parameters}

**Returns** prints and returns dictionary

**Return type** None or dict

#### **fit** (*self*, *X*, *y=None*)

Fits imputers on input data

**Parameters**

- **X** (*pd.DataFrame* or *np.ndarray*) – The input training data of shape [n\_samples, n\_features] to fit.
- **y** (*pd.Series*, optional) – The target training data of length [n\_samples]. Ignored.

**Returns** self**fit\_transform** (*self*, *X*, *y=None*)

Fits on X and transforms X

**Parameters**

- **X** (*pd.DataFrame*) – Data to fit and transform
- **y** (*pd.Series*) – Target data

**Returns** Transformed X**Return type** *pd.DataFrame***static load** (*file\_path*)

Loads component at file path

**Parameters** **file\_path** (*str*) – Location to load file**Returns** ComponentBase object**needs\_fitting** (*self*)

Returns boolean determining if component needs fitting before calling predict, predict\_proba, transform, or feature\_importances. This can be overridden to False for components that do not need to be fit or whose fit methods do nothing.

**property parameters** (*self*)

Returns the parameters which were used to initialize the component

**save** (*self*, *file\_path*, *pickle\_protocol=cloudpickle.DEFAULT\_PROTOCOL*)

Saves component at file path

**Parameters**

- **file\_path** (*str*) – Location to save file
- **pickle\_protocol** (*int*) – The pickle data stream format.

**Returns** None**transform** (*self*, *X*, *y=None*)

Transforms input data by imputing missing values.

**Parameters**

- **X** (*pd.DataFrame* or *np.ndarray*) – The input training data of shape [n\_samples, n\_features] to transform.
- **y** (*pd.Series*, optional) – The target training data of length [n\_samples]. Ignored.

**Returns** Transformed X**Return type** *pd.DataFrame*

**class** evalml.pipelines.components.**PolynomialDetrender** (*degree=1*, *random\_seed=0*,  
\*\*kwargs)

Removes trends from time series by fitting a polynomial to the data.

**Parameters**

- **degree** (*int*) – Degree for the polynomial. If 1, linear model is fit to the data. If 2, quadratic model is fit, etc. Defaults to 1.
- **random\_seed** (*int*) – Seed for the random number generator. Defaults to 0.

#### Attributes

<b>hyper-parameter_ranges</b>	{ “degree”: Integer(1, 3)}
<b>model_family</b>	ModelFamily.NONE
<b>modifies_features</b>	False
<b>modifies_target</b>	True
<b>name</b>	Polynomial Detrender

#### Methods

<i>clone</i>	Constructs a new component with the same parameters and random state.
<i>default_parameters</i>	Returns the default parameters for this component.
<i>describe</i>	Describe a component and its parameters
<i>fit</i>	Fits the PolynomialDetrender.
<i>fit_transform</i>	Removes fitted trend from target variable.
<i>inverse_transform</i>	Adds back fitted trend to target variable.
<i>load</i>	Loads component at file path
<i>needs_fitting</i>	Returns boolean determining if component needs fitting before
<i>parameters</i>	Returns the parameters which were used to initialize the component
<i>save</i>	Saves component at file path
<i>transform</i>	Removes fitted trend from target variable.

#### **clone** (*self*)

Constructs a new component with the same parameters and random state.

**Returns** A new instance of this component with identical parameters and random state.

#### **default\_parameters** (*cls*)

Returns the default parameters for this component.

Our convention is that `Component.default_parameters == Component().parameters`.

**Returns** default parameters for this component.

**Return type** dict

#### **describe** (*self*, *print\_name=False*, *return\_dict=False*)

Describe a component and its parameters

#### Parameters

- **print\_name** (*bool*, *optional*) – whether to print name of component
- **return\_dict** (*bool*, *optional*) – whether to return description as dictionary in the format {“name”: name, “parameters”: parameters}



**Returns** prints and returns dictionary

**Return type** None or dict

**fit** (*self*, *X*, *y=None*)

Fits the PolynomialDetrender.

**Parameters**

- **X** (*pd.DataFrame*, *optional*) – Ignored.
- **y** (*pd.Series*) – Target variable to detrend.

**Returns** *self*

**fit\_transform** (*self*, *X*, *y=None*)

Removes fitted trend from target variable.

**Parameters**

- **X** (*pd.DataFrame*, *optional*) – Ignored.
- **y** (*pd.Series*) – Target variable to detrend.

**Returns**

**The first element are the input features returned without modification.** The second element is the target variable *y* with the fitted trend removed.

**Return type** tuple of *pd.DataFrame*, *pd.Series*

**inverse\_transform** (*self*, *y*)

Adds back fitted trend to target variable.

**Parameters**

- **X** (*pd.DataFrame*, *optional*) – Ignored.
- **y** (*pd.Series*) – Target variable.

**Returns**

**The first element are the input features returned without modification.** The second element is the target variable *y* with the trend added back.

**Return type** tuple of *pd.DataFrame*, *pd.Series*

**static load** (*file\_path*)

Loads component at file path

**Parameters** **file\_path** (*str*) – Location to load file

**Returns** *ComponentBase* object

**needs\_fitting** (*self*)

Returns boolean determining if component needs fitting before calling *predict*, *predict\_proba*, *transform*, or *feature\_importances*. This can be overridden to *False* for components that do not need to be fit or whose fit methods do nothing.

**property parameters** (*self*)

Returns the parameters which were used to initialize the component

**save** (*self*, *file\_path*, *pickle\_protocol=cloudpickle.DEFAULT\_PROTOCOL*)

Saves component at file path

**Parameters**

- **file\_path** (*str*) – Location to save file

- `pickle_protocol` (*int*) – The pickle data stream format.

**Returns** None

**transform** (*self*, *X*, *y=None*)

Removes fitted trend from target variable.

**Parameters**

- **X** (*pd.DataFrame*, *optional*) – Ignored.
- **y** (*pd.Series*) – Target variable to detrend.

**Returns**

The input features are returned without modification. The target variable *y* is detrended

**Return type** tuple of *pd.DataFrame*, *pd.Series*

```
class evalml.pipelines.components.ProphetRegressor (date_column='ds', change-  
point_prior_scale=0.05, sea-  
sonality_prior_scale=10, hol-  
idays_prior_scale=10, sea-  
sonality_mode='additive',  
random_seed=0,  
stan_backend='CMDSTANPY',  
**kwargs)
```

Prophet is a procedure for forecasting time series data based on an additive model where non-linear trends are fit with yearly, weekly, and daily seasonality, plus holiday effects. It works best with time series that have strong seasonal effects and several seasons of historical data. Prophet is robust to missing data and shifts in the trend, and typically handles outliers well.

More information here: <https://facebook.github.io/prophet/>

**Attributes**

<b>hyper-parameter_ranges</b>	{ "changepoint_prior_scale": Real(0.001, 0.5), "seasonality_prior_scale": Real(0.01, 10), "holidays_prior_scale": Real(0.01, 10), "seasonality_mode": ["additive", "multiplicative"], }
<b>model_family</b>	ModelFamily.PROPHET
<b>modifies_features</b>	True
<b>modifies_target</b>	False
<b>name</b>	Prophet Regressor
<b>pre_dict_uses_y</b>	False
<b>supported_problem_types</b>	[ProblemTypes.TIME_SERIES_REGRESSION]

**Methods**

---

*build\_prophet\_df*

---

*clone*

Constructs a new component with the same parameters and random state.

---

*default\_parameters*

Returns the default parameters for this component.

continues on next page

Table 577 – continued from previous page

<code>describe</code>	Describe a component and its parameters
<code>feature_importance</code>	Returns array of 0's with len(1) as feature_importance is not defined for Prophet regressor.
<code>fit</code>	Fits component to data
<code>get_params</code>	
<code>load</code>	Loads component at file path
<code>needs_fitting</code>	Returns boolean determining if component needs fitting before
<code>parameters</code>	Returns the parameters which were used to initialize the component
<code>predict</code>	Make predictions using selected features.
<code>predict_proba</code>	Make probability estimates for labels.
<code>save</code>	Saves component at file path

**static build\_prophet\_df** (*X*, *y=None*, *date\_column='ds'*)

**clone** (*self*)

Constructs a new component with the same parameters and random state.

**Returns** A new instance of this component with identical parameters and random state.

**default\_parameters** (*cls*)

Returns the default parameters for this component.

Our convention is that `Component.default_parameters == Component().parameters`.

**Returns** default parameters for this component.

**Return type** dict

**describe** (*self*, *print\_name=False*, *return\_dict=False*)

Describe a component and its parameters

**Parameters**

- **print\_name** (*bool*, *optional*) – whether to print name of component
- **return\_dict** (*bool*, *optional*) – whether to return description as dictionary in the format {"name": name, "parameters": parameters}

**Returns** prints and returns dictionary

**Return type** None or dict

**property feature\_importance** (*self*)

Returns array of 0's with len(1) as feature\_importance is not defined for Prophet regressor.

**fit** (*self*, *X*, *y=None*)

Fits component to data

**Parameters**

- **X** (*list*, *pd.DataFrame* or *np.ndarray*) – The input training data of shape [n\_samples, n\_features]
- **y** (*list*, *pd.Series*, *np.ndarray*, *optional*) – The target training data of length [n\_samples]

**Returns** self

**get\_params** (*self*)

**static load** (*file\_path*)

Loads component at file path

**Parameters** *file\_path* (*str*) – Location to load file

**Returns** ComponentBase object

**needs\_fitting** (*self*)

Returns boolean determining if component needs fitting before calling predict, predict\_proba, transform, or feature\_importances. This can be overridden to False for components that do not need to be fit or whose fit methods do nothing.

**property parameters** (*self*)

Returns the parameters which were used to initialize the component

**predict** (*self*, *X*, *y=None*)

Make predictions using selected features.

**Parameters** *X* (*pd.DataFrame*, *np.ndarray*) – Data of shape [n\_samples, n\_features]

**Returns** Predicted values

**Return type** *pd.Series*

**predict\_proba** (*self*, *X*)

Make probability estimates for labels.

**Parameters** *X* (*pd.DataFrame*, or *np.ndarray*) – Features

**Returns** Probability estimates

**Return type** *pd.Series*

**save** (*self*, *file\_path*, *pickle\_protocol=cloudpickle.DEFAULT\_PROTOCOL*)

Saves component at file path

**Parameters**

- **file\_path** (*str*) – Location to save file
- **pickle\_protocol** (*int*) – The pickle data stream format.

**Returns** None

```
class evalml.pipelines.components.RandomForestClassifier (n_estimators=100,  
                                                         max_depth=6, n_jobs=-1,  
                                                         random_seed=0,  
                                                         **kwargs)
```

Random Forest Classifier.

**Parameters**

- **n\_estimators** (*float*) – The number of trees in the forest. Defaults to 100.
- **max\_depth** (*int*) – Maximum tree depth for base learners. Defaults to 6.
- **n\_jobs** (*int* or *None*) – Number of jobs to run in parallel. -1 uses all processes. Defaults to -1.
- **random\_seed** (*int*) – Seed for the random number generator. Defaults to 0.

**Attributes**

<b>hyper-parameter_ranges</b>	{ "n_estimators": Integer(10, 1000), "max_depth": Integer(1, 10), }
<b>model_family</b>	ModelFamily.RANDOM_FOREST
<b>modifies_features</b>	True
<b>modifies_target</b>	False
<b>name</b>	Random Forest Classifier
<b>pre-dict_uses_y</b>	False
<b>supported_problem_types</b>	[ ProblemTypes.BINARY, ProblemTypes.MULTICLASS, ProblemTypes.TIME_SERIES_BINARY, ProblemTypes.TIME_SERIES_MULTICLASS, ]

## Methods

<i>clone</i>	Constructs a new component with the same parameters and random state.
<i>default_parameters</i>	Returns the default parameters for this component.
<i>describe</i>	Describe a component and its parameters
<i>feature_importance</i>	Returns importance associated with each feature.
<i>fit</i>	Fits component to data
<i>load</i>	Loads component at file path
<i>needs_fitting</i>	Returns boolean determining if component needs fitting before
<i>parameters</i>	Returns the parameters which were used to initialize the component
<i>predict</i>	Make predictions using selected features.
<i>predict_proba</i>	Make probability estimates for labels.
<i>save</i>	Saves component at file path

### **clone** (*self*)

Constructs a new component with the same parameters and random state.

**Returns** A new instance of this component with identical parameters and random state.

### **default\_parameters** (*cls*)

Returns the default parameters for this component.

Our convention is that `Component.default_parameters == Component().parameters`.

**Returns** default parameters for this component.

**Return type** dict

### **describe** (*self*, *print\_name=False*, *return\_dict=False*)

Describe a component and its parameters

#### Parameters

- **print\_name** (*bool*, *optional*) – whether to print name of component
- **return\_dict** (*bool*, *optional*) – whether to return description as dictionary in the format {"name": name, "parameters": parameters}

**Returns** prints and returns dictionary

**Return type** None or dict

**property feature\_importance** (*self*)

Returns importance associated with each feature.

**Returns** Importance associated with each feature

**Return type** np.ndarray

**fit** (*self*, *X*, *y=None*)

Fits component to data

**Parameters**

- **X** (*list*, *pd.DataFrame* or *np.ndarray*) – The input training data of shape [n\_samples, n\_features]
- **y** (*list*, *pd.Series*, *np.ndarray*, *optional*) – The target training data of length [n\_samples]

**Returns** self

**static load** (*file\_path*)

Loads component at file path

**Parameters** **file\_path** (*str*) – Location to load file

**Returns** ComponentBase object

**needs\_fitting** (*self*)

Returns boolean determining if component needs fitting before calling predict, predict\_proba, transform, or feature\_importances. This can be overridden to False for components that do not need to be fit or whose fit methods do nothing.

**property parameters** (*self*)

Returns the parameters which were used to initialize the component

**predict** (*self*, *X*)

Make predictions using selected features.

**Parameters** **X** (*pd.DataFrame*, *np.ndarray*) – Data of shape [n\_samples, n\_features]

**Returns** Predicted values

**Return type** pd.Series

**predict\_proba** (*self*, *X*)

Make probability estimates for labels.

**Parameters** **X** (*pd.DataFrame*, or *np.ndarray*) – Features

**Returns** Probability estimates

**Return type** pd.Series

**save** (*self*, *file\_path*, *pickle\_protocol=cloudpickle.DEFAULT\_PROTOCOL*)

Saves component at file path

**Parameters**

- **file\_path** (*str*) – Location to save file
- **pickle\_protocol** (*int*) – The pickle data stream format.

**Returns** None

```
class evalml.pipelines.components.RandomForestRegressor (n_estimators=100,  
                                                    max_depth=6, n_jobs=-  
                                                    1, random_seed=0,  
                                                    **kwargs)
```

Random Forest Regressor.

#### Parameters

- **n\_estimators** (*float*) – The number of trees in the forest. Defaults to 100.
- **max\_depth** (*int*) – Maximum tree depth for base learners. Defaults to 6.
- **n\_jobs** (*int or None*) – Number of jobs to run in parallel. -1 uses all processes. Defaults to -1.
- **random\_seed** (*int*) – Seed for the random number generator. Defaults to 0.

#### Attributes

<b>hyper-parameter_ranges</b>	{ "n_estimators": Integer(10, 1000), "max_depth": Integer(1, 32), }
<b>model_family</b>	ModelFamily.RANDOM_FOREST
<b>modifies_features</b>	True
<b>modifies_target</b>	False
<b>name</b>	Random Forest Regressor
<b>predict_uses_y</b>	False
<b>supported_problem_types</b>	[ ProblemTypes.REGRESSION, ProblemTypes.TIME_SERIES_REGRESSION, ]

#### Methods

<i>clone</i>	Constructs a new component with the same parameters and random state.
<i>default_parameters</i>	Returns the default parameters for this component.
<i>describe</i>	Describe a component and its parameters
<i>feature_importance</i>	Returns importance associated with each feature.
<i>fit</i>	Fits component to data
<i>load</i>	Loads component at file path
<i>needs_fitting</i>	Returns boolean determining if component needs fitting before
<i>parameters</i>	Returns the parameters which were used to initialize the component
<i>predict</i>	Make predictions using selected features.
<i>predict_proba</i>	Make probability estimates for labels.
<i>save</i>	Saves component at file path

**clone** (*self*)

Constructs a new component with the same parameters and random state.

**Returns** A new instance of this component with identical parameters and random state.

**default\_parameters** (*cls*)

Returns the default parameters for this component.

Our convention is that `Component.default_parameters == Component().parameters`.

**Returns** default parameters for this component.

**Return type** dict

**describe** (*self*, *print\_name=False*, *return\_dict=False*)

Describe a component and its parameters

**Parameters**

- **print\_name** (*bool*, *optional*) – whether to print name of component
- **return\_dict** (*bool*, *optional*) – whether to return description as dictionary in the format {"name": name, "parameters": parameters}

**Returns** prints and returns dictionary

**Return type** None or dict

**property feature\_importance** (*self*)

Returns importance associated with each feature.

**Returns** Importance associated with each feature

**Return type** np.ndarray

**fit** (*self*, *X*, *y=None*)

Fits component to data

**Parameters**

- **X** (*list*, *pd.DataFrame* or *np.ndarray*) – The input training data of shape [n\_samples, n\_features]
- **y** (*list*, *pd.Series*, *np.ndarray*, *optional*) – The target training data of length [n\_samples]

**Returns** self

**static load** (*file\_path*)

Loads component at file path

**Parameters** **file\_path** (*str*) – Location to load file

**Returns** ComponentBase object

**needs\_fitting** (*self*)

Returns boolean determining if component needs fitting before calling predict, predict\_proba, transform, or feature\_importances. This can be overridden to False for components that do not need to be fit or whose fit methods do nothing.

**property parameters** (*self*)

Returns the parameters which were used to initialize the component

**predict** (*self*, *X*)

Make predictions using selected features.

**Parameters** **X** (*pd.DataFrame*, *np.ndarray*) – Data of shape [n\_samples, n\_features]

**Returns** Predicted values

**Return type** pd.Series



**predict\_proba** (*self*, *X*)

Make probability estimates for labels.

**Parameters** *X* (*pd.DataFrame*, or *np.ndarray*) – Features

**Returns** Probability estimates

**Return type** *pd.Series*

**save** (*self*, *file\_path*, *pickle\_protocol=cloudpickle.DEFAULT\_PROTOCOL*)

Saves component at file path

**Parameters**

- **file\_path** (*str*) – Location to save file
- **pickle\_protocol** (*int*) – The pickle data stream format.

**Returns** None

```
class evalml.pipelines.components.RFClassifierSelectFromModel (number_features=None,
                                                             n_estimators=10,
                                                             max_depth=None,
                                                             per-
                                                             cent_features=0.5,
                                                             threshold=-
                                                             np.inf, n_jobs=-
                                                             1, ran-
                                                             dom_seed=0,
                                                             **kwargs)
```

Selects top features based on importance weights using a Random Forest classifier.

**Parameters**

- **number\_features** (*int*) – The maximum number of features to select. If both *percent\_features* and *number\_features* are specified, take the greater number of features. Defaults to 0.5. Defaults to None.
- **n\_estimators** (*float*) – The number of trees in the forest. Defaults to 100.
- **max\_depth** (*int*) – Maximum tree depth for base learners. Defaults to 6.
- **percent\_features** (*float*) – Percentage of features to use. If both *percent\_features* and *number\_features* are specified, take the greater number of features. Defaults to 0.5.
- **threshold** (*string* or *float*) – The threshold value to use for feature selection. Features whose importance is greater or equal are kept while the others are discarded. If “median”, then the threshold value is the median of the feature importances. A scaling factor (e.g., “1.25\*mean”) may also be used. Defaults to -np.inf.
- **n\_jobs** (*int* or *None*) – Number of jobs to run in parallel. -1 uses all processes. Defaults to -1.
- **random\_seed** (*int*) – Seed for the random number generator. Defaults to 0.

**Attributes**

<b>hyper-parameter_ranges</b>	{ “percent_features”: Real(0.01, 1), “threshold”: [“mean”, -np.inf], }
<b>model_family</b>	ModelFamily.NONE
<b>modifies_features</b>	True
<b>modifies_target</b>	False
<b>name</b>	RF Classifier Select From Model

## Methods

<i>clone</i>	Constructs a new component with the same parameters and random state.
<i>default_parameters</i>	Returns the default parameters for this component.
<i>describe</i>	Describe a component and its parameters
<i>fit</i>	Fits component to data
<i>fit_transform</i>	Fits on X and transforms X
<i>get_names</i>	Get names of selected features.
<i>load</i>	Loads component at file path
<i>needs_fitting</i>	Returns boolean determining if component needs fitting before
<i>parameters</i>	Returns the parameters which were used to initialize the component
<i>save</i>	Saves component at file path
<i>transform</i>	Transforms input data by selecting features. If the component_obj does not have a transform method, will raise an MethodPropertyNotFoundError exception.

### **clone** (*self*)

Constructs a new component with the same parameters and random state.

**Returns** A new instance of this component with identical parameters and random state.

### **default\_parameters** (*cls*)

Returns the default parameters for this component.

Our convention is that `Component.default_parameters == Component().parameters`.

**Returns** default parameters for this component.

**Return type** dict

### **describe** (*self*, *print\_name=False*, *return\_dict=False*)

Describe a component and its parameters

#### Parameters

- **print\_name** (*bool*, *optional*) – whether to print name of component
- **return\_dict** (*bool*, *optional*) – whether to return description as dictionary in the format {“name”: name, “parameters”: parameters}

**Returns** prints and returns dictionary

**Return type** None or dict

**fit** (*self*, *X*, *y=None*)

Fits component to data

**Parameters**

- **X** (*list*, *pd.DataFrame* or *np.ndarray*) – The input training data of shape [n\_samples, n\_features]
- **y** (*list*, *pd.Series*, *np.ndarray*, *optional*) – The target training data of length [n\_samples]

**Returns** *self*

**fit\_transform** (*self*, *X*, *y=None*)

Fits on X and transforms X

**Parameters**

- **X** (*pd.DataFrame*) – Data to fit and transform
- **y** (*pd.Series*) – Target data

**Returns** Transformed X

**Return type** *pd.DataFrame*

**get\_names** (*self*)

Get names of selected features.

**Returns** List of the names of features selected

**Return type** *list[str]*

**static load** (*file\_path*)

Loads component at file path

**Parameters** **file\_path** (*str*) – Location to load file

**Returns** *ComponentBase* object

**needs\_fitting** (*self*)

Returns boolean determining if component needs fitting before calling predict, predict\_proba, transform, or feature\_importances. This can be overridden to False for components that do not need to be fit or whose fit methods do nothing.

**property parameters** (*self*)

Returns the parameters which were used to initialize the component

**save** (*self*, *file\_path*, *pickle\_protocol=cloudpickle.DEFAULT\_PROTOCOL*)

Saves component at file path

**Parameters**

- **file\_path** (*str*) – Location to save file
- **pickle\_protocol** (*int*) – The pickle data stream format.

**Returns** *None*

**transform** (*self*, *X*, *y=None*)

Transforms input data by selecting features. If the component\_obj does not have a transform method, will raise an *MethodPropertyNotFoundError* exception.

**Parameters**

- **X** (*pd.DataFrame*) – Data to transform.

- **y** (*pd.Series, optional*) – Target data. Ignored.

**Returns** Transformed X

**Return type** `pd.DataFrame`

```
class evalml.pipelines.components.RFRegressorSelectFromModel (number_features=None,
                                                             n_estimators=10,
                                                             max_depth=None,
                                                             per-
                                                             cent_features=0.5,
                                                             threshold=- np.inf,
                                                             n_jobs=- 1, ran-
                                                             dom_seed=0,
                                                             **kwargs)
```

Selects top features based on importance weights using a Random Forest regressor.

#### Parameters

- **number\_features** (*int*) – The maximum number of features to select. If both `percent_features` and `number_features` are specified, take the greater number of features. Defaults to 0.5. Defaults to None.
- **n\_estimators** (*float*) – The number of trees in the forest. Defaults to 100.
- **max\_depth** (*int*) – Maximum tree depth for base learners. Defaults to 6.
- **percent\_features** (*float*) – Percentage of features to use. If both `percent_features` and `number_features` are specified, take the greater number of features. Defaults to 0.5.
- **threshold** (*string or float*) – The threshold value to use for feature selection. Features whose importance is greater or equal are kept while the others are discarded. If “median”, then the threshold value is the median of the feature importances. A scaling factor (e.g., “1.25\*mean”) may also be used. Defaults to `-np.inf`.
- **n\_jobs** (*int or None*) – Number of jobs to run in parallel. -1 uses all processes. Defaults to -1.
- **random\_seed** (*int*) – Seed for the random number generator. Defaults to 0.

#### Attributes

<b>hyper-parameter_ranges</b>	{ “percent_features”: Real(0.01, 1), “threshold”: [“mean”, -np.inf], }
<b>model_family</b>	ModelFamily.NONE
<b>modifies_features</b>	True
<b>modifies_target</b>	False
<b>name</b>	RF Regressor Select From Model

#### Methods

<code>clone</code>	Constructs a new component with the same parameters and random state.
<code>default_parameters</code>	Returns the default parameters for this component.
<code>describe</code>	Describe a component and its parameters

continues on next page

Table 581 – continued from previous page

<code>fit</code>	Fits component to data
<code>fit_transform</code>	Fits on X and transforms X
<code>get_names</code>	Get names of selected features.
<code>load</code>	Loads component at file path
<code>needs_fitting</code>	Returns boolean determining if component needs fitting before
<code>parameters</code>	Returns the parameters which were used to initialize the component
<code>save</code>	Saves component at file path
<code>transform</code>	Transforms input data by selecting features. If the component_obj does not have a transform method, will raise an <code>MethodPropertyNotFoundError</code> exception.

**clone** (*self*)

Constructs a new component with the same parameters and random state.

**Returns** A new instance of this component with identical parameters and random state.

**default\_parameters** (*cls*)

Returns the default parameters for this component.

Our convention is that `Component.default_parameters == Component().parameters`.

**Returns** default parameters for this component.

**Return type** dict

**describe** (*self*, *print\_name=False*, *return\_dict=False*)

Describe a component and its parameters

**Parameters**

- **print\_name** (*bool*, *optional*) – whether to print name of component
- **return\_dict** (*bool*, *optional*) – whether to return description as dictionary in the format {"name": name, "parameters": parameters}

**Returns** prints and returns dictionary

**Return type** None or dict

**fit** (*self*, *X*, *y=None*)

Fits component to data

**Parameters**

- **X** (*list*, *pd.DataFrame* or *np.ndarray*) – The input training data of shape [n\_samples, n\_features]
- **y** (*list*, *pd.Series*, *np.ndarray*, *optional*) – The target training data of length [n\_samples]

**Returns** self

**fit\_transform** (*self*, *X*, *y=None*)

Fits on X and transforms X

**Parameters**

- **X** (*pd.DataFrame*) – Data to fit and transform

- **y** (*pd.Series*) – Target data

**Returns** Transformed X

**Return type** *pd.DataFrame*

**get\_names** (*self*)

Get names of selected features.

**Returns** List of the names of features selected

**Return type** *list[str]*

**static load** (*file\_path*)

Loads component at file path

**Parameters** **file\_path** (*str*) – Location to load file

**Returns** *ComponentBase* object

**needs\_fitting** (*self*)

Returns boolean determining if component needs fitting before calling *predict*, *predict\_proba*, *transform*, or *feature\_importances*. This can be overridden to *False* for components that do not need to be fit or whose fit methods do nothing.

**property parameters** (*self*)

Returns the parameters which were used to initialize the component

**save** (*self, file\_path, pickle\_protocol=cloudpickle.DEFAULT\_PROTOCOL*)

Saves component at file path

**Parameters**

- **file\_path** (*str*) – Location to save file
- **pickle\_protocol** (*int*) – The pickle data stream format.

**Returns** *None*

**transform** (*self, X, y=None*)

Transforms input data by selecting features. If the component\_obj does not have a transform method, will raise an *MethodPropertyNotFoundError* exception.

**Parameters**

- **x** (*pd.DataFrame*) – Data to transform.
- **y** (*pd.Series, optional*) – Target data. Ignored.

**Returns** Transformed X

**Return type** *pd.DataFrame*

**class** *evalml.pipelines.components.SelectByType* (*column\_types=None, random\_seed=0, \*\*kwargs*)

Selects columns by specified Woodwork logical type or semantic tag in input data.

**Parameters**

- **column\_types** (*string, ww.LogicalType, list(string), list(ww.LogicalType)*) – List of Woodwork types or tags, used to determine which columns to select.
- **random\_seed** (*int*) – Seed for the random number generator. Defaults to 0.

**Attributes**

<b>hyper-parameter_ranges</b>	{}
<b>model_family</b>	ModelFamily.NONE
<b>modifies_features</b>	True
<b>modifies_target</b>	False
<b>name</b>	Select Columns By Type Transformer
<b>needs_fitting</b>	False

**Methods**

<i>clone</i>	Constructs a new component with the same parameters and random state.
<i>default_parameters</i>	Returns the default parameters for this component.
<i>describe</i>	Describe a component and its parameters
<i>fit</i>	Fits the transformer by checking if column names are present in the dataset.
<i>fit_transform</i>	Fits on X and transforms X
<i>load</i>	Loads component at file path
<i>parameters</i>	Returns the parameters which were used to initialize the component
<i>save</i>	Saves component at file path
<i>transform</i>	Transforms data X.

**clone** (*self*)

Constructs a new component with the same parameters and random state.

**Returns** A new instance of this component with identical parameters and random state.

**default\_parameters** (*cls*)

Returns the default parameters for this component.

Our convention is that `Component.default_parameters == Component().parameters`.

**Returns** default parameters for this component.

**Return type** dict

**describe** (*self*, *print\_name=False*, *return\_dict=False*)

Describe a component and its parameters

**Parameters**

- **print\_name** (*bool*, *optional*) – whether to print name of component
- **return\_dict** (*bool*, *optional*) – whether to return description as dictionary in the format {"name": name, "parameters": parameters}

**Returns** prints and returns dictionary

**Return type** None or dict

**fit** (*self*, *X*, *y=None*)

Fits the transformer by checking if column names are present in the dataset.

**Parameters**

- **x** (*pd.DataFrame*) – Data to check.
- **y** (*pd.Series, optional*) – Targets.

**Returns** self**fit\_transform** (*self, X, y=None*)

Fits on X and transforms X

**Parameters**

- **x** (*pd.DataFrame*) – Data to fit and transform
- **y** (*pd.Series*) – Target data

**Returns** Transformed X**Return type** *pd.DataFrame***static load** (*file\_path*)

Loads component at file path

**Parameters** **file\_path** (*str*) – Location to load file**Returns** ComponentBase object**property parameters** (*self*)

Returns the parameters which were used to initialize the component

**save** (*self, file\_path, pickle\_protocol=cloudpickle.DEFAULT\_PROTOCOL*)

Saves component at file path

**Parameters**

- **file\_path** (*str*) – Location to save file
- **pickle\_protocol** (*int*) – The pickle data stream format.

**Returns** None**transform** (*self, X, y=None*)

Transforms data X.

**Parameters**

- **x** (*pd.DataFrame*) – Data to transform.
- **y** (*pd.Series, optional*) – Target data.

**Returns** Transformed X**Return type** *pd.DataFrame***class** evalml.pipelines.components.**SelectColumns** (*columns=None, random\_seed=0, \*\*kwargs*)

Selects specified columns in input data.

**Parameters**

- **columns** (*list(string)*) – List of column names, used to determine which columns to select.
- **random\_seed** (*int*) – Seed for the random number generator. Defaults to 0.



**Attributes**

<b>hyper-parameter_ranges</b>	{}
<b>model_family</b>	ModelFamily.NONE
<b>modifies_features</b>	True
<b>modifies_target</b>	False
<b>name</b>	Select Columns Transformer
<b>needs_fitting</b>	False

**Methods**

<i>clone</i>	Constructs a new component with the same parameters and random state.
<i>default_parameters</i>	Returns the default parameters for this component.
<i>describe</i>	Describe a component and its parameters
<i>fit</i>	Fits the transformer by checking if column names are present in the dataset.
<i>fit_transform</i>	Fits on X and transforms X
<i>load</i>	Loads component at file path
<i>parameters</i>	Returns the parameters which were used to initialize the component
<i>save</i>	Saves component at file path
<i>transform</i>	Transforms data X by selecting columns.

**clone** (*self*)

Constructs a new component with the same parameters and random state.

**Returns** A new instance of this component with identical parameters and random state.

**default\_parameters** (*cls*)

Returns the default parameters for this component.

Our convention is that `Component.default_parameters == Component().parameters`.

**Returns** default parameters for this component.

**Return type** dict

**describe** (*self*, *print\_name=False*, *return\_dict=False*)

Describe a component and its parameters

**Parameters**

- **print\_name** (*bool*, *optional*) – whether to print name of component
- **return\_dict** (*bool*, *optional*) – whether to return description as dictionary in the format {"name": name, "parameters": parameters}

**Returns** prints and returns dictionary

**Return type** None or dict

**fit** (*self*, *X*, *y=None*)

Fits the transformer by checking if column names are present in the dataset.

**Parameters**

- **x** (*pd.DataFrame*) – Data to check.
- **y** (*pd.Series*, *optional*) – Targets.

**Returns** *self***fit\_transform** (*self*, *X*, *y=None*)

Fits on X and transforms X

**Parameters**

- **x** (*pd.DataFrame*) – Data to fit and transform
- **y** (*pd.Series*) – Target data

**Returns** Transformed X**Return type** *pd.DataFrame***static load** (*file\_path*)

Loads component at file path

**Parameters** **file\_path** (*str*) – Location to load file**Returns** ComponentBase object**property parameters** (*self*)

Returns the parameters which were used to initialize the component

**save** (*self*, *file\_path*, *pickle\_protocol=cloudpickle.DEFAULT\_PROTOCOL*)

Saves component at file path

**Parameters**

- **file\_path** (*str*) – Location to save file
- **pickle\_protocol** (*int*) – The pickle data stream format.

**Returns** *None***transform** (*self*, *X*, *y=None*)

Transforms data X by selecting columns.

**Parameters**

- **x** (*pd.DataFrame*) – Data to transform.
- **y** (*pd.Series*, *optional*) – Targets.

**Returns** Transformed X.**Return type** *pd.DataFrame*

```
class evalml.pipelines.components.SimpleImputer (impute_strategy='most_frequent',  
                                              fill_value=None, random_seed=0,  
                                              **kwargs)
```

Imputes missing data according to a specified imputation strategy.

**Parameters**

- **impute\_strategy** (*string*) – Impute strategy to use. Valid values include “mean”, “median”, “most\_frequent”, “constant” for numerical data, and “most\_frequent”, “constant” for object data types.

- **fill\_value** (*string*) – When `impute_strategy == “constant”`, `fill_value` is used to replace missing data. Defaults to 0 when imputing numerical data and “missing\_value” for strings or object data types.
- **random\_seed** (*int*) – Seed for the random number generator. Defaults to 0.

#### Attributes

<b>hyper-parameter_ranges</b>	{ “impute_strategy”: [“mean”, “median”, “most_frequent”]}
<b>model_family</b>	ModelFamily.NONE
<b>modifies_features</b>	True
<b>modifies_target</b>	False
<b>name</b>	Simple Imputer

#### Methods

<i>clone</i>	Constructs a new component with the same parameters and random state.
<i>default_parameters</i>	Returns the default parameters for this component.
<i>describe</i>	Describe a component and its parameters
<i>fit</i>	Fits imputer to data. ‘None’ values are converted to np.nan before imputation and are
<i>fit_transform</i>	Fits on X and transforms X
<i>load</i>	Loads component at file path
<i>needs_fitting</i>	Returns boolean determining if component needs fitting before
<i>parameters</i>	Returns the parameters which were used to initialize the component
<i>save</i>	Saves component at file path
<i>transform</i>	Transforms input by imputing missing values. ‘None’ and np.nan values are treated as the same.

#### **clone** (*self*)

Constructs a new component with the same parameters and random state.

**Returns** A new instance of this component with identical parameters and random state.

#### **default\_parameters** (*cls*)

Returns the default parameters for this component.

Our convention is that `Component.default_parameters == Component().parameters`.

**Returns** default parameters for this component.

**Return type** dict

#### **describe** (*self*, *print\_name=False*, *return\_dict=False*)

Describe a component and its parameters

#### Parameters

- **print\_name** (*bool*, *optional*) – whether to print name of component

- **return\_dict** (*bool, optional*) – whether to return description as dictionary in the format {"name": name, "parameters": parameters}

**Returns** prints and returns dictionary

**Return type** None or dict

**fit** (*self, X, y=None*)

**Fits imputer to data.** ‘None’ values are converted to np.nan before imputation and are treated as the same.

**Parameters**

- **X** (*pd.DataFrame or np.ndarray*) – the input training data of shape [n\_samples, n\_features]
- **y** (*pd.Series, optional*) – the target training data of length [n\_samples]

**Returns** self

**fit\_transform** (*self, X, y=None*)

Fits on X and transforms X

**Parameters**

- **X** (*pd.DataFrame*) – Data to fit and transform
- **y** (*pd.Series, optional*) – Target data.

**Returns** Transformed X

**Return type** pd.DataFrame

**static load** (*file\_path*)

Loads component at file path

**Parameters** **file\_path** (*str*) – Location to load file

**Returns** ComponentBase object

**needs\_fitting** (*self*)

Returns boolean determining if component needs fitting before calling predict, predict\_proba, transform, or feature\_importances. This can be overridden to False for components that do not need to be fit or whose fit methods do nothing.

**property parameters** (*self*)

Returns the parameters which were used to initialize the component

**save** (*self, file\_path, pickle\_protocol=cloudpickle.DEFAULT\_PROTOCOL*)

Saves component at file path

**Parameters**

- **file\_path** (*str*) – Location to save file
- **pickle\_protocol** (*int*) – The pickle data stream format.

**Returns** None

**transform** (*self, X, y=None*)

Transforms input by imputing missing values. ‘None’ and np.nan values are treated as the same.

**Parameters**

- **X** (*pd.DataFrame*) – Data to transform

- **y** (*pd.Series, optional*) – Ignored.

**Returns** Transformed X

**Return type** `pd.DataFrame`

```
class evalml.pipelines.components.SklearnStackedEnsembleClassifier (input_pipelines=None,
                                                                    fi-
                                                                    nal_estimator=None,
                                                                    cv=None,
                                                                    n_jobs=-
                                                                    1, random-
                                                                    dom_seed=0,
                                                                    **kwargs)
```

Scikit-learn Stacked Ensemble Classifier.

#### Parameters

- **input\_pipelines** (*list(PipelineBase or subclass obj)*) – List of pipeline instances to use as the base estimators. This must not be None or an empty list or else `EnsembleMissingPipelinesError` will be raised.
- **final\_estimator** (*Estimator or subclass*) – The classifier used to combine the base estimators. If None, uses `LogisticRegressionClassifier`.
- **cv** (*int, cross-validation generator or an iterable*) – Determines the cross-validation splitting strategy used to train `final_estimator`. For `int/None` inputs, if the estimator is a classifier and `y` is either binary or multiclass, `StratifiedKFold` is used. Defaults to None. Possible inputs for `cv` are:
  - None: 3-fold cross validation
  - int: the number of folds in a (Stratified) KFold
  - An scikit-learn cross-validation generator object
  - An iterable yielding (train, test) splits
- **n\_jobs** (*int or None*) – Non-negative integer describing level of parallelism used for pipelines. None and 1 are equivalent. If set to -1, all CPUs are used. For `n_jobs` below -1, `(n_cpus + 1 + n_jobs)` are used. Defaults to -1. - Note: there could be some multi-process errors thrown for values of `n_jobs != 1`. If this is the case, please use `n_jobs = 1`.
- **random\_seed** (*int*) – Seed for the random number generator. Defaults to 0.

#### Attributes

<b>hyper-parameter_ranges</b>	{}
<b>model_family</b>	ModelFamily.ENSEMBLE
<b>modifies_features</b>	True
<b>modifies_target</b>	False
<b>name</b>	Sklearn Stacked Ensemble Classifier
<b>pre-dict_uses_y</b>	False
<b>supported_problem_types</b>	[ ProblemTypes.BINARY, ProblemTypes.MULTICLASS, ProblemTypes.TIME_SERIES_BINARY, ProblemTypes.TIME_SERIES_MULTICLASS, ]

**Methods**

<code>clone</code>	Constructs a new component with the same parameters and random state.
<code>default_parameters</code>	Returns the default parameters for stacked ensemble classes.
<code>describe</code>	Describe a component and its parameters
<code>feature_importance</code>	Not implemented for SklearnStackedEnsembleClassifier and SklearnStackedEnsembleRegressor
<code>fit</code>	Fits component to data
<code>load</code>	Loads component at file path
<code>needs_fitting</code>	Returns boolean determining if component needs fitting before
<code>parameters</code>	Returns the parameters which were used to initialize the component
<code>predict</code>	Make predictions using selected features.
<code>predict_proba</code>	Make probability estimates for labels.
<code>save</code>	Saves component at file path

**clone** (*self*)

Constructs a new component with the same parameters and random state.

**Returns** A new instance of this component with identical parameters and random state.

**default\_parameters** (*cls*)

Returns the default parameters for stacked ensemble classes.

**Returns** default parameters for this component.

**Return type** dict

**describe** (*self*, *print\_name=False*, *return\_dict=False*)

Describe a component and its parameters

**Parameters**

- **print\_name** (*bool*, *optional*) – whether to print name of component
- **return\_dict** (*bool*, *optional*) – whether to return description as dictionary in the format {"name": name, "parameters": parameters}

**Returns** prints and returns dictionary

**Return type** None or dict

**property feature\_importance** (*self*)

Not implemented for SklearnStackedEnsembleClassifier and SklearnStackedEnsembleRegressor

**fit** (*self*, *X*, *y=None*)

Fits component to data

**Parameters**

- **X** (*list*, *pd.DataFrame* or *np.ndarray*) – The input training data of shape [n\_samples, n\_features]
- **y** (*list*, *pd.Series*, *np.ndarray*, *optional*) – The target training data of length [n\_samples]

**Returns** self

**static load** (*file\_path*)

Loads component at file path

**Parameters** *file\_path* (*str*) – Location to load file

**Returns** ComponentBase object

**needs\_fitting** (*self*)

Returns boolean determining if component needs fitting before calling predict, predict\_proba, transform, or feature\_importances. This can be overridden to False for components that do not need to be fit or whose fit methods do nothing.

**property parameters** (*self*)

Returns the parameters which were used to initialize the component

**predict** (*self*, *X*)

Make predictions using selected features.

**Parameters** *X* (*pd.DataFrame*, *np.ndarray*) – Data of shape [n\_samples, n\_features]

**Returns** Predicted values

**Return type** *pd.Series*

**predict\_proba** (*self*, *X*)

Make probability estimates for labels.

**Parameters** *X* (*pd.DataFrame*, *or np.ndarray*) – Features

**Returns** Probability estimates

**Return type** *pd.Series*

**save** (*self*, *file\_path*, *pickle\_protocol=cloudpickle.DEFAULT\_PROTOCOL*)

Saves component at file path

**Parameters**

- **file\_path** (*str*) – Location to save file
- **pickle\_protocol** (*int*) – The pickle data stream format.

**Returns** None

```
class evalml.pipelines.components.SklearnStackedEnsembleRegressor (input_pipelines=None,
                                                                    fi-
                                                                    nal_estimator=None,
                                                                    cv=None,
                                                                    n_jobs=-
                                                                    1, ran-
                                                                    dom_seed=0,
                                                                    **kwargs)
```

Scikit-learn Stacked Ensemble Regressor.

**Parameters**

- **input\_pipelines** (*list (PipelineBase or subclass obj)*) – List of pipeline instances to use as the base estimators. This must not be None or an empty list or else EnsembleMissingPipelinesError will be raised.
- **final\_estimator** (*Estimator or subclass*) – The regressor used to combine the base estimators. If None, uses LinearRegressor.

- **cv** (*int, cross-validation generator or an iterable*) – Determines the cross-validation splitting strategy used to train `final_estimator`. For `int/None` inputs, `KFold` is used. Defaults to `None`. Possible inputs for `cv` are:
  - `None`: 3-fold cross validation
  - `int`: the number of folds in a (Stratified) `KFold`
  - An scikit-learn cross-validation generator object
  - An iterable yielding (train, test) splits
- **n\_jobs** (*int or None*) – Non-negative integer describing level of parallelism used for pipelines. `None` and `1` are equivalent. If set to `-1`, all CPUs are used. For `n_jobs` below `-1`, `(n_cpus + 1 + n_jobs)` are used. Defaults to `-1`. - Note: there could be some multi-process errors thrown for values of `n_jobs != 1`. If this is the case, please use `n_jobs = 1`.
- **random\_seed** (*int*) – Seed for the random number generator. Defaults to `0`.

#### Attributes

<b>hyper-parameter_ranges</b>	{}
<b>model_family</b>	ModelFamily.ENSEMBLE
<b>modifies_features</b>	True
<b>modifies_target</b>	False
<b>name</b>	Sklearn Stacked Ensemble Regressor
<b>predict_uses_y</b>	False
<b>supported_problem_types</b>	[ ProblemTypes.REGRESSION, ProblemTypes.TIME_SERIES_REGRESSION,]

#### Methods

<code>clone</code>	Constructs a new component with the same parameters and random state.
<code>default_parameters</code>	Returns the default parameters for stacked ensemble classes.
<code>describe</code>	Describe a component and its parameters
<code>feature_importance</code>	Not implemented for <code>SklearnStackedEnsembleClassifier</code> and <code>SklearnStackedEnsembleRegressor</code>
<code>fit</code>	Fits component to data
<code>load</code>	Loads component at file path
<code>needs_fitting</code>	Returns boolean determining if component needs fitting before
<code>parameters</code>	Returns the parameters which were used to initialize the component
<code>predict</code>	Make predictions using selected features.
<code>predict_proba</code>	Make probability estimates for labels.
<code>save</code>	Saves component at file path

#### **clone** (*self*)

Constructs a new component with the same parameters and random state.



**Returns** A new instance of this component with identical parameters and random state.

**default\_parameters** (*cls*)

Returns the default parameters for stacked ensemble classes.

**Returns** default parameters for this component.

**Return type** dict

**describe** (*self*, *print\_name=False*, *return\_dict=False*)

Describe a component and its parameters

**Parameters**

- **print\_name** (*bool*, *optional*) – whether to print name of component
- **return\_dict** (*bool*, *optional*) – whether to return description as dictionary in the format {"name": name, "parameters": parameters}

**Returns** prints and returns dictionary

**Return type** None or dict

**property feature\_importance** (*self*)

Not implemented for SklearnStackedEnsembleClassifier and SklearnStackedEnsembleRegressor

**fit** (*self*, *X*, *y=None*)

Fits component to data

**Parameters**

- **X** (*list*, *pd.DataFrame* or *np.ndarray*) – The input training data of shape [n\_samples, n\_features]
- **y** (*list*, *pd.Series*, *np.ndarray*, *optional*) – The target training data of length [n\_samples]

**Returns** self

**static load** (*file\_path*)

Loads component at file path

**Parameters** **file\_path** (*str*) – Location to load file

**Returns** ComponentBase object

**needs\_fitting** (*self*)

Returns boolean determining if component needs fitting before calling predict, predict\_proba, transform, or feature\_importances. This can be overridden to False for components that do not need to be fit or whose fit methods do nothing.

**property parameters** (*self*)

Returns the parameters which were used to initialize the component

**predict** (*self*, *X*)

Make predictions using selected features.

**Parameters** **X** (*pd.DataFrame*, or *np.ndarray*) – Data of shape [n\_samples, n\_features]

**Returns** Predicted values

**Return type** pd.Series

**predict\_proba** (*self*, *X*)

Make probability estimates for labels.

**Parameters** **X** (*pd.DataFrame*, or *np.ndarray*) – Features

**Returns** Probability estimates

**Return type** `pd.Series`

**save** (*self*, *file\_path*, *pickle\_protocol=cloudpickle.DEFAULT\_PROTOCOL*)

Saves component at file path

**Parameters**

- **file\_path** (*str*) – Location to save file
- **pickle\_protocol** (*int*) – The pickle data stream format.

**Returns** None

```
class evalml.pipelines.components.SMOTENCOversampler (sampling_ratio=0.25,
                                                    k_neighbors_default=5,
                                                    n_jobs=- 1, random_seed=0,
                                                    **kwargs)
```

SMOTENC Oversampler component. Uses SMOTENC to generate synthetic samples. Works on a mix of numerical and categorical columns. Input data must be Woodwork type, and this component is only run during training and not during predict.

**Parameters**

- **sampling\_ratio** (*float*) – This is the goal ratio of the minority to majority class, with range (0, 1]. A value of 0.25 means we want a 1:4 ratio of the minority to majority class after oversampling. We will create the a sampling dictionary using this ratio, with the keys corresponding to the class and the values responding to the number of samples. Defaults to 0.25.
- **k\_neighbors\_default** (*int*) – The number of nearest neighbors used to construct synthetic samples. This is the default value used, but the actual `k_neighbors` value might be smaller if there are less samples. Defaults to 5.
- **n\_jobs** (*int*) – The number of CPU cores to use. Defaults to -1.
- **random\_seed** (*int*) – The seed to use for random sampling. Defaults to 0.

**Attributes**

<b>hyper-parameter_ranges</b>	{}
<b>model_family</b>	ModelFamily.NONE
<b>modifies_features</b>	True
<b>modifies_target</b>	True
<b>name</b>	SMOTENC Oversampler

**Methods**

<code>clone</code>	Constructs a new component with the same parameters and random state.
<code>default_parameters</code>	Returns the default parameters for this component.
<code>describe</code>	Describe a component and its parameters
<code>fit</code>	Fits the sampler to the data.
<code>fit_transform</code>	Fits on X and transforms X

continues on next page

Table 587 – continued from previous page

<i>load</i>	Loads component at file path
<i>needs_fitting</i>	Returns boolean determining if component needs fitting before
<i>parameters</i>	Returns the parameters which were used to initialize the component
<i>save</i>	Saves component at file path
<i>transform</i>	Transforms the input data by sampling the data.

**clone** (*self*)

Constructs a new component with the same parameters and random state.

**Returns** A new instance of this component with identical parameters and random state.

**default\_parameters** (*cls*)

Returns the default parameters for this component.

Our convention is that `Component.default_parameters == Component().parameters`.

**Returns** default parameters for this component.

**Return type** dict

**describe** (*self*, *print\_name=False*, *return\_dict=False*)

Describe a component and its parameters

**Parameters**

- **print\_name** (*bool*, *optional*) – whether to print name of component
- **return\_dict** (*bool*, *optional*) – whether to return description as dictionary in the format {"name": name, "parameters": parameters}

**Returns** prints and returns dictionary

**Return type** None or dict

**fit** (*self*, *X*, *y*)

Fits the sampler to the data.

**Parameters**

- **X** (*pd.DataFrame*) – Input features.
- **y** (*pd.Series*) – Target.

**Returns** self

**fit\_transform** (*self*, *X*, *y*)

Fits on X and transforms X

**Parameters**

- **X** (*pd.DataFrame*) – Data to fit and transform
- **y** (*pd.Series*) – Target data

**Returns** Transformed X

**Return type** pd.DataFrame

**static load** (*file\_path*)

Loads component at file path

**Parameters** **file\_path** (*str*) – Location to load file

**Returns** ComponentBase object

**needs\_fitting** (*self*)

Returns boolean determining if component needs fitting before calling predict, predict\_proba, transform, or feature\_importances. This can be overridden to False for components that do not need to be fit or whose fit methods do nothing.

**property parameters** (*self*)

Returns the parameters which were used to initialize the component

**save** (*self*, *file\_path*, *pickle\_protocol*=cloudpickle.DEFAULT\_PROTOCOL)

Saves component at file path

**Parameters**

- **file\_path** (*str*) – Location to save file
- **pickle\_protocol** (*int*) – The pickle data stream format.

**Returns** None

**transform** (*self*, *X*, *y*=None)

Transforms the input data by sampling the data.

**Parameters**

- **X** (*pd.DataFrame*) – Training features.
- **y** (*pd.Series*) – Target.

**Returns** Transformed features and target.

**Return type** pd.DataFrame, pd.Series

```
class evalml.pipelines.components.SMOTENOverSampler (sampling_ratio=0.25,  
                                                    k_neighbors_default=5,  
                                                    n_jobs=- 1, random_seed=0,  
                                                    **kwargs)
```

SMOTEN Oversampler component. Uses SMOTEN to generate synthetic samples. Works for purely categorical datasets. This component is only run during training and not during predict.

**Parameters**

- **sampling\_ratio** (*float*) – This is the goal ratio of the minority to majority class, with range (0, 1]. A value of 0.25 means we want a 1:4 ratio of the minority to majority class after oversampling. We will create the a sampling dictionary using this ratio, with the keys corresponding to the class and the values responding to the number of samples. Defaults to 0.25.
- **k\_neighbors\_default** (*int*) – The number of nearest neighbors used to construct synthetic samples. This is the default value used, but the actual k\_neighbors value might be smaller if there are less samples. Defaults to 5.
- **n\_jobs** (*int*) – The number of CPU cores to use. Defaults to -1.
- **random\_seed** (*int*) – The seed to use for random sampling. Defaults to 0.

**Attributes**

<b>hyper-parameter_ranges</b>	{}
<b>model_family</b>	ModelFamily.NONE
<b>modifies_features</b>	True
<b>modifies_target</b>	True
<b>name</b>	SMOTEN Oversampler

## Methods

<i>clone</i>	Constructs a new component with the same parameters and random state.
<i>default_parameters</i>	Returns the default parameters for this component.
<i>describe</i>	Describe a component and its parameters
<i>fit</i>	Fits the sampler to the data.
<i>fit_transform</i>	Fits on X and transforms X
<i>load</i>	Loads component at file path
<i>needs_fitting</i>	Returns boolean determining if component needs fitting before
<i>parameters</i>	Returns the parameters which were used to initialize the component
<i>save</i>	Saves component at file path
<i>transform</i>	Transforms the input data by sampling the data.

### **clone** (*self*)

Constructs a new component with the same parameters and random state.

**Returns** A new instance of this component with identical parameters and random state.

### **default\_parameters** (*cls*)

Returns the default parameters for this component.

Our convention is that `Component.default_parameters == Component().parameters`.

**Returns** default parameters for this component.

**Return type** dict

### **describe** (*self*, *print\_name=False*, *return\_dict=False*)

Describe a component and its parameters

#### Parameters

- **print\_name** (*bool*, *optional*) – whether to print name of component
- **return\_dict** (*bool*, *optional*) – whether to return description as dictionary in the format {"name": name, "parameters": parameters}

**Returns** prints and returns dictionary

**Return type** None or dict

### **fit** (*self*, *X*, *y*)

Fits the sampler to the data.

#### Parameters

- **X** (*pd.DataFrame*) – Input features.
- **y** (*pd.Series*) – Target.

**Returns** self

**fit\_transform** (*self*, *X*, *y*)  
Fits on X and transforms X

**Parameters**

- **X** (*pd.DataFrame*) – Data to fit and transform
- **y** (*pd.Series*) – Target data

**Returns** Transformed X

**Return type** *pd.DataFrame*

**static load** (*file\_path*)  
Loads component at file path

**Parameters** **file\_path** (*str*) – Location to load file

**Returns** ComponentBase object

**needs\_fitting** (*self*)  
Returns boolean determining if component needs fitting before calling predict, predict\_proba, transform, or feature\_importances. This can be overridden to False for components that do not need to be fit or whose fit methods do nothing.

**property parameters** (*self*)  
Returns the parameters which were used to initialize the component

**save** (*self*, *file\_path*, *pickle\_protocol=cloudpickle.DEFAULT\_PROTOCOL*)  
Saves component at file path

**Parameters**

- **file\_path** (*str*) – Location to save file
- **pickle\_protocol** (*int*) – The pickle data stream format.

**Returns** None

**transform** (*self*, *X*, *y=None*)  
Transforms the input data by sampling the data.

**Parameters**

- **X** (*pd.DataFrame*) – Training features.
- **y** (*pd.Series*) – Target.

**Returns** Transformed features and target.

**Return type** *pd.DataFrame*, *pd.Series*

**class** evalml.pipelines.components.**SMOTEOverSampler** (*sampling\_ratio=0.25*,  
*k\_neighbors\_default=5*, *n\_jobs=-1*, *random\_seed=0*, *\*\*kwargs*)

SMOTE Oversampler component. Works on numerical datasets only. This component is only run during training and not during predict.

**Parameters**

- **sampling\_ratio** (*float*) – This is the goal ratio of the minority to majority class, with range (0, 1]. A value of 0.25 means we want a 1:4 ratio of the minority to majority class after oversampling. We will create the a sampling dictionary using this ratio, with the keys corresponding to the class and the values responding to the number of samples. Defaults to 0.25.
- **k\_neighbors\_default** (*int*) – The number of nearest neighbors used to construct synthetic samples. This is the default value used, but the actual k\_neighbors value might be smaller if there are less samples. Defaults to 5.
- **n\_jobs** (*int*) – The number of CPU cores to use. Defaults to -1.
- **random\_seed** (*int*) – The seed to use for random sampling. Defaults to 0.

#### Attributes

<b>hyper-parameter_ranges</b>	{}
<b>model_family</b>	ModelFamily.NONE
<b>modifies_features</b>	True
<b>modifies_target</b>	True
<b>name</b>	SMOTE Oversampler

#### Methods

<i>clone</i>	Constructs a new component with the same parameters and random state.
<i>default_parameters</i>	Returns the default parameters for this component.
<i>describe</i>	Describe a component and its parameters
<i>fit</i>	Fits the sampler to the data.
<i>fit_transform</i>	Fits on X and transforms X
<i>load</i>	Loads component at file path
<i>needs_fitting</i>	Returns boolean determining if component needs fitting before
<i>parameters</i>	Returns the parameters which were used to initialize the component
<i>save</i>	Saves component at file path
<i>transform</i>	Transforms the input data by sampling the data.

#### **clone** (*self*)

Constructs a new component with the same parameters and random state.

**Returns** A new instance of this component with identical parameters and random state.

#### **default\_parameters** (*cls*)

Returns the default parameters for this component.

Our convention is that `Component.default_parameters == Component().parameters`.

**Returns** default parameters for this component.

**Return type** dict

#### **describe** (*self*, *print\_name=False*, *return\_dict=False*)

Describe a component and its parameters

**Parameters**

- **print\_name** (*bool*, *optional*) – whether to print name of component
- **return\_dict** (*bool*, *optional*) – whether to return description as dictionary in the format {"name": name, "parameters": parameters}

**Returns** prints and returns dictionary

**Return type** None or dict

**fit** (*self*, *X*, *y*)

Fits the sampler to the data.

**Parameters**

- **X** (*pd.DataFrame*) – Input features.
- **y** (*pd.Series*) – Target.

**Returns** self

**fit\_transform** (*self*, *X*, *y*)

Fits on X and transforms X

**Parameters**

- **X** (*pd.DataFrame*) – Data to fit and transform
- **y** (*pd.Series*) – Target data

**Returns** Transformed X

**Return type** pd.DataFrame

**static load** (*file\_path*)

Loads component at file path

**Parameters** **file\_path** (*str*) – Location to load file

**Returns** ComponentBase object

**needs\_fitting** (*self*)

Returns boolean determining if component needs fitting before calling predict, predict\_proba, transform, or feature\_importances. This can be overridden to False for components that do not need to be fit or whose fit methods do nothing.

**property parameters** (*self*)

Returns the parameters which were used to initialize the component

**save** (*self*, *file\_path*, *pickle\_protocol=cloudpickle.DEFAULT\_PROTOCOL*)

Saves component at file path

**Parameters**

- **file\_path** (*str*) – Location to save file
- **pickle\_protocol** (*int*) – The pickle data stream format.

**Returns** None

**transform** (*self*, *X*, *y=None*)

Transforms the input data by sampling the data.

**Parameters**



- **X** (*pd.DataFrame*) – Training features.
- **y** (*pd.Series*) – Target.

**Returns** Transformed features and target.

**Return type** *pd.DataFrame*, *pd.Series*

**class** `evalml.pipelines.components.StandardScaler` (*random\_seed=0, \*\*kwargs*)  
A transformer that standardizes input features by removing the mean and scaling to unit variance.

**Parameters** `random_seed` (*int*) – Seed for the random number generator. Defaults to 0.

#### Attributes

<b>hyper-parameter_ranges</b>	{}
<b>model_family</b>	ModelFamily.NONE
<b>modifies_features</b>	True
<b>modifies_target</b>	False
<b>name</b>	Standard Scaler

#### Methods

<code>clone</code>	Constructs a new component with the same parameters and random state.
<code>default_parameters</code>	Returns the default parameters for this component.
<code>describe</code>	Describe a component and its parameters
<code>fit</code>	Fits component to data
<code>fit_transform</code>	Fits on X and transforms X
<code>load</code>	Loads component at file path
<code>needs_fitting</code>	Returns boolean determining if component needs fitting before
<code>parameters</code>	Returns the parameters which were used to initialize the component
<code>save</code>	Saves component at file path
<code>transform</code>	Transforms data X.

**clone** (*self*)

Constructs a new component with the same parameters and random state.

**Returns** A new instance of this component with identical parameters and random state.

**default\_parameters** (*cls*)

Returns the default parameters for this component.

Our convention is that `Component.default_parameters == Component().parameters`.

**Returns** default parameters for this component.

**Return type** *dict*

**describe** (*self*, *print\_name=False*, *return\_dict=False*)

Describe a component and its parameters

**Parameters**

- **print\_name** (*bool, optional*) – whether to print name of component
- **return\_dict** (*bool, optional*) – whether to return description as dictionary in the format {"name": name, "parameters": parameters}

**Returns** prints and returns dictionary

**Return type** None or dict

**fit** (*self, X, y=None*)

Fits component to data

**Parameters**

- **X** (*list, pd.DataFrame or np.ndarray*) – The input training data of shape [n\_samples, n\_features]
- **y** (*list, pd.Series, np.ndarray, optional*) – The target training data of length [n\_samples]

**Returns** self

**fit\_transform** (*self, X, y=None*)

Fits on X and transforms X

**Parameters**

- **X** (*pd.DataFrame*) – Data to fit and transform
- **y** (*pd.Series*) – Target data

**Returns** Transformed X

**Return type** pd.DataFrame

**static load** (*file\_path*)

Loads component at file path

**Parameters** **file\_path** (*str*) – Location to load file

**Returns** ComponentBase object

**needs\_fitting** (*self*)

Returns boolean determining if component needs fitting before calling predict, predict\_proba, transform, or feature\_importances. This can be overridden to False for components that do not need to be fit or whose fit methods do nothing.

**property parameters** (*self*)

Returns the parameters which were used to initialize the component

**save** (*self, file\_path, pickle\_protocol=cloudpickle.DEFAULT\_PROTOCOL*)

Saves component at file path

**Parameters**

- **file\_path** (*str*) – Location to save file
- **pickle\_protocol** (*int*) – The pickle data stream format.

**Returns** None

**transform** (*self, X, y=None*)

Transforms data X.

**Parameters**

- **X** (*pd.DataFrame*) – Data to transform.

- **y** (*pd.Series, optional*) – Target data.

**Returns** Transformed X

**Return type** `pd.DataFrame`

```
class evalml.pipelines.components.SVMClassifier(C=1.0, kernel='rbf', gamma='scale',
                                              probability=True, random_seed=0,
                                              **kwargs)
```

Support Vector Machine Classifier.

#### Parameters

- **C** (*float*) – The regularization parameter. The strength of the regularization is inversely proportional to C. Must be strictly positive. The penalty is a squared l2 penalty. Defaults to 1.0.
- **kernel** (*{ "linear", "poly", "rbf", "sigmoid", "precomputed" }*) – Specifies the kernel type to be used in the algorithm. Defaults to “rbf”.
- **gamma** (*{ "scale", "auto" } or float*) – Kernel coefficient for “rbf”, “poly” and “sigmoid”. Defaults to “scale”. - If gamma=“scale” (default) is passed then it uses  $1 / (n\_features * X.var())$  as value of gamma - if “auto”, uses  $1 / n\_features$
- **probability** (*boolean*) – Whether to enable probability estimates. Defaults to True.
- **random\_seed** (*int*) – Seed for the random number generator. Defaults to 0.

#### Attributes

<b>hyper-parameter_ranges</b>	{ “C”: Real(0, 10), “kernel”: [“linear”, “poly”, “rbf”, “sigmoid”, “precomputed”], “gamma”: [“scale”, “auto”], }
<b>model_family</b>	ModelFamily.SVM
<b>modifies_features</b>	True
<b>modifies_target</b>	False
<b>name</b>	SVM Classifier
<b>pre-dict_uses_y</b>	False
<b>supported_problem_types</b>	[ ProblemTypes.BINARY, ProblemTypes.MULTICLASS, ProblemTypes.TIME_SERIES_BINARY, ProblemTypes.TIME_SERIES_MULTICLASS, ]

#### Methods

<code>clone</code>	Constructs a new component with the same parameters and random state.
<code>default_parameters</code>	Returns the default parameters for this component.
<code>describe</code>	Describe a component and its parameters
<code>feature_importance</code>	Feature importance only works with linear kernels.
<code>fit</code>	Fits component to data
<code>load</code>	Loads component at file path
<code>needs_fitting</code>	Returns boolean determining if component needs fitting before
<code>parameters</code>	Returns the parameters which were used to initialize the component

continues on next page

Table 591 – continued from previous page

<code>predict</code>	Make predictions using selected features.
<code>predict_proba</code>	Make probability estimates for labels.
<code>save</code>	Saves component at file path

**clone** (*self*)

Constructs a new component with the same parameters and random state.

**Returns** A new instance of this component with identical parameters and random state.

**default\_parameters** (*cls*)

Returns the default parameters for this component.

Our convention is that `Component.default_parameters == Component().parameters`.

**Returns** default parameters for this component.

**Return type** dict

**describe** (*self*, *print\_name=False*, *return\_dict=False*)

Describe a component and its parameters

**Parameters**

- **print\_name** (*bool*, *optional*) – whether to print name of component
- **return\_dict** (*bool*, *optional*) – whether to return description as dictionary in the format {"name": name, "parameters": parameters}

**Returns** prints and returns dictionary

**Return type** None or dict

**property feature\_importance** (*self*)

Feature importance only works with linear kernels. If the kernel isn't linear, we return a numpy array of zeros

**fit** (*self*, *X*, *y=None*)

Fits component to data

**Parameters**

- **X** (*list*, *pd.DataFrame* or *np.ndarray*) – The input training data of shape [n\_samples, n\_features]
- **y** (*list*, *pd.Series*, *np.ndarray*, *optional*) – The target training data of length [n\_samples]

**Returns** self

**static load** (*file\_path*)

Loads component at file path

**Parameters** **file\_path** (*str*) – Location to load file

**Returns** ComponentBase object

**needs\_fitting** (*self*)

Returns boolean determining if component needs fitting before calling predict, predict\_proba, transform, or feature\_importances. This can be overridden to False for components that do not need to be fit or whose fit methods do nothing.

**property parameters** (*self*)

Returns the parameters which were used to initialize the component

**predict** (*self*, *X*)

Make predictions using selected features.

**Parameters** *X* (*pd.DataFrame*, *np.ndarray*) – Data of shape [n\_samples, n\_features]

**Returns** Predicted values

**Return type** *pd.Series*

**predict\_proba** (*self*, *X*)

Make probability estimates for labels.

**Parameters** *X* (*pd.DataFrame*, *or np.ndarray*) – Features

**Returns** Probability estimates

**Return type** *pd.Series*

**save** (*self*, *file\_path*, *pickle\_protocol=cloudpickle.DEFAULT\_PROTOCOL*)

Saves component at file path

**Parameters**

- **file\_path** (*str*) – Location to save file
- **pickle\_protocol** (*int*) – The pickle data stream format.

**Returns** None

**class** evalml.pipelines.components.**SVMRegressor** (*C=1.0*, *kernel='rbf'*, *gamma='scale'*, *random\_seed=0*, *\*\*kwargs*)

Support Vector Machine Regressor.

**Parameters**

- **C** (*float*) – The regularization parameter. The strength of the regularization is inversely proportional to C. Must be strictly positive. The penalty is a squared l2 penalty. Defaults to 1.0.
- **kernel** ({*"linear"*, *"poly"*, *"rbf"*, *"sigmoid"*, *"precomputed"*}) – Specifies the kernel type to be used in the algorithm. Defaults to “rbf”.
- **gamma** ({*"scale"*, *"auto"*} *or float*) – Kernel coefficient for “rbf”, “poly” and “sigmoid”. Defaults to “scale”. - If gamma=’scale’ (default) is passed then it uses  $1 / (n\_features * X.var())$  as value of gamma - if “auto”, uses  $1 / n\_features$
- **random\_seed** (*int*) – Seed for the random number generator. Defaults to 0.

**Attributes**

<b>hyper-parameter_ranges</b>	{ “C”: Real(0, 10), “kernel”: [“linear”, “poly”, “rbf”, “sigmoid”, “precomputed”], “gamma”: [“scale”, “auto”], }
<b>model_family</b>	ModelFamily.SVM
<b>modifies_features</b>	True
<b>modifies_target</b>	False
<b>name</b>	SVM Regressor
<b>predict_uses_y</b>	False
<b>supported_problem_types</b>	[ ProblemTypes.REGRESSION, ProblemTypes.TIME_SERIES_REGRESSION,]

## Methods

<code>clone</code>	Constructs a new component with the same parameters and random state.
<code>default_parameters</code>	Returns the default parameters for this component.
<code>describe</code>	Describe a component and its parameters
<code>feature_importance</code>	Feature importance only works with linear kernels.
<code>fit</code>	Fits component to data
<code>load</code>	Loads component at file path
<code>needs_fitting</code>	Returns boolean determining if component needs fitting before
<code>parameters</code>	Returns the parameters which were used to initialize the component
<code>predict</code>	Make predictions using selected features.
<code>predict_proba</code>	Make probability estimates for labels.
<code>save</code>	Saves component at file path

### **clone** (*self*)

Constructs a new component with the same parameters and random state.

**Returns** A new instance of this component with identical parameters and random state.

### **default\_parameters** (*cls*)

Returns the default parameters for this component.

Our convention is that `Component.default_parameters == Component().parameters`.

**Returns** default parameters for this component.

**Return type** dict

### **describe** (*self*, *print\_name=False*, *return\_dict=False*)

Describe a component and its parameters

#### Parameters

- **print\_name** (*bool*, *optional*) – whether to print name of component
- **return\_dict** (*bool*, *optional*) – whether to return description as dictionary in the format {"name": name, "parameters": parameters}

**Returns** prints and returns dictionary

**Return type** None or dict

### **property feature\_importance** (*self*)

Feature importance only works with linear kernels. If the kernel isn't linear, we return a numpy array of zeros

### **fit** (*self*, *X*, *y=None*)

Fits component to data

#### Parameters

- **X** (*list*, *pd.DataFrame* or *np.ndarray*) – The input training data of shape [n\_samples, n\_features]
- **y** (*list*, *pd.Series*, *np.ndarray*, *optional*) – The target training data of length [n\_samples]

**Returns** self

**static load** (*file\_path*)

Loads component at file path

**Parameters** *file\_path* (*str*) – Location to load file

**Returns** ComponentBase object

**needs\_fitting** (*self*)

Returns boolean determining if component needs fitting before calling predict, predict\_proba, transform, or feature\_importances. This can be overridden to False for components that do not need to be fit or whose fit methods do nothing.

**property parameters** (*self*)

Returns the parameters which were used to initialize the component

**predict** (*self*, *X*)

Make predictions using selected features.

**Parameters** *X* (*pd.DataFrame*, *np.ndarray*) – Data of shape [n\_samples, n\_features]

**Returns** Predicted values

**Return type** *pd.Series*

**predict\_proba** (*self*, *X*)

Make probability estimates for labels.

**Parameters** *X* (*pd.DataFrame*, or *np.ndarray*) – Features

**Returns** Probability estimates

**Return type** *pd.Series*

**save** (*self*, *file\_path*, *pickle\_protocol=cloudpickle.DEFAULT\_PROTOCOL*)

Saves component at file path

**Parameters**

- **file\_path** (*str*) – Location to save file
- **pickle\_protocol** (*int*) – The pickle data stream format.

**Returns** None

```
class evalml.pipelines.components.TargetEncoder (cols=None, smoothing=1.0,
                                                handle_unknown='value', handle_missing='value', random_seed=0,
                                                **kwargs)
```

A transformer that encodes categorical features into target encodings.

**Parameters**

- **cols** (*list*) – Columns to encode. If None, all string columns will be encoded, otherwise only the columns provided will be encoded. Defaults to None
- **smoothing** (*float*) – The smoothing factor to apply. The larger this value is, the more influence the expected target value has on the resulting target encodings. Must be strictly larger than 0. Defaults to 1.0
- **handle\_unknown** (*string*) – Determines how to handle unknown categories for a feature encountered. Options are 'value', 'error', and 'return\_nan'. Defaults to 'value', which replaces with the target mean
- **handle\_missing** (*string*) – Determines how to handle missing values encountered during *fit* or *transform*. Options are 'value', 'error', and 'return\_nan'. Defaults to 'value', which replaces with the target mean

- **random\_seed** (*int*) – Seed for the random number generator. Defaults to 0.

**Attributes**

<b>hyper-parameter_ranges</b>	{}
<b>model_family</b>	ModelFamily.NONE
<b>modifies_features</b>	True
<b>modifies_target</b>	False
<b>name</b>	Target Encoder

**Methods**

<i>clone</i>	Constructs a new component with the same parameters and random state.
<i>default_parameters</i>	Returns the default parameters for this component.
<i>describe</i>	Describe a component and its parameters
<i>fit</i>	Fits component to data
<i>fit_transform</i>	Fits on X and transforms X
<i>get_feature_names</i>	Return feature names for the input features after fitting.
<i>load</i>	Loads component at file path
<i>needs_fitting</i>	Returns boolean determining if component needs fitting before
<i>parameters</i>	Returns the parameters which were used to initialize the component
<i>save</i>	Saves component at file path
<i>transform</i>	Transforms data X.

**clone** (*self*)

Constructs a new component with the same parameters and random state.

**Returns** A new instance of this component with identical parameters and random state.

**default\_parameters** (*cls*)

Returns the default parameters for this component.

Our convention is that `Component.default_parameters == Component().parameters`.

**Returns** default parameters for this component.

**Return type** dict

**describe** (*self*, *print\_name=False*, *return\_dict=False*)

Describe a component and its parameters

**Parameters**

- **print\_name** (*bool*, *optional*) – whether to print name of component
- **return\_dict** (*bool*, *optional*) – whether to return description as dictionary in the format {"name": name, "parameters": parameters}

**Returns** prints and returns dictionary



**Return type** None or dict

**fit** (*self*, *X*, *y*)

Fits component to data

**Parameters**

- **X** (*list*, *pd.DataFrame* or *np.ndarray*) – The input training data of shape [n\_samples, n\_features]
- **y** (*list*, *pd.Series*, *np.ndarray*, *optional*) – The target training data of length [n\_samples]

**Returns** *self*

**fit\_transform** (*self*, *X*, *y*)

Fits on X and transforms X

**Parameters**

- **X** (*pd.DataFrame*) – Data to fit and transform
- **y** (*pd.Series*) – Target data

**Returns** Transformed X

**Return type** *pd.DataFrame*

**get\_feature\_names** (*self*)

Return feature names for the input features after fitting.

**Returns** The feature names after encoding

**Return type** *np.array*

**static load** (*file\_path*)

Loads component at file path

**Parameters** **file\_path** (*str*) – Location to load file

**Returns** *ComponentBase* object

**needs\_fitting** (*self*)

Returns boolean determining if component needs fitting before calling predict, predict\_proba, transform, or feature\_importances. This can be overridden to False for components that do not need to be fit or whose fit methods do nothing.

**property parameters** (*self*)

Returns the parameters which were used to initialize the component

**save** (*self*, *file\_path*, *pickle\_protocol=cloudpickle.DEFAULT\_PROTOCOL*)

Saves component at file path

**Parameters**

- **file\_path** (*str*) – Location to save file
- **pickle\_protocol** (*int*) – The pickle data stream format.

**Returns** None

**transform** (*self*, *X*, *y=None*)

Transforms data X.

**Parameters**

- **X** (*pd.DataFrame*) – Data to transform.

- **y** (*pd.Series, optional*) – Target data.

**Returns** Transformed X

**Return type** `pd.DataFrame`

```
class evalml.pipelines.components.TargetImputer (impute_strategy='most_frequent',  
                                                fill_value=None, random_seed=0,  
                                                **kwargs)
```

Imputes missing target data according to a specified imputation strategy.

#### Parameters

- **impute\_strategy** (*string*) – Impute strategy to use. Valid values include “mean”, “median”, “most\_frequent”, “constant” for numerical data, and “most\_frequent”, “constant” for object data types. Defaults to “most\_frequent”.
- **fill\_value** (*string*) – When `impute_strategy == “constant”`, `fill_value` is used to replace missing data. Defaults to `None` which uses 0 when imputing numerical data and “missing\_value” for strings or object data types.
- **random\_seed** (*int*) – Seed for the random number generator. Defaults to 0.

#### Attributes

<b>hyper-parameter_ranges</b>	{ “impute_strategy”: [“mean”, “median”, “most_frequent”]}
<b>model_family</b>	ModelFamily.NONE
<b>modifies_features</b>	False
<b>modifies_target</b>	True
<b>name</b>	Target Imputer

#### Methods

<i>clone</i>	Constructs a new component with the same parameters and random state.
<i>default_parameters</i>	Returns the default parameters for this component.
<i>describe</i>	Describe a component and its parameters
<i>fit</i>	Fits imputer to target data. ‘None’ values are converted to <code>np.nan</code> before imputation and are
<i>fit_transform</i>	Fits on and transforms the input target data.
<i>load</i>	Loads component at file path
<i>needs_fitting</i>	Returns boolean determining if component needs fitting before
<i>parameters</i>	Returns the parameters which were used to initialize the component
<i>save</i>	Saves component at file path
<i>transform</i>	Transforms input target data by imputing missing values. ‘None’ and <code>np.nan</code> values are treated as the same.

**clone** (*self*)

Constructs a new component with the same parameters and random state.

**Returns** A new instance of this component with identical parameters and random state.

**default\_parameters** (*cls*)

Returns the default parameters for this component.

Our convention is that `Component.default_parameters == Component().parameters`.

**Returns** default parameters for this component.

**Return type** dict

**describe** (*self*, *print\_name=False*, *return\_dict=False*)

Describe a component and its parameters

**Parameters**

- **print\_name** (*bool*, *optional*) – whether to print name of component
- **return\_dict** (*bool*, *optional*) – whether to return description as dictionary in the format {"name": name, "parameters": parameters}

**Returns** prints and returns dictionary

**Return type** None or dict

**fit** (*self*, *X*, *y*)

**Fits imputer to target data. ‘None’ values are converted to np.nan before imputation and are treated as the same.**

**Parameters**

- **X** (*pd.DataFrame* or *np.ndarray*) – The input training data of shape [n\_samples, n\_features]. Ignored.
- **y** (*pd.Series*, *optional*) – The target training data of length [n\_samples].

**Returns** self

**fit\_transform** (*self*, *X*, *y*)

Fits on and transforms the input target data.

**Parameters**

- **X** (*pd.DataFrame*) – Features. Ignored.
- **y** (*pd.Series*) – Target data to impute.

**Returns** The original X, transformed y

**Return type** (pd.DataFrame, pd.Series)

**static load** (*file\_path*)

Loads component at file path

**Parameters** **file\_path** (*str*) – Location to load file

**Returns** ComponentBase object

**needs\_fitting** (*self*)

Returns boolean determining if component needs fitting before calling predict, predict\_proba, transform, or feature\_importances. This can be overridden to False for components that do not need to be fit or whose fit methods do nothing.

**property parameters** (*self*)

Returns the parameters which were used to initialize the component

**save** (*self*, *file\_path*, *pickle\_protocol*=cloudpickle.DEFAULT\_PROTOCOL)

Saves component at file path

#### Parameters

- **file\_path** (*str*) – Location to save file
- **pickle\_protocol** (*int*) – The pickle data stream format.

**Returns** None

**transform** (*self*, *X*, *y*)

Transforms input target data by imputing missing values. ‘None’ and np.nan values are treated as the same.

#### Parameters

- **X** (*pd.DataFrame*) – Features. Ignored.
- **y** (*pd.Series*) – Target data to impute.

**Returns** The original X, transformed y

**Return type** (pd.DataFrame, pd.Series)

**class** evalml.pipelines.components.**TextFeaturizer** (*random\_seed*=0, *\*\*kwargs*)

Transformer that can automatically featurize text columns using featuretools’ nlp\_primitives.

Since models cannot handle non-numeric data, any text must be broken down into features that provide useful information about that text. This component splits each text column into several informative features: Diversity Score, Mean Characters per Word, Polarity Score, and LSA (Latent Semantic Analysis). Calling transform on this component will replace any text columns in the given dataset with these numeric columns.

**Parameters** **random\_seed** (*int*) – Seed for the random number generator. Defaults to 0.

#### Attributes

<b>hyper-parameter_ranges</b>	{}
<b>model_family</b>	ModelFamily.NONE
<b>modifies_features</b>	True
<b>modifies_target</b>	False
<b>name</b>	Text Featurization Component

#### Methods

<i>clone</i>	Constructs a new component with the same parameters and random state.
<i>default_parameters</i>	Returns the default parameters for this component.
<i>describe</i>	Describe a component and its parameters
<i>fit</i>	Fits component to data
<i>fit_transform</i>	Fits on X and transforms X
<i>load</i>	Loads component at file path
<i>needs_fitting</i>	Returns boolean determining if component needs fitting before
<i>parameters</i>	Returns the parameters which were used to initialize the component

continues on next page

Table 595 – continued from previous page

<i>save</i>	Saves component at file path
<i>transform</i>	Transforms data X by creating new features using existing text columns

**clone** (*self*)

Constructs a new component with the same parameters and random state.

**Returns** A new instance of this component with identical parameters and random state.

**default\_parameters** (*cls*)

Returns the default parameters for this component.

Our convention is that `Component.default_parameters == Component().parameters`.

**Returns** default parameters for this component.

**Return type** dict

**describe** (*self*, *print\_name=False*, *return\_dict=False*)

Describe a component and its parameters

**Parameters**

- **print\_name** (*bool*, *optional*) – whether to print name of component
- **return\_dict** (*bool*, *optional*) – whether to return description as dictionary in the format {"name": name, "parameters": parameters}

**Returns** prints and returns dictionary

**Return type** None or dict

**fit** (*self*, *X*, *y=None*)

Fits component to data

**Parameters**

- **X** (*pd.DataFrame* or *np.ndarray*) – The input training data of shape [n\_samples, n\_features]
- **y** (*pd.Series*, *np.ndarray*, *optional*) – The target training data of length [n\_samples]

**Returns** self

**fit\_transform** (*self*, *X*, *y=None*)

Fits on X and transforms X

**Parameters**

- **X** (*pd.DataFrame*) – Data to fit and transform
- **y** (*pd.Series*) – Target data

**Returns** Transformed X

**Return type** pd.DataFrame

**static load** (*file\_path*)

Loads component at file path

**Parameters** **file\_path** (*str*) – Location to load file

**Returns** ComponentBase object

**needs\_fitting** (*self*)

Returns boolean determining if component needs fitting before calling predict, predict\_proba, transform, or feature\_importances. This can be overridden to False for components that do not need to be fit or whose fit methods do nothing.

**property parameters** (*self*)

Returns the parameters which were used to initialize the component

**save** (*self*, *file\_path*, *pickle\_protocol=cloudpickle.DEFAULT\_PROTOCOL*)

Saves component at file path

**Parameters**

- **file\_path** (*str*) – Location to save file
- **pickle\_protocol** (*int*) – The pickle data stream format.

**Returns** None

**transform** (*self*, *X*, *y=None*)

Transforms data X by creating new features using existing text columns

**Parameters**

- **X** (*pd.DataFrame*) – The data to transform.
- **y** (*pd.Series*, *optional*) – Ignored.

**Returns** Transformed X

**Return type** pd.DataFrame

```
class evalml.pipelines.components.TimeSeriesBaselineEstimator (gap=1, random_seed=0,  
                                                                **kwargs)
```

Time series estimator that predicts using the naive forecasting approach.

This is useful as a simple baseline estimator for time series problems.

**Parameters**

- **gap** (*int*) – Gap between prediction date and target date and must be a positive integer. If gap is 0, target date will be shifted ahead by 1 time period. Defaults to 1.
- **random\_seed** (*int*) – Seed for the random number generator. Defaults to 0.

**Attributes**

<b>hyper-parameter_ranges</b>	{}
<b>model_family</b>	ModelFamily.BASELINE
<b>modifies_features</b>	True
<b>modifies_target</b>	False
<b>name</b>	Time Series Baseline Estimator
<b>predict_uses_y</b>	True
<b>supported_problem_types</b>	[ProblemTypes.TIME_SERIES_REGRESSION, ProblemTypes.TIME_SERIES_BINARY, ProblemTypes.TIME_SERIES_MULTICLASS,]

## Methods

<code>clone</code>	Constructs a new component with the same parameters and random state.
<code>default_parameters</code>	Returns the default parameters for this component.
<code>describe</code>	Describe a component and its parameters
<code>feature_importance</code>	Returns importance associated with each feature.
<code>fit</code>	Fits component to data
<code>load</code>	Loads component at file path
<code>needs_fitting</code>	Returns boolean determining if component needs fitting before
<code>parameters</code>	Returns the parameters which were used to initialize the component
<code>predict</code>	Make predictions using selected features.
<code>predict_proba</code>	Make probability estimates for labels.
<code>save</code>	Saves component at file path

### **clone** (*self*)

Constructs a new component with the same parameters and random state.

**Returns** A new instance of this component with identical parameters and random state.

### **default\_parameters** (*cls*)

Returns the default parameters for this component.

Our convention is that `Component.default_parameters == Component().parameters`.

**Returns** default parameters for this component.

**Return type** dict

### **describe** (*self*, *print\_name=False*, *return\_dict=False*)

Describe a component and its parameters

#### Parameters

- **print\_name** (*bool*, *optional*) – whether to print name of component
- **return\_dict** (*bool*, *optional*) – whether to return description as dictionary in the format {"name": name, "parameters": parameters}

**Returns** prints and returns dictionary

**Return type** None or dict

### **property feature\_importance** (*self*)

Returns importance associated with each feature.

Since baseline estimators do not use input features to calculate predictions, returns an array of zeroes.

**Returns** an array of zeroes

**Return type** np.ndarray (float)

### **fit** (*self*, *X*, *y=None*)

Fits component to data

#### Parameters

- **X** (*list*, *pd.DataFrame* or *np.ndarray*) – The input training data of shape [n\_samples, n\_features]

- **y** (*list*, *pd.Series*, *np.ndarray*, *optional*) – The target training data of length [n\_samples]

**Returns** self

**static load** (*file\_path*)

Loads component at file path

**Parameters** **file\_path** (*str*) – Location to load file

**Returns** ComponentBase object

**needs\_fitting** (*self*)

Returns boolean determining if component needs fitting before calling predict, predict\_proba, transform, or feature\_importances. This can be overridden to False for components that do not need to be fit or whose fit methods do nothing.

**property parameters** (*self*)

Returns the parameters which were used to initialize the component

**predict** (*self*, *X*, *y=None*)

Make predictions using selected features.

**Parameters** **X** (*pd.DataFrame*, *np.ndarray*) – Data of shape [n\_samples, n\_features]

**Returns** Predicted values

**Return type** *pd.Series*

**predict\_proba** (*self*, *X*, *y=None*)

Make probability estimates for labels.

**Parameters** **X** (*pd.DataFrame*, *or np.ndarray*) – Features

**Returns** Probability estimates

**Return type** *pd.Series*

**save** (*self*, *file\_path*, *pickle\_protocol=cloudpickle.DEFAULT\_PROTOCOL*)

Saves component at file path

**Parameters**

- **file\_path** (*str*) – Location to save file
- **pickle\_protocol** (*int*) – The pickle data stream format.

**Returns** None

**class** evalml.pipelines.components.**Transformer** (*parameters=None*, *compo-  
nent\_obj=None*, *random\_seed=0*,  
*\*\*kwargs*)

A component that may or may not need fitting that transforms data. These components are used before an estimator.

To implement a new Transformer, define your own class which is a subclass of Transformer, including a name and a list of acceptable ranges for any parameters to be tuned during the automl search (hyperparameters). Define an `__init__` method which sets up any necessary state and objects. Make sure your `__init__` only uses standard keyword arguments and calls `super().__init__()` with a parameters dict. You may also override the `fit`, `transform`, `fit_transform` and other methods in this class if appropriate.

To see some examples, check out the definitions of any Transformer component.

**Parameters**

- **parameters** (*dict*) – Dictionary of parameters for the component. Defaults to None.



- **component\_obj** (*obj*) – Third-party objects useful in component implementation. Defaults to None.
- **random\_seed** (*int*) – Seed for the random number generator. Defaults to 0.

#### Attributes

<b>model_family</b>	ModelFamily.NONE
<b>modifies_features</b>	True
<b>modifies_target</b>	False

#### Methods

<i>clone</i>	Constructs a new component with the same parameters and random state.
<i>default_parameters</i>	Returns the default parameters for this component.
<i>describe</i>	Describe a component and its parameters
<i>fit</i>	Fits component to data
<i>fit_transform</i>	Fits on X and transforms X
<i>load</i>	Loads component at file path
<i>name</i>	Returns string name of this component
<i>needs_fitting</i>	Returns boolean determining if component needs fitting before
<i>parameters</i>	Returns the parameters which were used to initialize the component
<i>save</i>	Saves component at file path
<i>transform</i>	Transforms data X.

#### **clone** (*self*)

Constructs a new component with the same parameters and random state.

**Returns** A new instance of this component with identical parameters and random state.

#### **default\_parameters** (*cls*)

Returns the default parameters for this component.

Our convention is that `Component.default_parameters == Component().parameters`.

**Returns** default parameters for this component.

**Return type** dict

#### **describe** (*self*, *print\_name=False*, *return\_dict=False*)

Describe a component and its parameters

##### Parameters

- **print\_name** (*bool*, *optional*) – whether to print name of component
- **return\_dict** (*bool*, *optional*) – whether to return description as dictionary in the format {"name": name, "parameters": parameters}

**Returns** prints and returns dictionary

**Return type** None or dict

**fit** (*self*, *X*, *y=None*)

Fits component to data

**Parameters**

- **X** (*list*, *pd.DataFrame* or *np.ndarray*) – The input training data of shape [n\_samples, n\_features]
- **y** (*list*, *pd.Series*, *np.ndarray*, *optional*) – The target training data of length [n\_samples]

**Returns** *self*

**fit\_transform** (*self*, *X*, *y=None*)

Fits on X and transforms X

**Parameters**

- **X** (*pd.DataFrame*) – Data to fit and transform
- **y** (*pd.Series*) – Target data

**Returns** Transformed X

**Return type** *pd.DataFrame*

**static load** (*file\_path*)

Loads component at file path

**Parameters** **file\_path** (*str*) – Location to load file

**Returns** *ComponentBase* object

**property name** (*cls*)

Returns string name of this component

**needs\_fitting** (*self*)

Returns boolean determining if component needs fitting before calling predict, predict\_proba, transform, or feature\_importances. This can be overridden to False for components that do not need to be fit or whose fit methods do nothing.

**property parameters** (*self*)

Returns the parameters which were used to initialize the component

**save** (*self*, *file\_path*, *pickle\_protocol=cloudpickle.DEFAULT\_PROTOCOL*)

Saves component at file path

**Parameters**

- **file\_path** (*str*) – Location to save file
- **pickle\_protocol** (*int*) – The pickle data stream format.

**Returns** *None*

**transform** (*self*, *X*, *y=None*)

Transforms data X.

**Parameters**

- **X** (*pd.DataFrame*) – Data to transform.
- **y** (*pd.Series*, *optional*) – Target data.

**Returns** Transformed X

**Return type** *pd.DataFrame*

```
class evalml.pipelines.components.Undersampler(sampling_ratio=0.25, sam-
                                             pling_ratio_dict=None,
                                             min_samples=100,
                                             min_percentage=0.1, random_seed=0,
                                             **kwargs)
```

Initializes an undersampling transformer to downsample the majority classes in the dataset.

This component is only run during training and not during predict.

### Parameters

- **sampling\_ratio** (*float*) – The smallest minority:majority ratio that is accepted as ‘balanced’. For instance, a 1:4 ratio would be represented as 0.25, while a 1:1 ratio is 1.0. Must be between 0 and 1, inclusive. Defaults to 0.25.
- **sampling\_ratio\_dict** (*dict*) – A dictionary specifying the desired balanced ratio for each target value. For instance, in a binary case where class 1 is the minority, we could specify: *sampling\_ratio\_dict*={0: 0.5, 1: 1}, which means we would undersample class 0 to have twice the number of samples as class 1 (minority:majority ratio = 0.5), and don’t sample class 1. Overrides *sampling\_ratio* if provided. Defaults to None.
- **min\_samples** (*int*) – The minimum number of samples that we must have for any class, pre or post sampling. If a class must be downsampled, it will not be downsampled past this value. To determine severe imbalance, the minority class must occur less often than this and must have a class ratio below *min\_percentage*. Must be greater than 0. Defaults to 100.
- **min\_percentage** (*float*) – The minimum percentage of the minimum class to total dataset that we tolerate, as long as it is above *min\_samples*. If *min\_percentage* and *min\_samples* are not met, treat this as severely imbalanced, and we will not resample the data. Must be between 0 and 0.5, inclusive. Defaults to 0.1.
- **random\_seed** (*int*) – The seed to use for random sampling. Defaults to 0.

### Attributes

<b>hyper-parameter_ranges</b>	{}
<b>model_family</b>	ModelFamily.NONE
<b>modifies_features</b>	True
<b>modifies_target</b>	True
<b>name</b>	Undersampler

### Methods

<i>clone</i>	Constructs a new component with the same parameters and random state.
<i>default_parameters</i>	Returns the default parameters for this component.
<i>describe</i>	Describe a component and its parameters
<i>fit</i>	Fits the sampler to the data.
<i>fit_transform</i>	Fits on X and transforms X
<i>load</i>	Loads component at file path
<i>needs_fitting</i>	Returns boolean determining if component needs fitting before

continues on next page

Table 598 – continued from previous page

<i>parameters</i>	Returns the parameters which were used to initialize the component
<i>save</i>	Saves component at file path
<i>transform</i>	Transforms the input data by sampling the data.

**clone** (*self*)

Constructs a new component with the same parameters and random state.

**Returns** A new instance of this component with identical parameters and random state.

**default\_parameters** (*cls*)

Returns the default parameters for this component.

Our convention is that `Component.default_parameters == Component().parameters`.

**Returns** default parameters for this component.

**Return type** dict

**describe** (*self*, *print\_name=False*, *return\_dict=False*)

Describe a component and its parameters

**Parameters**

- **print\_name** (*bool*, *optional*) – whether to print name of component
- **return\_dict** (*bool*, *optional*) – whether to return description as dictionary in the format {"name": name, "parameters": parameters}

**Returns** prints and returns dictionary

**Return type** None or dict

**fit** (*self*, *X*, *y*)

Fits the sampler to the data.

**Parameters**

- **X** (*pd.DataFrame*) – Input features.
- **y** (*pd.Series*) – Target.

**Returns** self

**fit\_transform** (*self*, *X*, *y*)

Fits on X and transforms X

**Parameters**

- **X** (*pd.DataFrame*) – Data to fit and transform
- **y** (*pd.Series*) – Target data

**Returns** Transformed X

**Return type** pd.DataFrame

**static load** (*file\_path*)

Loads component at file path

**Parameters** **file\_path** (*str*) – Location to load file

**Returns** ComponentBase object

**needs\_fitting** (*self*)

Returns boolean determining if component needs fitting before calling predict, predict\_proba, transform, or feature\_importances. This can be overridden to False for components that do not need to be fit or whose fit methods do nothing.

**property parameters** (*self*)

Returns the parameters which were used to initialize the component

**save** (*self*, *file\_path*, *pickle\_protocol=cloudpickle.DEFAULT\_PROTOCOL*)

Saves component at file path

**Parameters**

- **file\_path** (*str*) – Location to save file
- **pickle\_protocol** (*int*) – The pickle data stream format.

**Returns** None

**transform** (*self*, *X*, *y=None*)

Transforms the input data by sampling the data.

**Parameters**

- **X** (*pd.DataFrame*) – Training features.
- **y** (*pd.Series*) – Target.

**Returns** Transformed features and target.

**Return type** *pd.DataFrame*, *pd.Series*

**class** evalml.pipelines.components.**URLFeaturizer** (*random\_seed=0*, *\*\*kwargs*)

Transformer that can automatically extract features from URL.

**Parameters** **random\_seed** (*int*) – Seed for the random number generator. Defaults to 0.

**Attributes**

<b>hyper-parameter_ranges</b>	{}
<b>model_family</b>	ModelFamily.NONE
<b>modifies_features</b>	True
<b>modifies_target</b>	False
<b>name</b>	URL Featurizer

**Methods**

<i>clone</i>	Constructs a new component with the same parameters and random state.
<i>default_parameters</i>	Returns the default parameters for this component.
<i>describe</i>	Describe a component and its parameters
<i>fit</i>	Fits component to data
<i>fit_transform</i>	Fits on X and transforms X
<i>load</i>	Loads component at file path

continues on next page

Table 599 – continued from previous page

<code>needs_fitting</code>	Returns boolean determining if component needs fitting before
<code>parameters</code>	Returns the parameters which were used to initialize the component
<code>save</code>	Saves component at file path
<code>transform</code>	Transforms data X.

**clone** (*self*)

Constructs a new component with the same parameters and random state.

**Returns** A new instance of this component with identical parameters and random state.**default\_parameters** (*cls*)

Returns the default parameters for this component.

Our convention is that `Component.default_parameters == Component().parameters`.**Returns** default parameters for this component.**Return type** dict**describe** (*self*, *print\_name=False*, *return\_dict=False*)

Describe a component and its parameters

**Parameters**

- **print\_name** (*bool*, *optional*) – whether to print name of component
- **return\_dict** (*bool*, *optional*) – whether to return description as dictionary in the format {"name": name, "parameters": parameters}

**Returns** prints and returns dictionary**Return type** None or dict**fit** (*self*, *X*, *y=None*)

Fits component to data

**Parameters**

- **X** (*list*, *pd.DataFrame* or *np.ndarray*) – The input training data of shape [n\_samples, n\_features]
- **y** (*list*, *pd.Series*, *np.ndarray*, *optional*) – The target training data of length [n\_samples]

**Returns** self**fit\_transform** (*self*, *X*, *y=None*)

Fits on X and transforms X

**Parameters**

- **X** (*pd.DataFrame*) – Data to fit and transform
- **y** (*pd.Series*) – Target data

**Returns** Transformed X**Return type** pd.DataFrame**static load** (*file\_path*)

Loads component at file path

**Parameters** **file\_path** (*str*) – Location to load file

**Returns** ComponentBase object

**needs\_fitting** (*self*)

Returns boolean determining if component needs fitting before calling predict, predict\_proba, transform, or feature\_importances. This can be overridden to False for components that do not need to be fit or whose fit methods do nothing.

**property parameters** (*self*)

Returns the parameters which were used to initialize the component

**save** (*self*, *file\_path*, *pickle\_protocol*=cloudpickle.DEFAULT\_PROTOCOL)

Saves component at file path

**Parameters**

- **file\_path** (*str*) – Location to save file
- **pickle\_protocol** (*int*) – The pickle data stream format.

**Returns** None

**transform** (*self*, *X*, *y*=None)

Transforms data X.

**Parameters**

- **X** (*pd.DataFrame*) – Data to transform.
- **y** (*pd.Series*, *optional*) – Target data.

**Returns** Transformed X

**Return type** pd.DataFrame

```
class evalml.pipelines.components.XGBoostClassifier(eta=0.1, max_depth=6,
                                                    min_child_weight=1,
                                                    n_estimators=100, ran-
                                                    dom_seed=0, n_jobs=-1,
                                                    **kwargs)
```

XGBoost Classifier.

**Parameters**

- **eta** (*float*) – Boosting learning rate. Defaults to 0.1.
- **max\_depth** (*int*) – Maximum tree depth for base learners. Defaults to 6.
- **min\_child\_weight** (*float*) – Minimum sum of instance weight (hessian) needed in a child. Defaults to 1.0
- **n\_estimators** (*int*) – Number of gradient boosted trees. Equivalent to number of boosting rounds. Defaults to 100.
- **random\_seed** (*int*) – Seed for the random number generator. Defaults to 0.
- **n\_jobs** (*int*) – Number of parallel threads used to run xgboost. Note that creating thread contention will significantly slow down the algorithm. Defaults to -1.

**Attributes**

<b>hyper-parameter_ranges</b>	{ “eta”: Real(0.000001, 1), “max_depth”: Integer(1, 10), “min_child_weight”: Real(1, 10), “n_estimators”: Integer(1, 1000), }
<b>model_family</b>	ModelFamily.XGBOOST
<b>modifies_features</b>	True
<b>modifies_target</b>	False
<b>name</b>	XGBoost Classifier
<b>pre-dict_uses_y</b>	False
<b>SEED_MAX</b>	None
<b>SEED_MIN</b>	None
<b>supported_problem_types</b>	[ ProblemTypes.BINARY, ProblemTypes.MULTICLASS, ProblemTypes.TIME_SERIES_BINARY, ProblemTypes.TIME_SERIES_MULTICLASS, ]

## Methods

<i>clone</i>	Constructs a new component with the same parameters and random state.
<i>default_parameters</i>	Returns the default parameters for this component.
<i>describe</i>	Describe a component and its parameters
<i>feature_importance</i>	Returns importance associated with each feature.
<i>fit</i>	Fits component to data
<i>load</i>	Loads component at file path
<i>needs_fitting</i>	Returns boolean determining if component needs fitting before
<i>parameters</i>	Returns the parameters which were used to initialize the component
<i>predict</i>	Make predictions using selected features.
<i>predict_proba</i>	Make probability estimates for labels.
<i>save</i>	Saves component at file path

### **clone** (*self*)

Constructs a new component with the same parameters and random state.

**Returns** A new instance of this component with identical parameters and random state.

### **default\_parameters** (*cls*)

Returns the default parameters for this component.

Our convention is that `Component.default_parameters == Component().parameters`.

**Returns** default parameters for this component.

**Return type** dict

### **describe** (*self*, *print\_name=False*, *return\_dict=False*)

Describe a component and its parameters

#### Parameters

- **print\_name** (*bool*, *optional*) – whether to print name of component
- **return\_dict** (*bool*, *optional*) – whether to return description as dictionary in the format {“name”: name, “parameters”: parameters}



**Returns** prints and returns dictionary

**Return type** None or dict

**property feature\_importance** (*self*)

Returns importance associated with each feature.

**Returns** Importance associated with each feature

**Return type** np.ndarray

**fit** (*self*, *X*, *y=None*)

Fits component to data

**Parameters**

- **X** (*list*, *pd.DataFrame* or *np.ndarray*) – The input training data of shape [n\_samples, n\_features]
- **y** (*list*, *pd.Series*, *np.ndarray*, *optional*) – The target training data of length [n\_samples]

**Returns** self

**static load** (*file\_path*)

Loads component at file path

**Parameters** **file\_path** (*str*) – Location to load file

**Returns** ComponentBase object

**needs\_fitting** (*self*)

Returns boolean determining if component needs fitting before calling predict, predict\_proba, transform, or feature\_importances. This can be overridden to False for components that do not need to be fit or whose fit methods do nothing.

**property parameters** (*self*)

Returns the parameters which were used to initialize the component

**predict** (*self*, *X*)

Make predictions using selected features.

**Parameters** **X** (*pd.DataFrame*, *np.ndarray*) – Data of shape [n\_samples, n\_features]

**Returns** Predicted values

**Return type** pd.Series

**predict\_proba** (*self*, *X*)

Make probability estimates for labels.

**Parameters** **X** (*pd.DataFrame*, or *np.ndarray*) – Features

**Returns** Probability estimates

**Return type** pd.Series

**save** (*self*, *file\_path*, *pickle\_protocol=cloudpickle.DEFAULT\_PROTOCOL*)

Saves component at file path

**Parameters**

- **file\_path** (*str*) – Location to save file
- **pickle\_protocol** (*int*) – The pickle data stream format.

**Returns** None

```
class evalml.pipelines.components.XGBoostRegressor (eta=0.1,          max_depth=6,
                                                    min_child_weight=1,
                                                    n_estimators=100,      ran-
                                                    dom_seed=0,      n_jobs=-    1,
                                                    **kwargs)
```

XGBoost Regressor.

### Parameters

- **eta** (*float*) – Boosting learning rate. Defaults to 0.1.
- **max\_depth** (*int*) – Maximum tree depth for base learners. Defaults to 6.
- **min\_child\_weight** (*float*) – Minimum sum of instance weight (hessian) needed in a child. Defaults to 1.0
- **n\_estimators** (*int*) – Number of gradient boosted trees. Equivalent to number of boosting rounds. Defaults to 100.
- **random\_seed** (*int*) – Seed for the random number generator. Defaults to 0.
- **n\_jobs** (*int*) – Number of parallel threads used to run xgboost. Note that creating thread contention will significantly slow down the algorithm. Defaults to -1.

### Attributes

<b>hyper-parameter_ranges</b>	{ "eta": Real(0.000001, 1), "max_depth": Integer(1, 20), "min_child_weight": Real(1, 10), "n_estimators": Integer(1, 1000), }
<b>model_family</b>	ModelFamily.XGBOOST
<b>modifies_features</b>	True
<b>modifies_target</b>	False
<b>name</b>	XGBoost Regressor
<b>predict Uses y</b>	False
<b>SEED_MAX</b>	None
<b>SEED_MIN</b>	None
<b>supported_problem_types</b>	[ ProblemTypes.REGRESSION, ProblemTypes.TIME_SERIES_REGRESSION, ]

### Methods

<i>clone</i>	Constructs a new component with the same parameters and random state.
<i>default_parameters</i>	Returns the default parameters for this component.
<i>describe</i>	Describe a component and its parameters
<i>feature_importance</i>	Returns importance associated with each feature.
<i>fit</i>	Fits component to data
<i>load</i>	Loads component at file path
<i>needs_fitting</i>	Returns boolean determining if component needs fitting before
<i>parameters</i>	Returns the parameters which were used to initialize the component
<i>predict</i>	Make predictions using selected features.

continues on next page

Table 601 – continued from previous page

<code>predict_proba</code>	Make probability estimates for labels.
<code>save</code>	Saves component at file path

**clone** (*self*)

Constructs a new component with the same parameters and random state.

**Returns** A new instance of this component with identical parameters and random state.

**default\_parameters** (*cls*)

Returns the default parameters for this component.

Our convention is that `Component.default_parameters == Component().parameters`.

**Returns** default parameters for this component.

**Return type** dict

**describe** (*self*, *print\_name=False*, *return\_dict=False*)

Describe a component and its parameters

**Parameters**

- **print\_name** (*bool*, *optional*) – whether to print name of component
- **return\_dict** (*bool*, *optional*) – whether to return description as dictionary in the format {"name": name, "parameters": parameters}

**Returns** prints and returns dictionary

**Return type** None or dict

**property feature\_importance** (*self*)

Returns importance associated with each feature.

**Returns** Importance associated with each feature

**Return type** np.ndarray

**fit** (*self*, *X*, *y=None*)

Fits component to data

**Parameters**

- **X** (*list*, *pd.DataFrame* or *np.ndarray*) – The input training data of shape [n\_samples, n\_features]
- **y** (*list*, *pd.Series*, *np.ndarray*, *optional*) – The target training data of length [n\_samples]

**Returns** self

**static load** (*file\_path*)

Loads component at file path

**Parameters** **file\_path** (*str*) – Location to load file

**Returns** ComponentBase object

**needs\_fitting** (*self*)

Returns boolean determining if component needs fitting before calling predict, predict\_proba, transform, or feature\_importances. This can be overridden to False for components that do not need to be fit or whose fit methods do nothing.

**property parameters** (*self*)

Returns the parameters which were used to initialize the component

**predict** (*self*, *X*)

Make predictions using selected features.

**Parameters** **X** (*pd.DataFrame*, *np.ndarray*) – Data of shape [n\_samples, n\_features]

**Returns** Predicted values

**Return type** *pd.Series*

**predict\_proba** (*self*, *X*)

Make probability estimates for labels.

**Parameters** **X** (*pd.DataFrame*, *or np.ndarray*) – Features

**Returns** Probability estimates

**Return type** *pd.Series*

**save** (*self*, *file\_path*, *pickle\_protocol=cloudpickle.DEFAULT\_PROTOCOL*)

Saves component at file path

**Parameters**

- **file\_path** (*str*) – Location to save file
- **pickle\_protocol** (*int*) – The pickle data stream format.

**Returns** None

## Submodules

### binary\_classification\_pipeline

## Module Contents

## Classes Summary

<i>BinaryClassificationPipeline</i>	Pipeline subclass for all binary classification pipelines.
-------------------------------------	--

## Contents

**class** evalml.pipelines.binary\_classification\_pipeline.**BinaryClassificationPipeline** (*component*

*pa-  
ram-  
e-  
ters=None,  
cus-  
tom\_name=  
ran-  
dom\_seed=*

Pipeline subclass for all binary classification pipelines.

**Parameters**

- **component\_graph** (*list or dict*) – List of components in order. Accepts strings or ComponentBase subclasses in the list. Note that when duplicate components are specified in a list, the duplicate component names will be modified with the component’s index

in the list. For example, the component graph [Imputer, One Hot Encoder, Imputer, Logistic Regression Classifier] will have names ["Imputer", "One Hot Encoder", "Imputer\_2", "Logistic Regression Classifier"]

- **parameters** (*dict*) – Dictionary with component names as keys and dictionary of that component’s parameters as values. An empty dictionary or None implies using all default values for component parameters. Defaults to None.
- **custom\_name** (*str*) – Custom name for the pipeline. Defaults to None.
- **random\_seed** (*int*) – Seed for the random number generator. Defaults to 0.

#### Attributes

<b>problem_type</b>	ProblemTypes.BINARY
---------------------	---------------------

#### Methods

<i>can_tune_threshold_with_objective</i>	Determine whether the threshold of a binary classification pipeline can be tuned.
<i>classes_</i>	Gets the class names for the problem.
<i>clone</i>	Constructs a new pipeline with the same components, parameters, and random state.
<i>compute_estimator_features</i>	Transforms the data by applying all pre-processing components.
<i>create_objectives</i>	
<i>custom_name</i>	Custom name of the pipeline.
<i>describe</i>	Outputs pipeline details including component parameters
<i>feature_importance</i>	Importance associated with each feature. Features dropped by the feature selection are excluded.
<i>fit</i>	Build a classification model. For string and categorical targets, classes are sorted
<i>get_component</i>	Returns component by name
<i>get_hyperparameter_ranges</i>	Returns hyperparameter ranges from all components as a dictionary.
<i>graph</i>	Generate an image representing the pipeline graph.
<i>graph_feature_importance</i>	Generate a bar graph of the pipeline’s feature importance
<i>inverse_transform</i>	Apply component <i>inverse_transform</i> methods to estimator predictions in reverse order.
<i>load</i>	Loads pipeline at file path
<i>model_family</i>	Returns model family of this pipeline.
<i>name</i>	Name of the pipeline.
<i>new</i>	Constructs a new instance of the pipeline with the same component graph but with a different set of parameters.
<i>optimize_threshold</i>	Optimize the pipeline threshold given the objective to use. Only used for binary problems with objectives whose thresholds can be tuned.
<i>parameters</i>	Parameter dictionary for this pipeline.

continues on next page

Table 603 – continued from previous page

<code>predict</code>	Make predictions using selected features.
<code>predict_proba</code>	Make probability estimates for labels. Assumes that the column at index 1 represents the positive label case.
<code>save</code>	Saves pipeline at file path
<code>score</code>	Evaluate model performance on objectives
<code>summary</code>	A short summary of the pipeline structure, describing the list of components used.
<code>threshold</code>	Threshold used to make a prediction. Defaults to None.

**can\_tune\_threshold\_with\_objective** (*self*, *objective*)

Determine whether the threshold of a binary classification pipeline can be tuned.

**Parameters**

- **pipeline** (*PipelineBase*) – Binary classification pipeline.
- **objective** – Primary AutoMLSearch objective.

**property classes\_** (*self*)

Gets the class names for the problem.

**clone** (*self*)

Constructs a new pipeline with the same components, parameters, and random state.

**Returns** A new instance of this pipeline with identical components, parameters, and random state.

**compute\_estimator\_features** (*self*, *X*, *y=None*)

Transforms the data by applying all pre-processing components.

**Parameters** **X** (*pd.DataFrame*) – Input data to the pipeline to transform.

**Returns** New transformed features.

**Return type** *pd.DataFrame*

**static create\_objectives** (*objectives*)

**property custom\_name** (*self*)

Custom name of the pipeline.

**describe** (*self*, *return\_dict=False*)

Outputs pipeline details including component parameters

**Parameters** **return\_dict** (*bool*) – If True, return dictionary of information about pipeline. Defaults to False.

**Returns** Dictionary of all component parameters if *return\_dict* is True, else None

**Return type** *dict*

**property feature\_importance** (*self*)

Importance associated with each feature. Features dropped by the feature selection are excluded.

**Returns** *pd.DataFrame* including feature names and their corresponding importance

**fit** (*self*, *X*, *y*)

**Build a classification model. For string and categorical targets, classes are sorted** by `sorted(set(y))` and then are mapped to values between 0 and *n\_classes*-1.

**Parameters**

- **X** (*pd.DataFrame* or *np.ndarray*) – The input training data of shape [n\_samples, n\_features]
- **y** (*pd.Series*, *np.ndarray*) – The target training labels of length [n\_samples]

**Returns** self**get\_component** (*self*, *name*)

Returns component by name

**Parameters** **name** (*str*) – Name of component**Returns** Component to return**Return type** Component**get\_hyperparameter\_ranges** (*self*, *custom\_hyperparameters*)

Returns hyperparameter ranges from all components as a dictionary.

**Parameters** **custom\_hyperparameters** (*dict*) – Custom hyperparameters for the pipeline.**Returns** Dictionary of hyperparameter ranges for each component in the pipeline.**Return type** dict**graph** (*self*, *filepath=None*)

Generate an image representing the pipeline graph.

**Parameters** **filepath** (*str*, *optional*) – Path to where the graph should be saved. If set to None (as by default), the graph will not be saved.**Returns** Graph object that can be directly displayed in Jupyter notebooks.**Return type** graphviz.Digraph**graph\_feature\_importance** (*self*, *importance\_threshold=0*)

Generate a bar graph of the pipeline's feature importance

**Parameters** **importance\_threshold** (*float*, *optional*) – If provided, graph features with a permutation importance whose absolute value is larger than importance\_threshold. Defaults to zero.**Returns** plotly.Figure, a bar graph showing features and their corresponding importance**inverse\_transform** (*self*, *y*)

Apply component inverse\_transform methods to estimator predictions in reverse order.

Components that implement inverse\_transform are PolynomialDetrender, LabelEncoder (tbd).

**Parameters** **y** (*pd.Series*) – Final component features**static load** (*file\_path*)

Loads pipeline at file path

**Parameters** **file\_path** (*str*) – location to load file**Returns** PipelineBase object**property model\_family** (*self*)

Returns model family of this pipeline.

**property name** (*self*)

Name of the pipeline.

**new** (*self*, *parameters*, *random\_seed*=0)

**Constructs a new instance of the pipeline with the same component graph but with a different set of parameters.**  
Not to be confused with python's `__new__` method.

**Parameters**

- **parameters** (*dict*) – Dictionary with component names as keys and dictionary of that component's parameters as values. An empty dictionary or None implies using all default values for component parameters. Defaults to None.
- **random\_seed** (*int*) – Seed for the random number generator. Defaults to 0.

**Returns** A new instance of this pipeline with identical components.

**optimize\_threshold** (*self*, *X*, *y*, *y\_pred\_proba*, *objective*)

Optimize the pipeline threshold given the objective to use. Only used for binary problems with objectives whose thresholds can be tuned.

**Parameters**

- **X** (*pd.DataFrame*) – Input features
- **y** (*pd.Series*) – Input target values
- **y\_pred\_proba** (*pd.Series*) – The predicted probabilities of the target outputted by the pipeline
- **objective** (*ObjectiveBase*) – The objective to threshold with. Must have a tunable threshold.

**property parameters** (*self*)

Parameter dictionary for this pipeline.

**Returns** Dictionary of all component parameters.

**Return type** dict

**predict** (*self*, *X*, *objective*=None)

Make predictions using selected features.

**Parameters**

- **X** (*pd.DataFrame*, or *np.ndarray*) – Data of shape [n\_samples, n\_features]
- **objective** (*Object* or *string*) – The objective to use to make predictions

**Returns** Estimated labels

**Return type** pd.Series

**predict\_proba** (*self*, *X*)

Make probability estimates for labels. Assumes that the column at index 1 represents the positive label case.

**Parameters** **X** (*pd.DataFrame* or *np.ndarray*) – Data of shape [n\_samples, n\_features]

**Returns** Probability estimates

**Return type** pd.Series

**save** (*self*, *file\_path*, *pickle\_protocol*=cloudpickle.DEFAULT\_PROTOCOL)

Saves pipeline at file path

**Parameters**



- **file\_path** (*str*) – location to save file
- **pickle\_protocol** (*int*) – the pickle data stream format.

**Returns** None

**score** (*self*, *X*, *y*, *objectives*)

Evaluate model performance on objectives

**Parameters**

- **X** (*pd.DataFrame* or *np.ndarray*) – Data of shape [n\_samples, n\_features]
- **y** (*pd.Series*, or *np.ndarray*) – True labels of length [n\_samples]
- **objectives** (*list*) – List of objectives to score

**Returns** Ordered dictionary of objective scores

**Return type** dict

**property summary** (*self*)

A short summary of the pipeline structure, describing the list of components used. Example: Logistic Regression Classifier w/ Simple Imputer + One Hot Encoder

**property threshold** (*self*)

Threshold used to make a prediction. Defaults to None.

## binary\_classification\_pipeline\_mixin

### Module Contents

### Classes Summary

---

*BinaryClassificationPipelineMixin*

---

### Contents

**class** evalml.pipelines.binary\_classification\_pipeline\_mixin.**BinaryClassificationPipelineMixin**

**Methods**

<i>optimize_threshold</i>	Optimize the pipeline threshold given the objective to use. Only used for binary problems with objectives whose thresholds can be tuned.
<i>threshold</i>	Threshold used to make a prediction. Defaults to None.

**optimize\_threshold** (*self*, *X*, *y*, *y\_pred\_proba*, *objective*)

Optimize the pipeline threshold given the objective to use. Only used for binary problems with objectives whose thresholds can be tuned.

**Parameters**

- **X** (*pd.DataFrame*) – Input features
- **y** (*pd.Series*) – Input target values

- **y\_pred\_proba** (*pd.Series*) – The predicted probabilities of the target outputted by the pipeline
- **objective** (*ObjectiveBase*) – The objective to threshold with. Must have a tunable threshold.

**property threshold** (*self*)  
Threshold used to make a prediction. Defaults to None.

classification\_pipeline

Module Contents

Classes Summary

<i>ClassificationPipeline</i>	Pipeline subclass for all classification pipelines.
-------------------------------	---

Contents

**class** evalml.pipelines.classification\_pipeline.**ClassificationPipeline** (*component\_graph*,  
*parameters=**None*,  
*custom\_name=**None*,  
*random\_seed=**0*)

Pipeline subclass for all classification pipelines.

Parameters

- **component\_graph** (*list or dict*) – List of components in order. Accepts strings or ComponentBase subclasses in the list. Note that when duplicate components are specified in a list, the duplicate component names will be modified with the component’s index in the list. For example, the component graph [Imputer, One Hot Encoder, Imputer, Logistic Regression Classifier] will have names [“Imputer”, “One Hot Encoder”, “Imputer\_2”, “Logistic Regression Classifier”]
- **parameters** (*dict*) – Dictionary with component names as keys and dictionary of that component’s parameters as values. An empty dictionary or None implies using all default values for component parameters. Defaults to None.
- **custom\_name** (*str*) – Custom name for the pipeline. Defaults to None.
- **random\_seed** (*int*) – Seed for the random number generator. Defaults to 0.

Attributes

<b>problem_type</b>	None
---------------------	------

Methods

<code>can_tune_threshold_with_objective</code>	Determine whether the threshold of a binary classification pipeline can be tuned.
<code>classes_</code>	Gets the class names for the problem.
<code>clone</code>	Constructs a new pipeline with the same components, parameters, and random state.
<code>compute_estimator_features</code>	Transforms the data by applying all pre-processing components.
<code>create_objectives</code>	
<code>custom_name</code>	Custom name of the pipeline.
<code>describe</code>	Outputs pipeline details including component parameters
<code>feature_importance</code>	Importance associated with each feature. Features dropped by the feature selection are excluded.
<code>fit</code>	Build a classification model. For string and categorical targets, classes are sorted
<code>get_component</code>	Returns component by name
<code>get_hyperparameter_ranges</code>	Returns hyperparameter ranges from all components as a dictionary.
<code>graph</code>	Generate an image representing the pipeline graph.
<code>graph_feature_importance</code>	Generate a bar graph of the pipeline's feature importance
<code>inverse_transform</code>	Apply component <code>inverse_transform</code> methods to estimator predictions in reverse order.
<code>load</code>	Loads pipeline at file path
<code>model_family</code>	Returns model family of this pipeline.
<code>name</code>	Name of the pipeline.
<code>new</code>	Constructs a new instance of the pipeline with the same component graph but with a different set of parameters.
<code>parameters</code>	Parameter dictionary for this pipeline.
<code>predict</code>	Make predictions using selected features.
<code>predict_proba</code>	Make probability estimates for labels.
<code>save</code>	Saves pipeline at file path
<code>score</code>	Evaluate model performance on objectives
<code>summary</code>	A short summary of the pipeline structure, describing the list of components used.

**`can_tune_threshold_with_objective`** (*self*, *objective*)

Determine whether the threshold of a binary classification pipeline can be tuned.

#### Parameters

- **`pipeline`** (*PipelineBase*) – Binary classification pipeline.
- **`objective`** – Primary AutoMLSearch objective.

**`property classes_`** (*self*)

Gets the class names for the problem.

**`clone`** (*self*)

Constructs a new pipeline with the same components, parameters, and random state.

**Returns** A new instance of this pipeline with identical components, parameters, and random state.

**compute\_estimator\_features** (*self*, *X*, *y=None*)

Transforms the data by applying all pre-processing components.

**Parameters** **X** (*pd.DataFrame*) – Input data to the pipeline to transform.

**Returns** New transformed features.

**Return type** *pd.DataFrame*

**static create\_objectives** (*objectives*)

**property custom\_name** (*self*)

Custom name of the pipeline.

**describe** (*self*, *return\_dict=False*)

Outputs pipeline details including component parameters

**Parameters** **return\_dict** (*bool*) – If True, return dictionary of information about pipeline.  
Defaults to False.

**Returns** Dictionary of all component parameters if *return\_dict* is True, else None

**Return type** *dict*

**property feature\_importance** (*self*)

Importance associated with each feature. Features dropped by the feature selection are excluded.

**Returns** *pd.DataFrame* including feature names and their corresponding importance

**fit** (*self*, *X*, *y*)

**Build a classification model. For string and categorical targets, classes are sorted** by `sorted(set(y))` and then are mapped to values between 0 and *n\_classes*-1.

**Parameters**

- **X** (*pd.DataFrame* or *np.ndarray*) – The input training data of shape [*n\_samples*, *n\_features*]
- **y** (*pd.Series*, *np.ndarray*) – The target training labels of length [*n\_samples*]

**Returns** *self*

**get\_component** (*self*, *name*)

Returns component by name

**Parameters** **name** (*str*) – Name of component

**Returns** Component to return

**Return type** *Component*

**get\_hyperparameter\_ranges** (*self*, *custom\_hyperparameters*)

Returns hyperparameter ranges from all components as a dictionary.

**Parameters** **custom\_hyperparameters** (*dict*) – Custom hyperparameters for the pipeline.

**Returns** Dictionary of hyperparameter ranges for each component in the pipeline.

**Return type** *dict*

**graph** (*self*, *filepath=None*)

Generate an image representing the pipeline graph.

**Parameters** `filepath` (*str*, *optional*) – Path to where the graph should be saved. If set to None (as by default), the graph will not be saved.

**Returns** Graph object that can be directly displayed in Jupyter notebooks.

**Return type** `graphviz.Digraph`

**graph\_feature\_importance** (*self*, *importance\_threshold=0*)

Generate a bar graph of the pipeline's feature importance

**Parameters** `importance_threshold` (*float*, *optional*) – If provided, graph features with a permutation importance whose absolute value is larger than `importance_threshold`. Defaults to zero.

**Returns** `plotly.Figure`, a bar graph showing features and their corresponding importance

**inverse\_transform** (*self*, *y*)

Apply component `inverse_transform` methods to estimator predictions in reverse order.

Components that implement `inverse_transform` are `PolynomialDetrender`, `LabelEncoder` (tbd).

**Parameters** `y` (*pd.Series*) – Final component features

**static load** (*file\_path*)

Loads pipeline at file path

**Parameters** `file_path` (*str*) – location to load file

**Returns** `PipelineBase` object

**property model\_family** (*self*)

Returns model family of this pipeline.

**property name** (*self*)

Name of the pipeline.

**new** (*self*, *parameters*, *random\_seed=0*)

**Constructs a new instance of the pipeline with the same component graph but with a different set of parameters.**

Not to be confused with python's `__new__` method.

**Parameters**

- **parameters** (*dict*) – Dictionary with component names as keys and dictionary of that component's parameters as values. An empty dictionary or None implies using all default values for component parameters. Defaults to None.
- **random\_seed** (*int*) – Seed for the random number generator. Defaults to 0.

**Returns** A new instance of this pipeline with identical components.

**property parameters** (*self*)

Parameter dictionary for this pipeline.

**Returns** Dictionary of all component parameters.

**Return type** `dict`

**predict** (*self*, *X*, *objective=None*)

Make predictions using selected features.

**Parameters**

- **X** (*pd.DataFrame*, or *np.ndarray*) – Data of shape `[n_samples, n_features]`
- **objective** (*Object* or *string*) – The objective to use to make predictions

**Returns** Estimated labels

**Return type** `pd.Series`

**predict\_proba** (*self*, *X*)

Make probability estimates for labels.

**Parameters** **X** (*pd.DataFrame* or *np.ndarray*) – Data of shape [n\_samples, n\_features]

**Returns** Probability estimates

**Return type** `pd.DataFrame`

**save** (*self*, *file\_path*, *pickle\_protocol=cloudpickle.DEFAULT\_PROTOCOL*)

Saves pipeline at file path

**Parameters**

- **file\_path** (*str*) – location to save file
- **pickle\_protocol** (*int*) – the pickle data stream format.

**Returns** None

**score** (*self*, *X*, *y*, *objectives*)

Evaluate model performance on objectives

**Parameters**

- **X** (*pd.DataFrame* or *np.ndarray*) – Data of shape [n\_samples, n\_features]
- **y** (*pd.Series*, or *np.ndarray*) – True labels of length [n\_samples]
- **objectives** (*list*) – List of objectives to score

**Returns** Ordered dictionary of objective scores

**Return type** `dict`

**property summary** (*self*)

A short summary of the pipeline structure, describing the list of components used. Example: Logistic Regression Classifier w/ Simple Imputer + One Hot Encoder

## component\_graph

## Module Contents

## Classes Summary

---

*ComponentGraph*

Component graph for a pipeline as a directed acyclic graph (DAG).

---

## Attributes Summary

---

`logger`

---

## Contents

**class** evalml.pipelines.component\_graph.**ComponentGraph**(*component\_dict=None, random\_seed=0*)

Component graph for a pipeline as a directed acyclic graph (DAG).

### Parameters

- **component\_dict** (*dict*) – A dictionary which specifies the components and edges between components that should be used to create the component graph. Defaults to None.
- **random\_seed** (*int*) – Seed for the random number generator. Defaults to 0.

### Example

```
>>> component_dict = {'imputer': ['Imputer'], 'ohe': ['One Hot Encoder', 'imputer.
↳x'], 'estimator_1': ['Random Forest Classifier', 'ohe.x'], 'estimator_2': [
↳'Decision Tree Classifier', 'ohe.x'], 'final': ['Logistic Regression Classifier
↳', 'estimator_1', 'estimator_2']}
>>> component_graph = ComponentGraph(component_dict)
```

### Methods

<code>compute_final_component_features</code>	Transform all components save the final one, and gathers the data from any number of parents
<code>compute_order</code>	The order that components will be computed or called in.
<code>default_parameters</code>	The default parameter dictionary for this pipeline.
<code>describe</code>	Outputs component graph details including component parameters
<code>fit</code>	Fit each component in the graph.
<code>fit_features</code>	Fit all components save the final one, usually an estimator.
<code>generate_order</code>	Regenerated the topologically sorted order of the graph
<code>get_component</code>	Retrieves a single component object from the graph.
<code>get_estimators</code>	Gets a list of all the estimator components within this graph.
<code>get_inputs</code>	Retrieves all inputs for a given component.
<code>get_last_component</code>	Retrieves the component that is computed last in the graph, usually the final estimator.
<code>graph</code>	Generate an image representing the component graph
<code>instantiate</code>	Instantiates all uninstantiated components within the graph using the given parameters. An error will be

continues on next page

Table 610 – continued from previous page

<code>inverse_transform</code>	Apply component <code>inverse_transform</code> methods to estimator predictions in reverse order.
<code>predict</code>	Make predictions using selected features.

**compute\_final\_component\_features** (*self*, *X*, *y=None*)

Transform all components save the final one, and gathers the data from any number of parents to get all the information that should be fed to the final component.

**Parameters**

- **X** (*pd.DataFrame*) – Data of shape [n\_samples, n\_features].
- **y** (*pd.Series*) – The target training data of length [n\_samples]. Defaults to None.

**Returns** Transformed values.

**Return type** *pd.DataFrame*

**property compute\_order** (*self*)

The order that components will be computed or called in.

**property default\_parameters** (*self*)

The default parameter dictionary for this pipeline.

**Returns** Dictionary of all component default parameters.

**Return type** dict

**describe** (*self*, *return\_dict=False*)

Outputs component graph details including component parameters

**Parameters** **return\_dict** (*bool*) – If True, return dictionary of information about component graph. Defaults to False.

**Returns** Dictionary of all component parameters if `return_dict` is True, else None

**Return type** dict

**fit** (*self*, *X*, *y*)

Fit each component in the graph.

**Parameters**

- **X** (*pd.DataFrame*) – The input training data of shape [n\_samples, n\_features].
- **y** (*pd.Series*) – The target training data of length [n\_samples].

**fit\_features** (*self*, *X*, *y*)

Fit all components save the final one, usually an estimator.

**Parameters**

- **X** (*pd.DataFrame*) – The input training data of shape [n\_samples, n\_features].
- **y** (*pd.Series*) – The target training data of length [n\_samples].

**Returns** Transformed values.

**Return type** *pd.DataFrame*

**classmethod generate\_order** (*cls*, *component\_dict*)

Regenerated the topologically sorted order of the graph

**get\_component** (*self*, *component\_name*)

Retrieves a single component object from the graph.



**Parameters** `component_name` (*str*) – Name of the component to retrieve

**Returns** ComponentBase object

**get\_estimators** (*self*)

Gets a list of all the estimator components within this graph.

**Returns** All estimator objects within the graph.

**Return type** list

**get\_inputs** (*self*, *component\_name*)

Retrieves all inputs for a given component.

**Parameters** `component_name` (*str*) – Name of the component to look up.

**Returns** List of inputs for the component to use.

**Return type** list[str]

**get\_last\_component** (*self*)

Retrieves the component that is computed last in the graph, usually the final estimator.

**Returns** ComponentBase object

**graph** (*self*, *name=None*, *graph\_format=None*)

Generate an image representing the component graph

**Parameters**

- **name** (*str*) – Name of the graph. Defaults to None.
- **graph\_format** (*str*) – file format to save the graph in. Defaults to None.

**Returns** Graph object that can be directly displayed in Jupyter notebooks.

**Return type** graphviz.Digraph

**instantiate** (*self*, *parameters*)

Instantiates all uninstantiated components within the graph using the given parameters. An error will be raised if a component is already instantiated but the parameters dict contains arguments for that component.

**Parameters** `parameters` (*dict*) – Dictionary with component names as keys and dictionary of that component's parameters as values. An empty dictionary {} or None implies using all default values for component parameters.

**inverse\_transform** (*self*, *y*)

Apply component inverse\_transform methods to estimator predictions in reverse order.

Components that implement inverse\_transform are PolynomialDetrender, LabelEncoder (tbd).

**Parameters** `y` – (pd.Series): Final component features

**predict** (*self*, *X*)

Make predictions using selected features.

**Parameters** `X` (*pd.DataFrame*) – Data of shape [n\_samples, n\_features].

**Returns** Predicted values.

**Return type** pd.Series

evalml.pipelines.component\_graph.logger

multiclass\_classification\_pipeline

Module Contents

Classes Summary

<i>MulticlassClassificationPipeline</i>	Pipeline subclass for all multiclass classification pipelines.
---	--

Contents

`class evalml.pipelines.multiclass_classification_pipeline.MulticlassClassificationPipeline`

Pipeline subclass for all multiclass classification pipelines.

Parameters

- **component\_graph** (*list or dict*) – List of components in order. Accepts strings or ComponentBase subclasses in the list. Note that when duplicate components are specified in a list, the duplicate component names will be modified with the component’s index in the list. For example, the component graph [Imputer, One Hot Encoder, Imputer, Logistic Regression Classifier] will have names [“Imputer”, “One Hot Encoder”, “Imputer\_2”, “Logistic Regression Classifier”]
- **parameters** (*dict*) – Dictionary with component names as keys and dictionary of that component’s parameters as values. An empty dictionary or None implies using all default values for component parameters. Defaults to None.
- **custom\_name** (*str*) – Custom name for the pipeline. Defaults to None.
- **random\_seed** (*int*) – Seed for the random number generator. Defaults to 0.

Attributes

<b>problem_type</b>	ProblemTypes.MULTICLASS
---------------------	-------------------------

Methods

<i>can_tune_threshold_with_objective</i>	Determine whether the threshold of a binary classification pipeline can be tuned.
<i>classes_</i>	Gets the class names for the problem.
<i>clone</i>	Constructs a new pipeline with the same components, parameters, and random state.

continues on next page

Table 612 – continued from previous page

<code>compute_estimator_features</code>	Transforms the data by applying all pre-processing components.
<code>create_objectives</code>	
<code>custom_name</code>	Custom name of the pipeline.
<code>describe</code>	Outputs pipeline details including component parameters
<code>feature_importance</code>	Importance associated with each feature. Features dropped by the feature selection are excluded.
<code>fit</code>	Build a classification model. For string and categorical targets, classes are sorted
<code>get_component</code>	Returns component by name
<code>get_hyperparameter_ranges</code>	Returns hyperparameter ranges from all components as a dictionary.
<code>graph</code>	Generate an image representing the pipeline graph.
<code>graph_feature_importance</code>	Generate a bar graph of the pipeline's feature importance
<code>inverse_transform</code>	Apply component <code>inverse_transform</code> methods to estimator predictions in reverse order.
<code>load</code>	Loads pipeline at file path
<code>model_family</code>	Returns model family of this pipeline.
<code>name</code>	Name of the pipeline.
<code>new</code>	Constructs a new instance of the pipeline with the same component graph but with a different set of parameters.
<code>parameters</code>	Parameter dictionary for this pipeline.
<code>predict</code>	Make predictions using selected features.
<code>predict_proba</code>	Make probability estimates for labels.
<code>save</code>	Saves pipeline at file path
<code>score</code>	Evaluate model performance on objectives
<code>summary</code>	A short summary of the pipeline structure, describing the list of components used.

**can\_tune\_threshold\_with\_objective** (*self*, *objective*)

Determine whether the threshold of a binary classification pipeline can be tuned.

#### Parameters

- **pipeline** (*PipelineBase*) – Binary classification pipeline.
- **objective** – Primary AutoMLSearch objective.

**property classes\_** (*self*)

Gets the class names for the problem.

**clone** (*self*)

Constructs a new pipeline with the same components, parameters, and random state.

**Returns** A new instance of this pipeline with identical components, parameters, and random state.

**compute\_estimator\_features** (*self*, *X*, *y=None*)

Transforms the data by applying all pre-processing components.

**Parameters** **x** (*pd.DataFrame*) – Input data to the pipeline to transform.

**Returns** New transformed features.

**Return type** `pd.DataFrame`

**static** `create_objectives` (*objectives*)

**property** `custom_name` (*self*)

Custom name of the pipeline.

**describe** (*self*, *return\_dict=False*)

Outputs pipeline details including component parameters

**Parameters** `return_dict` (*bool*) – If True, return dictionary of information about pipeline.  
Defaults to False.

**Returns** Dictionary of all component parameters if `return_dict` is True, else None

**Return type** `dict`

**property** `feature_importance` (*self*)

Importance associated with each feature. Features dropped by the feature selection are excluded.

**Returns** `pd.DataFrame` including feature names and their corresponding importance

**fit** (*self*, *X*, *y*)

**Build a classification model. For string and categorical targets, classes are sorted** by `sorted(set(y))`  
and then are mapped to values between 0 and `n_classes-1`.

**Parameters**

- **X** (*pd.DataFrame* or *np.ndarray*) – The input training data of shape `[n_samples, n_features]`
- **y** (*pd.Series*, *np.ndarray*) – The target training labels of length `[n_samples]`

**Returns** *self*

**get\_component** (*self*, *name*)

Returns component by name

**Parameters** `name` (*str*) – Name of component

**Returns** Component to return

**Return type** Component

**get\_hyperparameter\_ranges** (*self*, *custom\_hyperparameters*)

Returns hyperparameter ranges from all components as a dictionary.

**Parameters** `custom_hyperparameters` (*dict*) – Custom hyperparameters for the pipeline.

**Returns** Dictionary of hyperparameter ranges for each component in the pipeline.

**Return type** `dict`

**graph** (*self*, *filepath=None*)

Generate an image representing the pipeline graph.

**Parameters** `filepath` (*str*, *optional*) – Path to where the graph should be saved. If set to None (as by default), the graph will not be saved.

**Returns** Graph object that can be directly displayed in Jupyter notebooks.

**Return type** `graphviz.Digraph`

**graph\_feature\_importance** (*self*, *importance\_threshold=0*)

Generate a bar graph of the pipeline's feature importance

**Parameters** **importance\_threshold** (*float*, *optional*) – If provided, graph features with a permutation importance whose absolute value is larger than importance\_threshold. Defaults to zero.

**Returns** *plotly.Figure*, a bar graph showing features and their corresponding importance

**inverse\_transform** (*self*, *y*)

Apply component inverse\_transform methods to estimator predictions in reverse order.

Components that implement inverse\_transform are PolynomialDetrender, LabelEncoder (tbd).

**Parameters** **y** (*pd.Series*) – Final component features

**static load** (*file\_path*)

Loads pipeline at file path

**Parameters** **file\_path** (*str*) – location to load file

**Returns** PipelineBase object

**property model\_family** (*self*)

Returns model family of this pipeline.

**property name** (*self*)

Name of the pipeline.

**new** (*self*, *parameters*, *random\_seed=0*)

**Constructs a new instance of the pipeline with the same component graph but with a different set of parameters.**

Not to be confused with python's `__new__` method.

**Parameters**

- **parameters** (*dict*) – Dictionary with component names as keys and dictionary of that component's parameters as values. An empty dictionary or None implies using all default values for component parameters. Defaults to None.
- **random\_seed** (*int*) – Seed for the random number generator. Defaults to 0.

**Returns** A new instance of this pipeline with identical components.

**property parameters** (*self*)

Parameter dictionary for this pipeline.

**Returns** Dictionary of all component parameters.

**Return type** *dict*

**predict** (*self*, *X*, *objective=None*)

Make predictions using selected features.

**Parameters**

- **X** (*pd.DataFrame*, or *np.ndarray*) – Data of shape [n\_samples, n\_features]
- **objective** (*Object* or *string*) – The objective to use to make predictions

**Returns** Estimated labels

**Return type** *pd.Series*

**predict\_proba** (*self*, *X*)

Make probability estimates for labels.

**Parameters** **X** (*pd.DataFrame* or *np.ndarray*) – Data of shape [n\_samples, n\_features]

**Returns** Probability estimates

**Return type** *pd.DataFrame*

**save** (*self*, *file\_path*, *pickle\_protocol=cloudpickle.DEFAULT\_PROTOCOL*)

Saves pipeline at file path

**Parameters**

- **file\_path** (*str*) – location to save file
- **pickle\_protocol** (*int*) – the pickle data stream format.

**Returns** None

**score** (*self*, *X*, *y*, *objectives*)

Evaluate model performance on objectives

**Parameters**

- **X** (*pd.DataFrame* or *np.ndarray*) – Data of shape [n\_samples, n\_features]
- **y** (*pd.Series*, or *np.ndarray*) – True labels of length [n\_samples]
- **objectives** (*list*) – List of objectives to score

**Returns** Ordered dictionary of objective scores

**Return type** *dict*

**property summary** (*self*)

A short summary of the pipeline structure, describing the list of components used. Example: Logistic Regression Classifier w/ Simple Imputer + One Hot Encoder

## pipeline\_base

### Module Contents

### Classes Summary

---

*PipelineBase*

Machine learning pipeline made out of transformers and a estimator.

---

### Attributes Summary

---

*logger*

---

## Contents

evalml.pipelines.pipeline\_base.logger

**class** evalml.pipelines.pipeline\_base.PipelineBase(*component\_graph*, *parameters=None*, *custom\_name=None*, *random\_seed=0*)

Machine learning pipeline made out of transformers and a estimator.

### Parameters

- **component\_graph** (*list or dict*) – List of components in order. Accepts strings or ComponentBase subclasses in the list. Note that when duplicate components are specified in a list, the duplicate component names will be modified with the component’s index in the list. For example, the component graph [Imputer, One Hot Encoder, Imputer, Logistic Regression Classifier] will have names [“Imputer”, “One Hot Encoder”, “Imputer\_2”, “Logistic Regression Classifier”].
- **parameters** (*dict*) – Dictionary with component names as keys and dictionary of that component’s parameters as values. An empty dictionary or None implies using all default values for component parameters. Defaults to None.
- **custom\_name** (*str*) – Custom name for the pipeline. Defaults to None.
- **random\_seed** (*int*) – Seed for the random number generator. Defaults to 0.

### Attributes

<b>problem_type</b>	None
---------------------	------

### Methods

<i>can_tune_threshold_with_objective</i>	Determine whether the threshold of a binary classification pipeline can be tuned.
<i>clone</i>	Constructs a new pipeline with the same components, parameters, and random state.
<i>compute_estimator_features</i>	Transforms the data by applying all pre-processing components.
<i>create_objectives</i>	
<i>custom_name</i>	Custom name of the pipeline.
<i>describe</i>	Outputs pipeline details including component parameters
<i>feature_importance</i>	Importance associated with each feature. Features dropped by the feature selection are excluded.
<i>fit</i>	Build a model.
<i>get_component</i>	Returns component by name
<i>get_hyperparameter_ranges</i>	Returns hyperparameter ranges from all components as a dictionary.
<i>graph</i>	Generate an image representing the pipeline graph.
<i>graph_feature_importance</i>	Generate a bar graph of the pipeline’s feature importance
<i>inverse_transform</i>	Apply component inverse_transform methods to estimator predictions in reverse order.

continues on next page

Table 615 – continued from previous page

<code>load</code>	Loads pipeline at file path
<code>model_family</code>	Returns model family of this pipeline.
<code>name</code>	Name of the pipeline.
<code>new</code>	Constructs a new instance of the pipeline with the same component graph but with a different set of parameters.
<code>parameters</code>	Parameter dictionary for this pipeline.
<code>predict</code>	Make predictions using selected features.
<code>save</code>	Saves pipeline at file path
<code>score</code>	Evaluate model performance on current and additional objectives.
<code>summary</code>	A short summary of the pipeline structure, describing the list of components used.

**can\_tune\_threshold\_with\_objective** (*self*, *objective*)

Determine whether the threshold of a binary classification pipeline can be tuned.

**Parameters**

- **pipeline** (`PipelineBase`) – Binary classification pipeline.
- **objective** – Primary AutoMLSearch objective.

**clone** (*self*)

Constructs a new pipeline with the same components, parameters, and random state.

**Returns** A new instance of this pipeline with identical components, parameters, and random state.

**compute\_estimator\_features** (*self*, *X*, *y=None*)

Transforms the data by applying all pre-processing components.

**Parameters** **X** (`pd.DataFrame`) – Input data to the pipeline to transform.

**Returns** New transformed features.

**Return type** `pd.DataFrame`

**static create\_objectives** (*objectives*)

**property custom\_name** (*self*)

Custom name of the pipeline.

**describe** (*self*, *return\_dict=False*)

Outputs pipeline details including component parameters

**Parameters** **return\_dict** (`bool`) – If True, return dictionary of information about pipeline. Defaults to False.

**Returns** Dictionary of all component parameters if `return_dict` is True, else None

**Return type** `dict`

**property feature\_importance** (*self*)

Importance associated with each feature. Features dropped by the feature selection are excluded.

**Returns** `pd.DataFrame` including feature names and their corresponding importance

**abstract fit** (*self*, *X*, *y*)

Build a model.

**Parameters**



- **X** (*pd.DataFrame* or *np.ndarray*) – The input training data of shape [n\_samples, n\_features].
- **y** (*pd.Series*, *np.ndarray*) – The target training data of length [n\_samples].

**Returns** self

**get\_component** (*self*, *name*)

Returns component by name

**Parameters** **name** (*str*) – Name of component

**Returns** Component to return

**Return type** Component

**get\_hyperparameter\_ranges** (*self*, *custom\_hyperparameters*)

Returns hyperparameter ranges from all components as a dictionary.

**Parameters** **custom\_hyperparameters** (*dict*) – Custom hyperparameters for the pipeline.

**Returns** Dictionary of hyperparameter ranges for each component in the pipeline.

**Return type** dict

**graph** (*self*, *filepath=None*)

Generate an image representing the pipeline graph.

**Parameters** **filepath** (*str*, *optional*) – Path to where the graph should be saved. If set to None (as by default), the graph will not be saved.

**Returns** Graph object that can be directly displayed in Jupyter notebooks.

**Return type** graphviz.Digraph

**graph\_feature\_importance** (*self*, *importance\_threshold=0*)

Generate a bar graph of the pipeline's feature importance

**Parameters** **importance\_threshold** (*float*, *optional*) – If provided, graph features with a permutation importance whose absolute value is larger than importance\_threshold. Defaults to zero.

**Returns** plotly.Figure, a bar graph showing features and their corresponding importance

**inverse\_transform** (*self*, *y*)

Apply component inverse\_transform methods to estimator predictions in reverse order.

Components that implement inverse\_transform are PolynomialDetrender, LabelEncoder (tbd).

**Parameters** **y** (*pd.Series*) – Final component features

**static load** (*file\_path*)

Loads pipeline at file path

**Parameters** **file\_path** (*str*) – location to load file

**Returns** PipelineBase object

**property model\_family** (*self*)

Returns model family of this pipeline.

**property name** (*self*)

Name of the pipeline.

**new** (*self*, *parameters*, *random\_seed=0*)

Constructs a new instance of the pipeline with the same component graph but with a different set of parameters. Not to be confused with python's `__new__` method.

**Parameters**

- **parameters** (*dict*) – Dictionary with component names as keys and dictionary of that component's parameters as values. An empty dictionary or None implies using all default values for component parameters. Defaults to None.
- **random\_seed** (*int*) – Seed for the random number generator. Defaults to 0.

**Returns** A new instance of this pipeline with identical components.

**property parameters** (*self*)

Parameter dictionary for this pipeline.

**Returns** Dictionary of all component parameters.

**Return type** dict

**predict** (*self*, *X*, *objective=None*)

Make predictions using selected features.

**Parameters**

- **X** (*pd.DataFrame*, or *np.ndarray*) – Data of shape [n\_samples, n\_features].
- **objective** (*Object* or *string*) – The objective to use to make predictions.

**Returns** Predicted values.

**Return type** pd.Series

**save** (*self*, *file\_path*, *pickle\_protocol=cloudpickle.DEFAULT\_PROTOCOL*)

Saves pipeline at file path

**Parameters**

- **file\_path** (*str*) – location to save file
- **pickle\_protocol** (*int*) – the pickle data stream format.

**Returns** None

**abstract score** (*self*, *X*, *y*, *objectives*)

Evaluate model performance on current and additional objectives.

**Parameters**

- **X** (*pd.DataFrame* or *np.ndarray*) – Data of shape [n\_samples, n\_features].
- **y** (*pd.Series*, *np.ndarray*) – True labels of length [n\_samples].
- **objectives** (*list*) – Non-empty list of objectives to score on.

**Returns** Ordered dictionary of objective scores.

**Return type** dict

**property summary** (*self*)

A short summary of the pipeline structure, describing the list of components used. Example: Logistic Regression Classifier w/ Simple Imputer + One Hot Encoder

## pipeline\_meta

### Module Contents

### Classes Summary

<i>PipelineBaseMeta</i>	Metaclass that overrides creating a new pipeline by wrapping methods with validators and setters
<i>TimeSeriesPipelineBaseMeta</i>	Metaclass that overrides creating a new time series pipeline by wrapping methods with validators and setters

### Contents

**class** evalml.pipelines.pipeline\_meta.**PipelineBaseMeta**

Metaclass that overrides creating a new pipeline by wrapping methods with validators and setters

#### Attributes

<b>FIT_METHODS</b>	['fit', 'fit_transform']
<b>METHODS_TO_CHECK</b>	['predict', 'predict_proba', 'transform', 'inverse_transform']
<b>PROPERTIES_TO_CHECK</b>	['feature_importance']

#### Methods

<i>check_for_fit</i>	<i>check_for_fit</i> wraps a method that validates if <i>self.is_fitted</i> is <i>True</i> .
<i>register</i>	Register a virtual subclass of an ABC.
<i>set_fit</i>	

**classmethod** **check\_for\_fit** (*cls*, *method*)

*check\_for\_fit* wraps a method that validates if *self.is\_fitted* is *True*. It raises an exception if *False* and calls and returns the wrapped method if *True*.

**register** (*cls*, *subclass*)

Register a virtual subclass of an ABC.

Returns the subclass, to allow usage as a class decorator.

**classmethod** **set\_fit** (*cls*, *method*)

**class** evalml.pipelines.pipeline\_meta.**TimeSeriesPipelineBaseMeta**

Metaclass that overrides creating a new time series pipeline by wrapping methods with validators and setters

#### Attributes

<b>FIT_METHODS</b>	['fit', 'fit_transform']
<b>METHODS_TO_CHECK</b>	['predict', 'predict_proba', 'transform', 'inverse_transform']
<b>PROPERTIES_TO_CHECK</b>	['feature_importance']

## Methods

<code>check_for_fit</code>	<code>check_for_fit</code> wraps a method that validates if <code>self._is_fitted</code> is <code>True</code> .
<code>register</code>	Register a virtual subclass of an ABC.
<code>set_fit</code>	

**classmethod** `check_for_fit` (*cls*, *method*)

`check_for_fit` wraps a method that validates if `self._is_fitted` is `True`. It raises an exception if `False` and calls and returns the wrapped method if `True`.

**register** (*cls*, *subclass*)

Register a virtual subclass of an ABC.

Returns the subclass, to allow usage as a class decorator.

**classmethod** `set_fit` (*cls*, *method*)

## regression\_pipeline

### Module Contents

### Classes Summary

<code>RegressionPipeline</code>	Pipeline subclass for all regression pipelines.
---------------------------------	---

## Contents

**class** `evalml.pipelines.regression_pipeline.RegressionPipeline` (*component\_graph*, *parameters=None*, *custom\_name=None*, *random\_seed=0*)

Pipeline subclass for all regression pipelines.

### Parameters

- **component\_graph** (*list* or *dict*) – List of components in order. Accepts strings or `ComponentBase` subclasses in the list. Note that when duplicate components are specified in a list, the duplicate component names will be modified with the component's index in the list. For example, the component graph [Imputer, One Hot Encoder, Imputer, Logistic Regression Classifier] will have names ["Imputer", "One Hot Encoder", "Imputer\_2", "Logistic Regression Classifier"]
- **parameters** (*dict*) – Dictionary with component names as keys and dictionary of that

component's parameters as values. An empty dictionary or None implies using all default values for component parameters. Defaults to None.

- **custom\_name** (*str*) – Custom name for the pipeline. Defaults to None.
- **random\_seed** (*int*) – Seed for the random number generator. Defaults to 0.

#### Attributes

<b>problem_type</b>	ProblemTypes.REGRESSION
---------------------	-------------------------

#### Methods

<i>can_tune_threshold_with_objective</i>	Determine whether the threshold of a binary classification pipeline can be tuned.
<i>clone</i>	Constructs a new pipeline with the same components, parameters, and random state.
<i>compute_estimator_features</i>	Transforms the data by applying all pre-processing components.
<i>create_objectives</i>	
<i>custom_name</i>	Custom name of the pipeline.
<i>describe</i>	Outputs pipeline details including component parameters
<i>feature_importance</i>	Importance associated with each feature. Features dropped by the feature selection are excluded.
<i>fit</i>	Build a regression model.
<i>get_component</i>	Returns component by name
<i>get_hyperparameter_ranges</i>	Returns hyperparameter ranges from all components as a dictionary.
<i>graph</i>	Generate an image representing the pipeline graph.
<i>graph_feature_importance</i>	Generate a bar graph of the pipeline's feature importance
<i>inverse_transform</i>	Apply component <i>inverse_transform</i> methods to estimator predictions in reverse order.
<i>load</i>	Loads pipeline at file path
<i>model_family</i>	Returns model family of this pipeline.
<i>name</i>	Name of the pipeline.
<i>new</i>	Constructs a new instance of the pipeline with the same component graph but with a different set of parameters.
<i>parameters</i>	Parameter dictionary for this pipeline.
<i>predict</i>	Make predictions using selected features.
<i>save</i>	Saves pipeline at file path
<i>score</i>	Evaluate model performance on current and additional objectives
<i>summary</i>	A short summary of the pipeline structure, describing the list of components used.

**can\_tune\_threshold\_with\_objective** (*self*, *objective*)

Determine whether the threshold of a binary classification pipeline can be tuned.

#### Parameters

- **pipeline** (*PipelineBase*) – Binary classification pipeline.
- **objective** – Primary AutoMLSearch objective.

**clone** (*self*)

Constructs a new pipeline with the same components, parameters, and random state.

**Returns** A new instance of this pipeline with identical components, parameters, and random state.

**compute\_estimator\_features** (*self, X, y=None*)

Transforms the data by applying all pre-processing components.

**Parameters** **X** (*pd.DataFrame*) – Input data to the pipeline to transform.

**Returns** New transformed features.

**Return type** *pd.DataFrame*

**static create\_objectives** (*objectives*)

**property custom\_name** (*self*)

Custom name of the pipeline.

**describe** (*self, return\_dict=False*)

Outputs pipeline details including component parameters

**Parameters** **return\_dict** (*bool*) – If True, return dictionary of information about pipeline. Defaults to False.

**Returns** Dictionary of all component parameters if *return\_dict* is True, else None

**Return type** *dict*

**property feature\_importance** (*self*)

Importance associated with each feature. Features dropped by the feature selection are excluded.

**Returns** *pd.DataFrame* including feature names and their corresponding importance

**fit** (*self, X, y*)

Build a regression model.

**Parameters**

- **X** (*pd.DataFrame* or *np.ndarray*) – The input training data of shape [n\_samples, n\_features]
- **y** (*pd.Series, np.ndarray*) – The target training data of length [n\_samples]

**Returns** *self*

**get\_component** (*self, name*)

Returns component by name

**Parameters** **name** (*str*) – Name of component

**Returns** Component to return

**Return type** Component

**get\_hyperparameter\_ranges** (*self, custom\_hyperparameters*)

Returns hyperparameter ranges from all components as a dictionary.

**Parameters** **custom\_hyperparameters** (*dict*) – Custom hyperparameters for the pipeline.

**Returns** Dictionary of hyperparameter ranges for each component in the pipeline.

**Return type** dict

**graph** (*self*, *filepath=None*)

Generate an image representing the pipeline graph.

**Parameters** **filepath** (*str*, *optional*) – Path to where the graph should be saved. If set to None (as by default), the graph will not be saved.

**Returns** Graph object that can be directly displayed in Jupyter notebooks.

**Return type** graphviz.Digraph

**graph\_feature\_importance** (*self*, *importance\_threshold=0*)

Generate a bar graph of the pipeline's feature importance

**Parameters** **importance\_threshold** (*float*, *optional*) – If provided, graph features with a permutation importance whose absolute value is larger than importance\_threshold. Defaults to zero.

**Returns** plotly.Figure, a bar graph showing features and their corresponding importance

**inverse\_transform** (*self*, *y*)

Apply component inverse\_transform methods to estimator predictions in reverse order.

Components that implement inverse\_transform are PolynomialDetrender, LabelEncoder (tbd).

**Parameters** **y** (*pd.Series*) – Final component features

**static load** (*file\_path*)

Loads pipeline at file path

**Parameters** **file\_path** (*str*) – location to load file

**Returns** PipelineBase object

**property model\_family** (*self*)

Returns model family of this pipeline.

**property name** (*self*)

Name of the pipeline.

**new** (*self*, *parameters*, *random\_seed=0*)

**Constructs a new instance of the pipeline with the same component graph but with a different set of parameters.**

Not to be confused with python's \_\_new\_\_ method.

**Parameters**

- **parameters** (*dict*) – Dictionary with component names as keys and dictionary of that component's parameters as values. An empty dictionary or None implies using all default values for component parameters. Defaults to None.
- **random\_seed** (*int*) – Seed for the random number generator. Defaults to 0.

**Returns** A new instance of this pipeline with identical components.

**property parameters** (*self*)

Parameter dictionary for this pipeline.

**Returns** Dictionary of all component parameters.

**Return type** dict

**predict** (*self*, *X*, *objective=None*)

Make predictions using selected features.

#### Parameters

- **X** (*pd.DataFrame*, or *np.ndarray*) – Data of shape [n\_samples, n\_features].
- **objective** (*Object* or *string*) – The objective to use to make predictions.

**Returns** Predicted values.

**Return type** *pd.Series*

**save** (*self*, *file\_path*, *pickle\_protocol=cloudpickle.DEFAULT\_PROTOCOL*)

Saves pipeline at file path

#### Parameters

- **file\_path** (*str*) – location to save file
- **pickle\_protocol** (*int*) – the pickle data stream format.

**Returns** None

**score** (*self*, *X*, *y*, *objectives*)

Evaluate model performance on current and additional objectives

#### Parameters

- **X** (*pd.DataFrame*, or *np.ndarray*) – Data of shape [n\_samples, n\_features]
- **y** (*pd.Series*, or *np.ndarray*) – True values of length [n\_samples]
- **objectives** (*list*) – Non-empty list of objectives to score on

**Returns** Ordered dictionary of objective scores

**Return type** *dict*

**property summary** (*self*)

A short summary of the pipeline structure, describing the list of components used. Example: Logistic Regression Classifier w/ Simple Imputer + One Hot Encoder

## time\_series\_classification\_pipelines

### Module Contents

### Classes Summary

<i>TimeSeriesBinaryClassificationPipeline</i>	Pipeline base class for time series binary classification problems.
<i>TimeSeriesClassificationPipeline</i>	Pipeline base class for time series classification problems.
<i>TimeSeriesMulticlassClassificationPipeline</i>	Pipeline base class for time series multiclass classification problems.



## Contents

**class** evalml.pipelines.time\_series\_classification\_pipelines.**TimeSeriesBinaryClassification**

Pipeline base class for time series binary classification problems.

### Parameters

- **component\_graph** (*list or dict*) – List of components in order. Accepts strings or ComponentBase subclasses in the list. Note that when duplicate components are specified in a list, the duplicate component names will be modified with the component’s index in the list. For example, the component graph [Imputer, One Hot Encoder, Imputer, Logistic Regression Classifier] will have names [“Imputer”, “One Hot Encoder”, “Imputer\_2”, “Logistic Regression Classifier”]
- **parameters** (*dict*) – Dictionary with component names as keys and dictionary of that component’s parameters as values. An empty dictionary {} implies using all default values for component parameters. Pipeline-level parameters such as date\_index, gap, and max\_delay must be specified with the “pipeline” key. For example: Pipeline(parameters={“pipeline”: {“date\_index”: “Date”, “max\_delay”: 4, “gap”: 2}}).
- **random\_seed** (*int*) – Seed for the random number generator. Defaults to 0.

### Attributes

<b>problem_type</b>	None
---------------------	------

### Methods

<i>can_tune_threshold_with_objective</i>	Determine whether the threshold of a binary classification pipeline can be tuned.
<i>classes_</i>	Gets the class names for the problem.
<i>clone</i>	Constructs a new pipeline with the same components, parameters, and random state.
<i>compute_estimator_features</i>	Transforms the data by applying all pre-processing components.
<i>create_objectives</i>	
<i>custom_name</i>	Custom name of the pipeline.
<i>describe</i>	Outputs pipeline details including component parameters
<i>feature_importance</i>	Importance associated with each feature. Features dropped by the feature selection are excluded.
<i>fit</i>	Fit a time series classification pipeline.
<i>get_component</i>	Returns component by name

continues on next page

Table 622 – continued from previous page

<code>get_hyperparameter_ranges</code>	Returns hyperparameter ranges from all components as a dictionary.
<code>graph</code>	Generate an image representing the pipeline graph.
<code>graph_feature_importance</code>	Generate a bar graph of the pipeline’s feature importance
<code>inverse_transform</code>	Apply component <code>inverse_transform</code> methods to estimator predictions in reverse order.
<code>load</code>	Loads pipeline at file path
<code>model_family</code>	Returns model family of this pipeline.
<code>name</code>	Name of the pipeline.
<code>new</code>	Constructs a new instance of the pipeline with the same component graph but with a different set of parameters.
<code>optimize_threshold</code>	Optimize the pipeline threshold given the objective to use. Only used for binary problems with objectives whose thresholds can be tuned.
<code>parameters</code>	Parameter dictionary for this pipeline.
<code>predict</code>	Make predictions using selected features.
<code>predict_proba</code>	Make probability estimates for labels.
<code>save</code>	Saves pipeline at file path
<code>score</code>	Evaluate model performance on current and additional objectives.
<code>summary</code>	A short summary of the pipeline structure, describing the list of components used.
<code>threshold</code>	Threshold used to make a prediction. Defaults to None.

**`can_tune_threshold_with_objective`** (*self*, *objective*)

Determine whether the threshold of a binary classification pipeline can be tuned.

#### Parameters

- **`pipeline`** (*PipelineBase*) – Binary classification pipeline.
- **`objective`** – Primary AutoMLSearch objective.

**`property classes_`** (*self*)

Gets the class names for the problem.

**`clone`** (*self*)

Constructs a new pipeline with the same components, parameters, and random state.

**Returns** A new instance of this pipeline with identical components, parameters, and random state.

**`compute_estimator_features`** (*self*, *X*, *y=None*)

Transforms the data by applying all pre-processing components.

**Parameters** ***X*** (*pd.DataFrame*) – Input data to the pipeline to transform.

**Returns** New transformed features.

**Return type** *pd.DataFrame*

**`static create_objectives`** (*objectives*)

**`property custom_name`** (*self*)

Custom name of the pipeline.

**describe** (*self*, *return\_dict=False*)

Outputs pipeline details including component parameters

**Parameters** **return\_dict** (*bool*) – If True, return dictionary of information about pipeline. Defaults to False.

**Returns** Dictionary of all component parameters if *return\_dict* is True, else None

**Return type** dict

**property feature\_importance** (*self*)

Importance associated with each feature. Features dropped by the feature selection are excluded.

**Returns** pd.DataFrame including feature names and their corresponding importance

**fit** (*self*, *X*, *y*)

Fit a time series classification pipeline.

**Parameters**

- **X** (*pd.DataFrame* or *np.ndarray*) – The input training data of shape [n\_samples, n\_features]
- **y** (*pd.Series*, *np.ndarray*) – The target training targets of length [n\_samples]

**Returns** self

**get\_component** (*self*, *name*)

Returns component by name

**Parameters** **name** (*str*) – Name of component

**Returns** Component to return

**Return type** Component

**get\_hyperparameter\_ranges** (*self*, *custom\_hyperparameters*)

Returns hyperparameter ranges from all components as a dictionary.

**Parameters** **custom\_hyperparameters** (*dict*) – Custom hyperparameters for the pipeline.

**Returns** Dictionary of hyperparameter ranges for each component in the pipeline.

**Return type** dict

**graph** (*self*, *filepath=None*)

Generate an image representing the pipeline graph.

**Parameters** **filepath** (*str*, *optional*) – Path to where the graph should be saved. If set to None (as by default), the graph will not be saved.

**Returns** Graph object that can be directly displayed in Jupyter notebooks.

**Return type** graphviz.Digraph

**graph\_feature\_importance** (*self*, *importance\_threshold=0*)

Generate a bar graph of the pipeline's feature importance

**Parameters** **importance\_threshold** (*float*, *optional*) – If provided, graph features with a permutation importance whose absolute value is larger than *importance\_threshold*. Defaults to zero.

**Returns** plotly.Figure, a bar graph showing features and their corresponding importance

**inverse\_transform** (*self*, *y*)

Apply component inverse\_transform methods to estimator predictions in reverse order.

Components that implement inverse\_transform are PolynomialDetrender, LabelEncoder (tbd).

**Parameters** *y* (*pd.Series*) – Final component features

**static load** (*file\_path*)

Loads pipeline at file path

**Parameters** *file\_path* (*str*) – location to load file

**Returns** PipelineBase object

**property model\_family** (*self*)

Returns model family of this pipeline.

**property name** (*self*)

Name of the pipeline.

**new** (*self*, *parameters*, *random\_seed=0*)

**Constructs a new instance of the pipeline with the same component graph but with a different set of parameters.**

Not to be confused with python's `__new__` method.

**Parameters**

- **parameters** (*dict*) – Dictionary with component names as keys and dictionary of that component's parameters as values. An empty dictionary or None implies using all default values for component parameters. Defaults to None.
- **random\_seed** (*int*) – Seed for the random number generator. Defaults to 0.

**Returns** A new instance of this pipeline with identical components.

**optimize\_threshold** (*self*, *X*, *y*, *y\_pred\_proba*, *objective*)

Optimize the pipeline threshold given the objective to use. Only used for binary problems with objectives whose thresholds can be tuned.

**Parameters**

- **X** (*pd.DataFrame*) – Input features
- **y** (*pd.Series*) – Input target values
- **y\_pred\_proba** (*pd.Series*) – The predicted probabilities of the target outputted by the pipeline
- **objective** (*ObjectiveBase*) – The objective to threshold with. Must have a tunable threshold.

**property parameters** (*self*)

Parameter dictionary for this pipeline.

**Returns** Dictionary of all component parameters.

**Return type** dict

**predict** (*self*, *X*, *y=None*, *objective=None*)

Make predictions using selected features.

**Parameters**

- **X** (*pd.DataFrame*, or *np.ndarray*) – Data of shape [n\_samples, n\_features]

- **y** (*pd.Series, np.ndarray, None*) – The target training targets of length [n\_samples]
- **objective** (*Object or string*) – The objective to use to make predictions

**Returns** Predicted values.

**Return type** *pd.Series*

**predict\_proba** (*self, X, y=None*)

Make probability estimates for labels.

**Parameters** **X** (*pd.DataFrame or np.ndarray*) – Data of shape [n\_samples, n\_features]

**Returns** Probability estimates

**Return type** *pd.DataFrame*

**save** (*self, file\_path, pickle\_protocol=cloudpickle.DEFAULT\_PROTOCOL*)

Saves pipeline at file path

**Parameters**

- **file\_path** (*str*) – location to save file
- **pickle\_protocol** (*int*) – the pickle data stream format.

**Returns** *None*

**score** (*self, X, y, objectives*)

Evaluate model performance on current and additional objectives.

**Parameters**

- **X** (*pd.DataFrame or np.ndarray*) – Data of shape [n\_samples, n\_features]
- **y** (*pd.Series*) – True labels of length [n\_samples]
- **objectives** (*list*) – Non-empty list of objectives to score on

**Returns** Ordered dictionary of objective scores

**Return type** *dict*

**property summary** (*self*)

A short summary of the pipeline structure, describing the list of components used. Example: Logistic Regression Classifier w/ Simple Imputer + One Hot Encoder

**property threshold** (*self*)

Threshold used to make a prediction. Defaults to *None*.

**class** `evalml.pipelines.time_series_classification_pipelines.TimeSeriesClassificationPipeline`

Pipeline base class for time series classification problems.

**Parameters**

- **component\_graph** (*list or dict*) – List of components in order. Accepts strings or ComponentBase subclasses in the list. Note that when duplicate components are specified in a list, the duplicate component names will be modified with the component’s index in the list. For example, the component graph [Imputer, One Hot Encoder, Imputer, Logistic Regression Classifier] will have names [“Imputer”, “One Hot Encoder”, “Imputer\_2”, “Logistic Regression Classifier”]
- **parameters** (*dict*) – Dictionary with component names as keys and dictionary of that component’s parameters as values. An empty dictionary {} implies using all default values for component parameters. Pipeline-level parameters such as date\_index, gap, and max\_delay must be specified with the “pipeline” key. For example: Pipeline(parameters={“pipeline”: {“date\_index”: “Date”, “max\_delay”: 4, “gap”: 2}}).
- **random\_seed** (*int*) – Seed for the random number generator. Defaults to 0.

#### Attributes

<b>problem_type</b>	None
---------------------	------

#### Methods

<i>can_tune_threshold_with_objective</i>	Determine whether the threshold of a binary classification pipeline can be tuned.
<i>classes_</i>	Gets the class names for the problem.
<i>clone</i>	Constructs a new pipeline with the same components, parameters, and random state.
<i>compute_estimator_features</i>	Transforms the data by applying all pre-processing components.
<i>create_objectives</i>	
<i>custom_name</i>	Custom name of the pipeline.
<i>describe</i>	Outputs pipeline details including component parameters
<i>feature_importance</i>	Importance associated with each feature. Features dropped by the feature selection are excluded.
<i>fit</i>	Fit a time series classification pipeline.
<i>get_component</i>	Returns component by name
<i>get_hyperparameter_ranges</i>	Returns hyperparameter ranges from all components as a dictionary.
<i>graph</i>	Generate an image representing the pipeline graph.
<i>graph_feature_importance</i>	Generate a bar graph of the pipeline’s feature importance
<i>inverse_transform</i>	Apply component inverse_transform methods to estimator predictions in reverse order.
<i>load</i>	Loads pipeline at file path
<i>model_family</i>	Returns model family of this pipeline.
<i>name</i>	Name of the pipeline.
<i>new</i>	Constructs a new instance of the pipeline with the same component graph but with a different set of parameters.
<i>parameters</i>	Parameter dictionary for this pipeline.
<i>predict</i>	Make predictions using selected features.

continues on next page

Table 623 – continued from previous page

<code>predict_proba</code>	Make probability estimates for labels.
<code>save</code>	Saves pipeline at file path
<code>score</code>	Evaluate model performance on current and additional objectives.
<code>summary</code>	A short summary of the pipeline structure, describing the list of components used.

**can\_tune\_threshold\_with\_objective** (*self*, *objective*)

Determine whether the threshold of a binary classification pipeline can be tuned.

**Parameters**

- **pipeline** (*PipelineBase*) – Binary classification pipeline.
- **objective** – Primary AutoMLSearch objective.

**property classes\_** (*self*)

Gets the class names for the problem.

**clone** (*self*)

Constructs a new pipeline with the same components, parameters, and random state.

**Returns** A new instance of this pipeline with identical components, parameters, and random state.

**compute\_estimator\_features** (*self*, *X*, *y=None*)

Transforms the data by applying all pre-processing components.

**Parameters** **X** (*pd.DataFrame*) – Input data to the pipeline to transform.

**Returns** New transformed features.

**Return type** *pd.DataFrame*

**static create\_objectives** (*objectives*)

**property custom\_name** (*self*)

Custom name of the pipeline.

**describe** (*self*, *return\_dict=False*)

Outputs pipeline details including component parameters

**Parameters** **return\_dict** (*bool*) – If True, return dictionary of information about pipeline. Defaults to False.

**Returns** Dictionary of all component parameters if *return\_dict* is True, else None

**Return type** *dict*

**property feature\_importance** (*self*)

Importance associated with each feature. Features dropped by the feature selection are excluded.

**Returns** *pd.DataFrame* including feature names and their corresponding importance

**fit** (*self*, *X*, *y*)

Fit a time series classification pipeline.

**Parameters**

- **X** (*pd.DataFrame* or *np.ndarray*) – The input training data of shape [n\_samples, n\_features]
- **y** (*pd.Series*, *np.ndarray*) – The target training targets of length [n\_samples]

**Returns** self

**get\_component** (*self*, *name*)

Returns component by name

**Parameters** **name** (*str*) – Name of component

**Returns** Component to return

**Return type** Component

**get\_hyperparameter\_ranges** (*self*, *custom\_hyperparameters*)

Returns hyperparameter ranges from all components as a dictionary.

**Parameters** **custom\_hyperparameters** (*dict*) – Custom hyperparameters for the pipeline.

**Returns** Dictionary of hyperparameter ranges for each component in the pipeline.

**Return type** dict

**graph** (*self*, *filepath=None*)

Generate an image representing the pipeline graph.

**Parameters** **filepath** (*str*, *optional*) – Path to where the graph should be saved. If set to None (as by default), the graph will not be saved.

**Returns** Graph object that can be directly displayed in Jupyter notebooks.

**Return type** graphviz.Digraph

**graph\_feature\_importance** (*self*, *importance\_threshold=0*)

Generate a bar graph of the pipeline's feature importance

**Parameters** **importance\_threshold** (*float*, *optional*) – If provided, graph features with a permutation importance whose absolute value is larger than importance\_threshold. Defaults to zero.

**Returns** plotly.Figure, a bar graph showing features and their corresponding importance

**inverse\_transform** (*self*, *y*)

Apply component inverse\_transform methods to estimator predictions in reverse order.

Components that implement inverse\_transform are PolynomialDetrender, LabelEncoder (tbd).

**Parameters** **y** (*pd.Series*) – Final component features

**static load** (*file\_path*)

Loads pipeline at file path

**Parameters** **file\_path** (*str*) – location to load file

**Returns** PipelineBase object

**property model\_family** (*self*)

Returns model family of this pipeline.

**property name** (*self*)

Name of the pipeline.

**new** (*self*, *parameters*, *random\_seed=0*)

**Constructs a new instance of the pipeline with the same component graph but with a different set of parameters.**

Not to be confused with python's \_\_new\_\_ method.

**Parameters**



- **parameters** (*dict*) – Dictionary with component names as keys and dictionary of that component's parameters as values. An empty dictionary or None implies using all default values for component parameters. Defaults to None.
- **random\_seed** (*int*) – Seed for the random number generator. Defaults to 0.

**Returns** A new instance of this pipeline with identical components.

**property parameters** (*self*)

Parameter dictionary for this pipeline.

**Returns** Dictionary of all component parameters.

**Return type** dict

**predict** (*self*, *X*, *y=None*, *objective=None*)

Make predictions using selected features.

**Parameters**

- **X** (*pd.DataFrame*, or *np.ndarray*) – Data of shape [n\_samples, n\_features]
- **y** (*pd.Series*, *np.ndarray*, *None*) – The target training targets of length [n\_samples]
- **objective** (*Object* or *string*) – The objective to use to make predictions

**Returns** Predicted values.

**Return type** pd.Series

**predict\_proba** (*self*, *X*, *y=None*)

Make probability estimates for labels.

**Parameters X** (*pd.DataFrame* or *np.ndarray*) – Data of shape [n\_samples, n\_features]

**Returns** Probability estimates

**Return type** pd.DataFrame

**save** (*self*, *file\_path*, *pickle\_protocol=cloudpickle.DEFAULT\_PROTOCOL*)

Saves pipeline at file path

**Parameters**

- **file\_path** (*str*) – location to save file
- **pickle\_protocol** (*int*) – the pickle data stream format.

**Returns** None

**score** (*self*, *X*, *y*, *objectives*)

Evaluate model performance on current and additional objectives.

**Parameters**

- **X** (*pd.DataFrame* or *np.ndarray*) – Data of shape [n\_samples, n\_features]
- **y** (*pd.Series*) – True labels of length [n\_samples]
- **objectives** (*list*) – Non-empty list of objectives to score on

**Returns** Ordered dictionary of objective scores

**Return type** dict

**property summary** (*self*)

A short summary of the pipeline structure, describing the list of components used. Example: Logistic Regression Classifier w/ Simple Imputer + One Hot Encoder

**class** evalml.pipelines.time\_series\_classification\_pipelines.**TimeSeriesMulticlassClassifier**

Pipeline base class for time series multiclass classification problems.

**Parameters**

- **component\_graph** (*list or dict*) – List of components in order. Accepts strings or ComponentBase subclasses in the list. Note that when duplicate components are specified in a list, the duplicate component names will be modified with the component’s index in the list. For example, the component graph [Imputer, One Hot Encoder, Imputer, Logistic Regression Classifier] will have names [“Imputer”, “One Hot Encoder”, “Imputer\_2”, “Logistic Regression Classifier”]
- **parameters** (*dict*) – Dictionary with component names as keys and dictionary of that component’s parameters as values. An empty dictionary {} implies using all default values for component parameters. Pipeline-level parameters such as date\_index, gap, and max\_delay must be specified with the “pipeline” key. For example: Pipeline(parameters={“pipeline”: {“date\_index”: “Date”, “max\_delay”: 4, “gap”: 2}}).
- **random\_seed** (*int*) – Seed for the random number generator. Defaults to 0.

**Attributes**

<b>problem_type</b>	ProblemTypes.TIME_SERIES_MULTICLASS
---------------------	-------------------------------------

**Methods**

<i>can_tune_threshold_with_objective</i>	Determine whether the threshold of a binary classification pipeline can be tuned.
<i>classes_</i>	Gets the class names for the problem.
<i>clone</i>	Constructs a new pipeline with the same components, parameters, and random state.
<i>compute_estimator_features</i>	Transforms the data by applying all pre-processing components.
<i>create_objectives</i>	
<i>custom_name</i>	Custom name of the pipeline.
<i>describe</i>	Outputs pipeline details including component parameters
<i>feature_importance</i>	Importance associated with each feature. Features dropped by the feature selection are excluded.
<i>fit</i>	Fit a time series classification pipeline.
<i>get_component</i>	Returns component by name

continues on next page

Table 624 – continued from previous page

<code>get_hyperparameter_ranges</code>	Returns hyperparameter ranges from all components as a dictionary.
<code>graph</code>	Generate an image representing the pipeline graph.
<code>graph_feature_importance</code>	Generate a bar graph of the pipeline’s feature importance
<code>inverse_transform</code>	Apply component <code>inverse_transform</code> methods to estimator predictions in reverse order.
<code>load</code>	Loads pipeline at file path
<code>model_family</code>	Returns model family of this pipeline.
<code>name</code>	Name of the pipeline.
<code>new</code>	Constructs a new instance of the pipeline with the same component graph but with a different set of parameters.
<code>parameters</code>	Parameter dictionary for this pipeline.
<code>predict</code>	Make predictions using selected features.
<code>predict_proba</code>	Make probability estimates for labels.
<code>save</code>	Saves pipeline at file path
<code>score</code>	Evaluate model performance on current and additional objectives.
<code>summary</code>	A short summary of the pipeline structure, describing the list of components used.

**`can_tune_threshold_with_objective`** (*self*, *objective*)

Determine whether the threshold of a binary classification pipeline can be tuned.

**Parameters**

- **`pipeline`** (*PipelineBase*) – Binary classification pipeline.
- **`objective`** – Primary AutoMLSearch objective.

**`property classes_`** (*self*)

Gets the class names for the problem.

**`clone`** (*self*)

Constructs a new pipeline with the same components, parameters, and random state.

**Returns** A new instance of this pipeline with identical components, parameters, and random state.

**`compute_estimator_features`** (*self*, *X*, *y=None*)

Transforms the data by applying all pre-processing components.

**Parameters** **`X`** (*pd.DataFrame*) – Input data to the pipeline to transform.

**Returns** New transformed features.

**Return type** *pd.DataFrame*

**`static create_objectives`** (*objectives*)

**`property custom_name`** (*self*)

Custom name of the pipeline.

**`describe`** (*self*, *return\_dict=False*)

Outputs pipeline details including component parameters

**Parameters** **`return_dict`** (*bool*) – If True, return dictionary of information about pipeline. Defaults to False.

**Returns** Dictionary of all component parameters if `return_dict` is `True`, else `None`

**Return type** dict

**property** `feature_importance` (*self*)

Importance associated with each feature. Features dropped by the feature selection are excluded.

**Returns** `pd.DataFrame` including feature names and their corresponding importance

**fit** (*self*, *X*, *y*)

Fit a time series classification pipeline.

**Parameters**

- **X** (`pd.DataFrame` or `np.ndarray`) – The input training data of shape `[n_samples, n_features]`
- **y** (`pd.Series`, `np.ndarray`) – The target training targets of length `[n_samples]`

**Returns** *self*

**get\_component** (*self*, *name*)

Returns component by name

**Parameters** **name** (*str*) – Name of component

**Returns** Component to return

**Return type** Component

**get\_hyperparameter\_ranges** (*self*, *custom\_hyperparameters*)

Returns hyperparameter ranges from all components as a dictionary.

**Parameters** **custom\_hyperparameters** (*dict*) – Custom hyperparameters for the pipeline.

**Returns** Dictionary of hyperparameter ranges for each component in the pipeline.

**Return type** dict

**graph** (*self*, *filepath=None*)

Generate an image representing the pipeline graph.

**Parameters** **filepath** (*str*, *optional*) – Path to where the graph should be saved. If set to `None` (as by default), the graph will not be saved.

**Returns** Graph object that can be directly displayed in Jupyter notebooks.

**Return type** `graphviz.Digraph`

**graph\_feature\_importance** (*self*, *importance\_threshold=0*)

Generate a bar graph of the pipeline's feature importance

**Parameters** **importance\_threshold** (*float*, *optional*) – If provided, graph features with a permutation importance whose absolute value is larger than `importance_threshold`. Defaults to zero.

**Returns** `plotly.Figure`, a bar graph showing features and their corresponding importance

**inverse\_transform** (*self*, *y*)

Apply component `inverse_transform` methods to estimator predictions in reverse order.

Components that implement `inverse_transform` are `PolynomialDetrender`, `LabelEncoder` (tbd).

**Parameters** **y** (`pd.Series`) – Final component features

**static load** (*file\_path*)

Loads pipeline at file path

**Parameters** *file\_path* (*str*) – location to load file

**Returns** PipelineBase object

**property model\_family** (*self*)

Returns model family of this pipeline.

**property name** (*self*)

Name of the pipeline.

**new** (*self, parameters, random\_seed=0*)

**Constructs a new instance of the pipeline with the same component graph but with a different set of parameters.**

Not to be confused with python's `__new__` method.

**Parameters**

- **parameters** (*dict*) – Dictionary with component names as keys and dictionary of that component's parameters as values. An empty dictionary or None implies using all default values for component parameters. Defaults to None.
- **random\_seed** (*int*) – Seed for the random number generator. Defaults to 0.

**Returns** A new instance of this pipeline with identical components.

**property parameters** (*self*)

Parameter dictionary for this pipeline.

**Returns** Dictionary of all component parameters.

**Return type** dict

**predict** (*self, X, y=None, objective=None*)

Make predictions using selected features.

**Parameters**

- **X** (*pd.DataFrame, or np.ndarray*) – Data of shape [n\_samples, n\_features]
- **y** (*pd.Series, np.ndarray, None*) – The target training targets of length [n\_samples]
- **objective** (*Object or string*) – The objective to use to make predictions

**Returns** Predicted values.

**Return type** pd.Series

**predict\_proba** (*self, X, y=None*)

Make probability estimates for labels.

**Parameters** **X** (*pd.DataFrame or np.ndarray*) – Data of shape [n\_samples, n\_features]

**Returns** Probability estimates

**Return type** pd.DataFrame

**save** (*self, file\_path, pickle\_protocol=cloudpickle.DEFAULT\_PROTOCOL*)

Saves pipeline at file path

**Parameters**

- **file\_path** (*str*) – location to save file
- **pickle\_protocol** (*int*) – the pickle data stream format.

**Returns** None**score** (*self, X, y, objectives*)

Evaluate model performance on current and additional objectives.

## Parameters

- **X** (*pd.DataFrame* or *np.ndarray*) – Data of shape [n\_samples, n\_features]
- **y** (*pd.Series*) – True labels of length [n\_samples]
- **objectives** (*list*) – Non-empty list of objectives to score on

**Returns** Ordered dictionary of objective scores

**Return type** dict

property summary (*self*)

A short summary of the pipeline structure, describing the list of components used. Example: Logistic Regression Classifier w/ Simple Imputer + One Hot Encoder

## time\_series\_regression\_pipeline

## Module Contents

## Classes Summary

<i>TimeSeriesRegressionPipeline</i>	Pipeline base class for time series regression problems.
-------------------------------------	--

## Contents

```
class evalml.pipelines.time_series_regression_pipeline.TimeSeriesRegressionPipeline(components)
```

pa-  
ram-  
e-  
ters=Non  
cus-  
tom\_name  
ran-  
dom\_see

Pipeline base class for time series regression problems.

## Parameters

- **component\_graph** (*list or dict*) – List of components in order. Accepts strings or ComponentBase subclasses in the list. Note that when duplicate components are specified in a list, the duplicate component names will be modified with the component’s index in the list. For example, the component graph [Imputer, One Hot Encoder, Imputer, Logistic Regression Classifier] will have names [“Imputer”, “One Hot Encoder”, “Imputer\_2”, “Logistic Regression Classifier”]
- **parameters** (*dict*) – Dictionary with component names as keys and dictionary of that component’s parameters as values. An empty dictionary {} implies us-

ing all default values for component parameters. Pipeline-level parameters such as `date_index`, `gap`, and `max_delay` must be specified with the “pipeline” key. For example: `Pipeline(parameters={"pipeline": {"date_index": "Date", "max_delay": 4, "gap": 2}})`.

- **`random_seed`** (*int*) – Seed for the random number generator. Defaults to 0.

#### Attributes

<b>problem_type</b>	ProblemTypes.TIME_SERIES_REGRESSIO
---------------------	------------------------------------

#### Methods

<i>can_tune_threshold_with_objective</i>	Determine whether the threshold of a binary classification pipeline can be tuned.
<i>clone</i>	Constructs a new pipeline with the same components, parameters, and random state.
<i>compute_estimator_features</i>	Transforms the data by applying all pre-processing components.
<i>create_objectives</i>	
<i>custom_name</i>	Custom name of the pipeline.
<i>describe</i>	Outputs pipeline details including component parameters
<i>feature_importance</i>	Importance associated with each feature. Features dropped by the feature selection are excluded.
<i>fit</i>	Fit a time series regression pipeline.
<i>get_component</i>	Returns component by name
<i>get_hyperparameter_ranges</i>	Returns hyperparameter ranges from all components as a dictionary.
<i>graph</i>	Generate an image representing the pipeline graph.
<i>graph_feature_importance</i>	Generate a bar graph of the pipeline’s feature importance
<i>inverse_transform</i>	Apply component <code>inverse_transform</code> methods to estimator predictions in reverse order.
<i>load</i>	Loads pipeline at file path
<i>model_family</i>	Returns model family of this pipeline.
<i>name</i>	Name of the pipeline.
<i>new</i>	Constructs a new instance of the pipeline with the same component graph but with a different set of parameters.
<i>parameters</i>	Parameter dictionary for this pipeline.
<i>predict</i>	Make predictions using selected features.
<i>save</i>	Saves pipeline at file path
<i>score</i>	Evaluate model performance on current and additional objectives.
<i>summary</i>	A short summary of the pipeline structure, describing the list of components used.

**`can_tune_threshold_with_objective`** (*self*, *objective*)

Determine whether the threshold of a binary classification pipeline can be tuned.

#### Parameters

- **pipeline** (*PipelineBase*) – Binary classification pipeline.
- **objective** – Primary AutoMLSearch objective.

**clone** (*self*)

Constructs a new pipeline with the same components, parameters, and random state.

**Returns** A new instance of this pipeline with identical components, parameters, and random state.

**compute\_estimator\_features** (*self, X, y=None*)

Transforms the data by applying all pre-processing components.

**Parameters** **X** (*pd.DataFrame*) – Input data to the pipeline to transform.

**Returns** New transformed features.

**Return type** *pd.DataFrame*

**static create\_objectives** (*objectives*)

**property custom\_name** (*self*)

Custom name of the pipeline.

**describe** (*self, return\_dict=False*)

Outputs pipeline details including component parameters

**Parameters** **return\_dict** (*bool*) – If True, return dictionary of information about pipeline. Defaults to False.

**Returns** Dictionary of all component parameters if *return\_dict* is True, else None

**Return type** *dict*

**property feature\_importance** (*self*)

Importance associated with each feature. Features dropped by the feature selection are excluded.

**Returns** *pd.DataFrame* including feature names and their corresponding importance

**fit** (*self, X, y*)

Fit a time series regression pipeline.

**Parameters**

- **X** (*pd.DataFrame* or *np.ndarray*) – The input training data of shape [n\_samples, n\_features]
- **y** (*pd.Series*, *np.ndarray*) – The target training targets of length [n\_samples]

**Returns** *self*

**get\_component** (*self, name*)

Returns component by name

**Parameters** **name** (*str*) – Name of component

**Returns** Component to return

**Return type** Component

**get\_hyperparameter\_ranges** (*self, custom\_hyperparameters*)

Returns hyperparameter ranges from all components as a dictionary.

**Parameters** **custom\_hyperparameters** (*dict*) – Custom hyperparameters for the pipeline.

**Returns** Dictionary of hyperparameter ranges for each component in the pipeline.



**Return type** dict

**graph** (*self*, *filepath=None*)

Generate an image representing the pipeline graph.

**Parameters** **filepath** (*str*, *optional*) – Path to where the graph should be saved. If set to None (as by default), the graph will not be saved.

**Returns** Graph object that can be directly displayed in Jupyter notebooks.

**Return type** graphviz.Digraph

**graph\_feature\_importance** (*self*, *importance\_threshold=0*)

Generate a bar graph of the pipeline's feature importance

**Parameters** **importance\_threshold** (*float*, *optional*) – If provided, graph features with a permutation importance whose absolute value is larger than importance\_threshold. Defaults to zero.

**Returns** plotly.Figure, a bar graph showing features and their corresponding importance

**inverse\_transform** (*self*, *y*)

Apply component inverse\_transform methods to estimator predictions in reverse order.

Components that implement inverse\_transform are PolynomialDetrender, LabelEncoder (tbd).

**Parameters** **y** (*pd.Series*) – Final component features

**static load** (*file\_path*)

Loads pipeline at file path

**Parameters** **file\_path** (*str*) – location to load file

**Returns** PipelineBase object

**property model\_family** (*self*)

Returns model family of this pipeline.

**property name** (*self*)

Name of the pipeline.

**new** (*self*, *parameters*, *random\_seed=0*)

**Constructs a new instance of the pipeline with the same component graph but with a different set of parameters.**

Not to be confused with python's \_\_new\_\_ method.

**Parameters**

- **parameters** (*dict*) – Dictionary with component names as keys and dictionary of that component's parameters as values. An empty dictionary or None implies using all default values for component parameters. Defaults to None.
- **random\_seed** (*int*) – Seed for the random number generator. Defaults to 0.

**Returns** A new instance of this pipeline with identical components.

**property parameters** (*self*)

Parameter dictionary for this pipeline.

**Returns** Dictionary of all component parameters.

**Return type** dict

**predict** (*self*, *X*, *y=None*, *objective=None*)

Make predictions using selected features.

**Parameters**

- **X** (*pd.DataFrame, or np.ndarray*) – Data of shape [n\_samples, n\_features]
- **y** (*pd.Series, np.ndarray, None*) – The target training targets of length [n\_samples]
- **objective** (*Object or string*) – The objective to use to make predictions

**Returns** Predicted values.

**Return type** *pd.Series*

**save** (*self, file\_path, pickle\_protocol=cloudpickle.DEFAULT\_PROTOCOL*)  
Saves pipeline at file path

**Parameters**

- **file\_path** (*str*) – location to save file
- **pickle\_protocol** (*int*) – the pickle data stream format.

**Returns** None

**score** (*self, X, y, objectives*)  
Evaluate model performance on current and additional objectives.

**Parameters**

- **X** (*pd.DataFrame or np.ndarray*) – Data of shape [n\_samples, n\_features]
- **y** (*pd.Series*) – True labels of length [n\_samples]
- **objectives** (*list*) – Non-empty list of objectives to score on

**Returns** Ordered dictionary of objective scores

**Return type** *dict*

**property summary** (*self*)  
A short summary of the pipeline structure, describing the list of components used. Example: Logistic Regression Classifier w/ Simple Imputer + One Hot Encoder

**utils****Module Contents****Functions**

---

<i>generate_pipeline_code</i>	Creates and returns a string that contains the Python imports and code required for running the EvalML pipeline.
<i>make_pipeline</i>	Given input data, target data, an estimator class and the problem type,

---

## Attributes Summary

---

*logger*

---

## Contents

`evalml.pipelines.utils.generate_pipeline_code(element)`

Creates and returns a string that contains the Python imports and code required for running the EvalML pipeline.

**Parameters** `element` (*pipeline instance*) – The instance of the pipeline to generate string Python code

**Returns** String representation of Python code that can be run separately in order to recreate the pipeline instance. Does not include code for custom component implementation.

`evalml.pipelines.utils.logger`

`evalml.pipelines.utils.make_pipeline(X, y, estimator, problem_type, parameters=None, sampler_name=None, extra_components=None)`

Given input data, target data, an estimator class and the problem type, generates a pipeline class with a preprocessing chain which was recommended based on the inputs. The pipeline will be a subclass of the appropriate pipeline base class for the specified problem\_type.

### Parameters

- **X** (*pd.DataFrame*) – The input data of shape [n\_samples, n\_features]
- **y** (*pd.Series*) – The target data of length [n\_samples]
- **estimator** (*Estimator*) – Estimator for pipeline
- **problem\_type** (*ProblemTypes* or *str*) – Problem type for pipeline to generate
- **parameters** (*dict*) – Dictionary with component names as keys and dictionary of that component's parameters as values. An empty dictionary or None implies using all default values for component parameters.
- **sampler\_name** (*str*) – The name of the sampler component to add to the pipeline. Only used in classification problems. Defaults to None
- **extra\_components** – List of extra components to be added after preprocessing components. Defaults to None.

## Package Contents

## Classes Summary

<i>ARIMAREgressor</i>	Autoregressive Integrated Moving Average Model.
<i>CatBoostClassifier</i>	CatBoost Classifier, a classifier that uses gradient-boosting on decision trees.
<i>CatBoostRegressor</i>	CatBoost Regressor, a regressor that uses gradient-boosting on decision trees.

continues on next page

Table 629 – continued from previous page

<i>ComponentGraph</i>	Component graph for a pipeline as a directed acyclic graph (DAG).
<i>DecisionTreeClassifier</i>	Decision Tree Classifier.
<i>DecisionTreeRegressor</i>	Decision Tree Regressor.
<i>DelayedFeatureTransformer</i>	Transformer that delays input features and target variable for time series problems.
<i>DFSTransformer</i>	Featuretools DFS component that generates features for the input features.
<i>ElasticNetClassifier</i>	Elastic Net Classifier. Uses Logistic Regression with elasticnet penalty as the base estimator.
<i>ElasticNetRegressor</i>	Elastic Net Regressor.
<i>Estimator</i>	A component that fits and predicts given data.
<i>ExtraTreesClassifier</i>	Extra Trees Classifier.
<i>ExtraTreesRegressor</i>	Extra Trees Regressor.
<i>FeatureSelector</i>	Selects top features based on importance weights.
<i>KNeighborsClassifier</i>	K-Nearest Neighbors Classifier.
<i>LightGBMClassifier</i>	LightGBM Classifier.
<i>LightGBMRegressor</i>	LightGBM Regressor.
<i>LinearRegressor</i>	Linear Regressor.
<i>LogisticRegressionClassifier</i>	Logistic Regression Classifier.
<i>MulticlassClassificationPipeline</i>	Pipeline subclass for all multiclass classification pipelines.
<i>OneHotEncoder</i>	A transformer that encodes categorical features in a one-hot numeric array.
<i>PerColumnImputer</i>	Imputes missing data according to a specified imputation strategy per column.
<i>PipelineBase</i>	Machine learning pipeline made out of transformers and a estimator.
<i>ProphetRegressor</i>	Prophet is a procedure for forecasting time series data based on an additive model where non-linear trends are fit with yearly, weekly, and daily seasonality, plus holiday effects.
<i>RandomForestClassifier</i>	Random Forest Classifier.
<i>RandomForestRegressor</i>	Random Forest Regressor.
<i>RegressionPipeline</i>	Pipeline subclass for all regression pipelines.
<i>RFClassifierSelectFromModel</i>	Selects top features based on importance weights using a Random Forest classifier.
<i>RFRegressorSelectFromModel</i>	Selects top features based on importance weights using a Random Forest regressor.
<i>SimpleImputer</i>	Imputes missing data according to a specified imputation strategy.
<i>SklearnStackedEnsembleClassifier</i>	Scikit-learn Stacked Ensemble Classifier.
<i>SklearnStackedEnsembleRegressor</i>	Scikit-learn Stacked Ensemble Regressor.
<i>StandardScaler</i>	A transformer that standardizes input features by removing the mean and scaling to unit variance.
<i>SVMClassifier</i>	Support Vector Machine Classifier.
<i>SVMRegressor</i>	Support Vector Machine Regressor.
<i>TargetEncoder</i>	A transformer that encodes categorical features into target encodings.
<i>TimeSeriesBinaryClassificationPipeline</i>	Pipeline base class for time series binary classification problems.

continues on next page

Table 629 – continued from previous page

<i>TimeSeriesClassificationPipeline</i>	Pipeline base class for time series classification problems.
<i>TimeSeriesMulticlassClassificationPipeline</i>	Pipeline base class for time series multiclass classification problems.
<i>TimeSeriesRegressionPipeline</i>	Pipeline base class for time series regression problems.
<i>Transformer</i>	A component that may or may not need fitting that transforms data.
<i>XGBoostClassifier</i>	XGBoost Classifier.
<i>XGBoostRegressor</i>	XGBoost Regressor.

## Contents

**class** evalml.pipelines.**ARIMAREgressor**(*date\_index=None, trend=None, start\_p=2, d=0, start\_q=2, max\_p=5, max\_d=2, max\_q=5, seasonal=True, n\_jobs=-1, random\_seed=0, \*\*kwargs*)

Autoregressive Integrated Moving Average Model. The three parameters (p, d, q) are the AR order, the degree of differencing, and the MA order. More information here: [https://www.statsmodels.org/devel/generated/statsmodels.tsa.arima\\_model.ARIMA.html](https://www.statsmodels.org/devel/generated/statsmodels.tsa.arima_model.ARIMA.html)

Currently ARIMAREgressor isn't supported via conda install. It's recommended that it be installed via PyPI.

### Parameters

- **date\_index** (*str*) – Specifies the name of the column in X that provides the datetime objects. Defaults to None.
- **trend** (*str*) – Controls the deterministic trend. Options are ['n', 'c', 't', 'ct'] where 'c' is a constant term, 't' indicates a linear trend, and 'ct' is both. Can also be an iterable when defining a polynomial, such as [1, 1, 0, 1].
- **start\_p** (*int*) – Minimum Autoregressive order. Defaults to 2.
- **d** (*int*) – Minimum Differencing degree. Defaults to 0.
- **start\_q** (*int*) – Minimum Moving Average order. Defaults to 2.
- **max\_p** (*int*) – Maximum Autoregressive order. Defaults to 5.
- **max\_d** (*int*) – Maximum Differencing degree. Defaults to 2.
- **max\_q** (*int*) – Maximum Moving Average order. Defaults to 5.
- **seasonal** (*boolean*) – Whether to fit a seasonal model to ARIMA. Defaults to True.
- **n\_jobs** (*int or None*) – Non-negative integer describing level of parallelism used for pipelines. Defaults to -1.
- **random\_seed** (*int*) – Seed for the random number generator. Defaults to 0.

### Attributes

<b>hyper-parameter_ranges</b>	{ "start_p": Integer(1, 3), "d": Integer(0, 2), "start_q": Integer(1, 3), "max_p": Integer(3, 10), "max_d": Integer(2, 5), "max_q": Integer(3, 10), "seasonal": [True, False], }
<b>model_family</b>	ModelFamily.ARIMA
<b>modifies_features</b>	True
<b>modifies_target</b>	False
<b>name</b>	ARIMA Regressor
<b>pre-dict_uses_y</b>	False
<b>supported_problem_types</b>	[ProblemTypes.TIME_SERIES_REGRESSION]

## Methods

<i>clone</i>	Constructs a new component with the same parameters and random state.
<i>default_parameters</i>	Returns the default parameters for this component.
<i>describe</i>	Describe a component and its parameters
<i>feature_importance</i>	Returns array of 0's with a length of 1 as feature_importance is not defined for ARIMA regressor.
<i>fit</i>	Fits component to data
<i>load</i>	Loads component at file path
<i>needs_fitting</i>	Returns boolean determining if component needs fitting before
<i>parameters</i>	Returns the parameters which were used to initialize the component
<i>predict</i>	Make predictions using selected features.
<i>predict_proba</i>	Make probability estimates for labels.
<i>save</i>	Saves component at file path

### **clone** (*self*)

Constructs a new component with the same parameters and random state.

**Returns** A new instance of this component with identical parameters and random state.

### **default\_parameters** (*cls*)

Returns the default parameters for this component.

Our convention is that `Component.default_parameters == Component().parameters`.

**Returns** default parameters for this component.

**Return type** dict

### **describe** (*self*, *print\_name=False*, *return\_dict=False*)

Describe a component and its parameters

#### Parameters

- **print\_name** (*bool*, *optional*) – whether to print name of component
- **return\_dict** (*bool*, *optional*) – whether to return description as dictionary in the format {"name": name, "parameters": parameters}

**Returns** prints and returns dictionary

**Return type** None or dict

**property feature\_importance** (*self*)

Returns array of 0's with a length of 1 as feature\_importance is not defined for ARIMA regressor.

**fit** (*self*, *X*, *y=None*)

Fits component to data

**Parameters**

- **X** (*list*, *pd.DataFrame* or *np.ndarray*) – The input training data of shape [n\_samples, n\_features]
- **y** (*list*, *pd.Series*, *np.ndarray*, *optional*) – The target training data of length [n\_samples]

**Returns** *self*

**static load** (*file\_path*)

Loads component at file path

**Parameters** **file\_path** (*str*) – Location to load file

**Returns** ComponentBase object

**needs\_fitting** (*self*)

Returns boolean determining if component needs fitting before calling predict, predict\_proba, transform, or feature\_importances. This can be overridden to False for components that do not need to be fit or whose fit methods do nothing.

**property parameters** (*self*)

Returns the parameters which were used to initialize the component

**predict** (*self*, *X*, *y=None*)

Make predictions using selected features.

**Parameters** **X** (*pd.DataFrame*, *np.ndarray*) – Data of shape [n\_samples, n\_features]

**Returns** Predicted values

**Return type** *pd.Series*

**predict\_proba** (*self*, *X*)

Make probability estimates for labels.

**Parameters** **X** (*pd.DataFrame*, or *np.ndarray*) – Features

**Returns** Probability estimates

**Return type** *pd.Series*

**save** (*self*, *file\_path*, *pickle\_protocol=cloudpickle.DEFAULT\_PROTOCOL*)

Saves component at file path

**Parameters**

- **file\_path** (*str*) – Location to save file
- **pickle\_protocol** (*int*) – The pickle data stream format.

**Returns** None

**class evalml.pipelines.CatBoostClassifier** (*n\_estimators=10*, *eta=0.03*, *max\_depth=6*,  
*bootstrap\_type=None*, *silent=True*, *allow\_writing\_files=False*, *random\_seed=0*,  
*n\_jobs=-1*, *\*\*kwargs*)

CatBoost Classifier, a classifier that uses gradient-boosting on decision trees. CatBoost is an open-source library

and natively supports categorical features.

For more information, check out <https://catboost.ai/>

### Parameters

- **n\_estimators** (*float*) – The maximum number of trees to build. Defaults to 10.
- **eta** (*float*) – The learning rate. Defaults to 0.03.
- **max\_depth** (*int*) – The maximum tree depth for base learners. Defaults to 6.
- **bootstrap\_type** (*string*) – Defines the method for sampling the weights of objects. Available methods are ‘Bayesian’, ‘Bernoulli’, ‘MVS’. Defaults to None.
- **silent** (*boolean*) – Whether to use the “silent” logging mode. Defaults to True.
- **allow\_writing\_files** (*boolean*) – Whether to allow writing snapshot files while training. Defaults to False.
- **n\_jobs** (*int or None*) – Number of jobs to run in parallel. -1 uses all processes. Defaults to -1.
- **random\_seed** (*int*) – Seed for the random number generator. Defaults to 0.

### Attributes

<b>hyper-parameter_ranges</b>	{ “n_estimators”: Integer(4, 100), “eta”: Real(0.000001, 1), “max_depth”: Integer(4, 10), }
<b>model_family</b>	ModelFamily.CATBOOST
<b>modifies_features</b>	True
<b>modifies_target</b>	False
<b>name</b>	CatBoost Classifier
<b>predict_uses_y</b>	False
<b>supported_problem_types</b>	[ ProblemTypes.BINARY, ProblemTypes.MULTICLASS, ProblemTypes.TIME_SERIES_BINARY, ProblemTypes.TIME_SERIES_MULTICLASS, ]

### Methods

<i>clone</i>	Constructs a new component with the same parameters and random state.
<i>default_parameters</i>	Returns the default parameters for this component.
<i>describe</i>	Describe a component and its parameters
<i>feature_importance</i>	Returns importance associated with each feature.
<i>fit</i>	Fits component to data
<i>load</i>	Loads component at file path
<i>needs_fitting</i>	Returns boolean determining if component needs fitting before
<i>parameters</i>	Returns the parameters which were used to initialize the component
<i>predict</i>	Make predictions using selected features.
<i>predict_proba</i>	Make probability estimates for labels.
<i>save</i>	Saves component at file path



**clone** (*self*)

Constructs a new component with the same parameters and random state.

**Returns** A new instance of this component with identical parameters and random state.

**default\_parameters** (*cls*)

Returns the default parameters for this component.

Our convention is that `Component.default_parameters == Component().parameters`.

**Returns** default parameters for this component.

**Return type** dict

**describe** (*self*, *print\_name=False*, *return\_dict=False*)

Describe a component and its parameters

**Parameters**

- **print\_name** (*bool*, *optional*) – whether to print name of component
- **return\_dict** (*bool*, *optional*) – whether to return description as dictionary in the format {"name": name, "parameters": parameters}

**Returns** prints and returns dictionary

**Return type** None or dict

**property feature\_importance** (*self*)

Returns importance associated with each feature.

**Returns** Importance associated with each feature

**Return type** np.ndarray

**fit** (*self*, *X*, *y=None*)

Fits component to data

**Parameters**

- **X** (*list*, *pd.DataFrame* or *np.ndarray*) – The input training data of shape [n\_samples, n\_features]
- **y** (*list*, *pd.Series*, *np.ndarray*, *optional*) – The target training data of length [n\_samples]

**Returns** self

**static load** (*file\_path*)

Loads component at file path

**Parameters** **file\_path** (*str*) – Location to load file

**Returns** ComponentBase object

**needs\_fitting** (*self*)

Returns boolean determining if component needs fitting before calling predict, predict\_proba, transform, or feature\_importances. This can be overridden to False for components that do not need to be fit or whose fit methods do nothing.

**property parameters** (*self*)

Returns the parameters which were used to initialize the component

**predict** (*self*, *X*)

Make predictions using selected features.

**Parameters** **X** (*pd.DataFrame*, *np.ndarray*) – Data of shape [n\_samples, n\_features]

**Returns** Predicted values

**Return type** `pd.Series`

**predict\_proba** (*self*, *X*)

Make probability estimates for labels.

**Parameters** *X* (`pd.DataFrame`, or `np.ndarray`) – Features

**Returns** Probability estimates

**Return type** `pd.Series`

**save** (*self*, *file\_path*, *pickle\_protocol=cloudpickle.DEFAULT\_PROTOCOL*)

Saves component at file path

**Parameters**

- **file\_path** (*str*) – Location to save file
- **pickle\_protocol** (*int*) – The pickle data stream format.

**Returns** `None`

```
class evalml.pipelines.CatBoostRegressor (n_estimators=10, eta=0.03, max_depth=6,  
                                           bootstrap_type=None, silent=False, al-  
                                           low_writing_files=False, random_seed=0,  
                                           n_jobs=-1, **kwargs)
```

CatBoost Regressor, a regressor that uses gradient-boosting on decision trees. CatBoost is an open-source library and natively supports categorical features.

For more information, check out <https://catboost.ai/>

**Parameters**

- **n\_estimators** (*float*) – The maximum number of trees to build. Defaults to 10.
- **eta** (*float*) – The learning rate. Defaults to 0.03.
- **max\_depth** (*int*) – The maximum tree depth for base learners. Defaults to 6.
- **bootstrap\_type** (*string*) – Defines the method for sampling the weights of objects. Available methods are ‘Bayesian’, ‘Bernoulli’, ‘MVS’. Defaults to `None`.
- **silent** (*boolean*) – Whether to use the “silent” logging mode. Defaults to `True`.
- **allow\_writing\_files** (*boolean*) – Whether to allow writing snapshot files while training. Defaults to `False`.
- **n\_jobs** (*int* or *None*) – Number of jobs to run in parallel. -1 uses all processes. Defaults to -1.
- **random\_seed** (*int*) – Seed for the random number generator. Defaults to 0.

**Attributes**

<b>hyper-parameter_ranges</b>	{ “n_estimators”: Integer(4, 100), “eta”: Real(0.000001, 1), “max_depth”: Integer(4, 10), }
<b>model_family</b>	ModelFamily.CATBOOST
<b>modifies_features</b>	True
<b>modifies_target</b>	False
<b>name</b>	CatBoost Regressor
<b>pre-dict_uses_y</b>	False
<b>supported_problem_types</b>	[ ProblemTypes.REGRESSION, ProblemTypes.TIME_SERIES_REGRESSION, ]

## Methods

<i>clone</i>	Constructs a new component with the same parameters and random state.
<i>default_parameters</i>	Returns the default parameters for this component.
<i>describe</i>	Describe a component and its parameters
<i>feature_importance</i>	Returns importance associated with each feature.
<i>fit</i>	Fits component to data
<i>load</i>	Loads component at file path
<i>needs_fitting</i>	Returns boolean determining if component needs fitting before
<i>parameters</i>	Returns the parameters which were used to initialize the component
<i>predict</i>	Make predictions using selected features.
<i>predict_proba</i>	Make probability estimates for labels.
<i>save</i>	Saves component at file path

### **clone** (*self*)

Constructs a new component with the same parameters and random state.

**Returns** A new instance of this component with identical parameters and random state.

### **default\_parameters** (*cls*)

Returns the default parameters for this component.

Our convention is that `Component.default_parameters == Component().parameters`.

**Returns** default parameters for this component.

**Return type** dict

### **describe** (*self*, *print\_name=False*, *return\_dict=False*)

Describe a component and its parameters

#### Parameters

- **print\_name** (*bool*, *optional*) – whether to print name of component
- **return\_dict** (*bool*, *optional*) – whether to return description as dictionary in the format {“name”: name, “parameters”: parameters}

**Returns** prints and returns dictionary

**Return type** None or dict

**property feature\_importance** (*self*)

Returns importance associated with each feature.

**Returns** Importance associated with each feature

**Return type** np.ndarray

**fit** (*self*, *X*, *y=None*)

Fits component to data

**Parameters**

- **X** (*list*, *pd.DataFrame* or *np.ndarray*) – The input training data of shape [n\_samples, n\_features]
- **y** (*list*, *pd.Series*, *np.ndarray*, *optional*) – The target training data of length [n\_samples]

**Returns** self

**static load** (*file\_path*)

Loads component at file path

**Parameters** **file\_path** (*str*) – Location to load file

**Returns** ComponentBase object

**needs\_fitting** (*self*)

Returns boolean determining if component needs fitting before calling predict, predict\_proba, transform, or feature\_importances. This can be overridden to False for components that do not need to be fit or whose fit methods do nothing.

**property parameters** (*self*)

Returns the parameters which were used to initialize the component

**predict** (*self*, *X*)

Make predictions using selected features.

**Parameters** **X** (*pd.DataFrame*, *np.ndarray*) – Data of shape [n\_samples, n\_features]

**Returns** Predicted values

**Return type** pd.Series

**predict\_proba** (*self*, *X*)

Make probability estimates for labels.

**Parameters** **X** (*pd.DataFrame*, or *np.ndarray*) – Features

**Returns** Probability estimates

**Return type** pd.Series

**save** (*self*, *file\_path*, *pickle\_protocol=cloudpickle.DEFAULT\_PROTOCOL*)

Saves component at file path

**Parameters**

- **file\_path** (*str*) – Location to save file
- **pickle\_protocol** (*int*) – The pickle data stream format.

**Returns** None

**class** evalml.pipelines.**ComponentGraph** (*component\_dict=None, random\_seed=0*)

Component graph for a pipeline as a directed acyclic graph (DAG).

#### Parameters

- **component\_dict** (*dict*) – A dictionary which specifies the components and edges between components that should be used to create the component graph. Defaults to None.
- **random\_seed** (*int*) – Seed for the random number generator. Defaults to 0.

#### Example

```
>>> component_dict = {'imputer': ['Imputer'], 'ohe': ['One Hot Encoder', 'imputer.
↪x'], 'estimator_1': ['Random Forest Classifier', 'ohe.x'], 'estimator_2': [
↪'Decision Tree Classifier', 'ohe.x'], 'final': ['Logistic Regression Classifier
↪', 'estimator_1', 'estimator_2']}
>>> component_graph = ComponentGraph(component_dict)
```

#### Methods

<code>compute_final_component_features</code>	Transform all components save the final one, and gathers the data from any number of parents
<code>compute_order</code>	The order that components will be computed or called in.
<code>default_parameters</code>	The default parameter dictionary for this pipeline.
<code>describe</code>	Outputs component graph details including component parameters
<code>fit</code>	Fit each component in the graph.
<code>fit_features</code>	Fit all components save the final one, usually an estimator.
<code>generate_order</code>	Regenerated the topologically sorted order of the graph
<code>get_component</code>	Retrieves a single component object from the graph.
<code>get_estimators</code>	Gets a list of all the estimator components within this graph.
<code>get_inputs</code>	Retrieves all inputs for a given component.
<code>get_last_component</code>	Retrieves the component that is computed last in the graph, usually the final estimator.
<code>graph</code>	Generate an image representing the component graph
<code>instantiate</code>	Instantiates all uninstantiated components within the graph using the given parameters. An error will be
<code>inverse_transform</code>	Apply component <code>inverse_transform</code> methods to estimator predictions in reverse order.
<code>predict</code>	Make predictions using selected features.

**compute\_final\_component\_features** (*self, X, y=None*)

Transform all components save the final one, and gathers the data from any number of parents to get all the information that should be fed to the final component.

#### Parameters

- **X** (*pd.DataFrame*) – Data of shape `[n_samples, n_features]`.
- **y** (*pd.Series*) – The target training data of length `[n_samples]`. Defaults to None.

**Returns** Transformed values.

**Return type** `pd.DataFrame`

**property** `compute_order` (*self*)

The order that components will be computed or called in.

**property** `default_parameters` (*self*)

The default parameter dictionary for this pipeline.

**Returns** Dictionary of all component default parameters.

**Return type** `dict`

**describe** (*self*, *return\_dict=False*)

Outputs component graph details including component parameters

**Parameters** `return_dict` (*bool*) – If True, return dictionary of information about component graph. Defaults to False.

**Returns** Dictionary of all component parameters if `return_dict` is True, else None

**Return type** `dict`

**fit** (*self*, *X*, *y*)

Fit each component in the graph.

**Parameters**

- **X** (*pd.DataFrame*) – The input training data of shape `[n_samples, n_features]`.
- **y** (*pd.Series*) – The target training data of length `[n_samples]`.

**fit\_features** (*self*, *X*, *y*)

Fit all components save the final one, usually an estimator.

**Parameters**

- **X** (*pd.DataFrame*) – The input training data of shape `[n_samples, n_features]`.
- **y** (*pd.Series*) – The target training data of length `[n_samples]`.

**Returns** Transformed values.

**Return type** `pd.DataFrame`

**classmethod** `generate_order` (*cls*, *component\_dict*)

Regenerated the topologically sorted order of the graph

**get\_component** (*self*, *component\_name*)

Retrieves a single component object from the graph.

**Parameters** `component_name` (*str*) – Name of the component to retrieve

**Returns** `ComponentBase` object

**get\_estimators** (*self*)

Gets a list of all the estimator components within this graph.

**Returns** All estimator objects within the graph.

**Return type** `list`

**get\_inputs** (*self*, *component\_name*)

Retrieves all inputs for a given component.

**Parameters** `component_name` (*str*) – Name of the component to look up.

**Returns** List of inputs for the component to use.

**Return type** list[str]

**get\_last\_component** (*self*)

Retrieves the component that is computed last in the graph, usually the final estimator.

**Returns** ComponentBase object

**graph** (*self*, *name=None*, *graph\_format=None*)

Generate an image representing the component graph

**Parameters**

- **name** (*str*) – Name of the graph. Defaults to None.
- **graph\_format** (*str*) – file format to save the graph in. Defaults to None.

**Returns** Graph object that can be directly displayed in Jupyter notebooks.

**Return type** graphviz.Digraph

**instantiate** (*self*, *parameters*)

Instantiates all uninstantiated components within the graph using the given parameters. An error will be raised if a component is already instantiated but the parameters dict contains arguments for that component.

**Parameters** **parameters** (*dict*) – Dictionary with component names as keys and dictionary of that component’s parameters as values. An empty dictionary {} or None implies using all default values for component parameters.

**inverse\_transform** (*self*, *y*)

Apply component inverse\_transform methods to estimator predictions in reverse order.

Components that implement inverse\_transform are PolynomialDetrender, LabelEncoder (tbd).

**Parameters** **y** – (pd.Series): Final component features

**predict** (*self*, *X*)

Make predictions using selected features.

**Parameters** **X** (*pd.DataFrame*) – Data of shape [n\_samples, n\_features].

**Returns** Predicted values.

**Return type** pd.Series

```
class evalml.pipelines.DecisionTreeClassifier (criterion='gini', max_features='auto',  
                                              max_depth=6, min_samples_split=2,  
                                              min_weight_fraction_leaf=0.0, random_seed=0, **kwargs)
```

Decision Tree Classifier.

**Parameters**

- **criterion** ({*"gini"*, *"entropy"*}) – The function to measure the quality of a split. Supported criteria are “gini” for the Gini impurity and “entropy” for the information gain. Defaults to “gini”.
- **max\_features** (*int*, *float* or {*"auto"*, *"sqrt"*, *"log2"*}) – The number of features to consider when looking for the best split:
  - If int, then consider max\_features features at each split.
  - If float, then max\_features is a fraction and int(max\_features \* n\_features) features are considered at each split.
  - If “auto”, then max\_features=sqrt(n\_features).

- If “sqrt”, then `max_features=sqrt(n_features)`.
- If “log2”, then `max_features=log2(n_features)`.
- If None, then `max_features = n_features`.

The search for a split does not stop until at least one valid partition of the node samples is found, even if it requires to effectively inspect more than `max_features` features. Defaults to “auto”.

- **`max_depth`** (*int*) – The maximum depth of the tree. Defaults to 6.
- **`min_samples_split`** (*int or float*) – The minimum number of samples required to split an internal node:
  - If int, then consider `min_samples_split` as the minimum number.
  - If float, then `min_samples_split` is a fraction and `ceil(min_samples_split * n_samples)` are the minimum number of samples for each split.

Defaults to 2.

- **`min_weight_fraction_leaf`** (*float*) – The minimum weighted fraction of the sum total of weights (of all the input samples) required to be at a leaf node. Defaults to 0.0.
- **`random_seed`** (*int*) – Seed for the random number generator. Defaults to 0.

#### Attributes

<b>hyper-parameter_ranges</b>	{ “criterion”: [“gini”, “entropy”], “max_features”: [“auto”, “sqrt”, “log2”], “max_depth”: Integer(4, 10), }
<b>model_family</b>	ModelFamily.DECISION_TREE
<b>modifies_features</b>	True
<b>modifies_target</b>	False
<b>name</b>	Decision Tree Classifier
<b>predict_uses_y</b>	False
<b>supported_problem_types</b>	[ ProblemTypes.BINARY, ProblemTypes.MULTICLASS, ProblemTypes.TIME_SERIES_BINARY, ProblemTypes.TIME_SERIES_MULTICLASS, ]

#### Methods

<code>clone</code>	Constructs a new component with the same parameters and random state.
<code>default_parameters</code>	Returns the default parameters for this component.
<code>describe</code>	Describe a component and its parameters
<code>feature_importance</code>	Returns importance associated with each feature.
<code>fit</code>	Fits component to data
<code>load</code>	Loads component at file path
<code>needs_fitting</code>	Returns boolean determining if component needs fitting before
<code>parameters</code>	Returns the parameters which were used to initialize the component
<code>predict</code>	Make predictions using selected features.

continues on next page



Table 634 – continued from previous page

<code>predict_proba</code>	Make probability estimates for labels.
<code>save</code>	Saves component at file path

**clone** (*self*)

Constructs a new component with the same parameters and random state.

**Returns** A new instance of this component with identical parameters and random state.

**default\_parameters** (*cls*)

Returns the default parameters for this component.

Our convention is that `Component.default_parameters == Component().parameters`.

**Returns** default parameters for this component.

**Return type** dict

**describe** (*self*, *print\_name=False*, *return\_dict=False*)

Describe a component and its parameters

**Parameters**

- **print\_name** (*bool*, *optional*) – whether to print name of component
- **return\_dict** (*bool*, *optional*) – whether to return description as dictionary in the format {"name": name, "parameters": parameters}

**Returns** prints and returns dictionary

**Return type** None or dict

**property feature\_importance** (*self*)

Returns importance associated with each feature.

**Returns** Importance associated with each feature

**Return type** np.ndarray

**fit** (*self*, *X*, *y=None*)

Fits component to data

**Parameters**

- **X** (*list*, *pd.DataFrame* or *np.ndarray*) – The input training data of shape [n\_samples, n\_features]
- **y** (*list*, *pd.Series*, *np.ndarray*, *optional*) – The target training data of length [n\_samples]

**Returns** self

**static load** (*file\_path*)

Loads component at file path

**Parameters** **file\_path** (*str*) – Location to load file

**Returns** ComponentBase object

**needs\_fitting** (*self*)

Returns boolean determining if component needs fitting before calling predict, predict\_proba, transform, or feature\_importances. This can be overridden to False for components that do not need to be fit or whose fit methods do nothing.

**property parameters** (*self*)

Returns the parameters which were used to initialize the component

**predict** (*self*, *X*)

Make predictions using selected features.

**Parameters** *X* (*pd.DataFrame*, *np.ndarray*) – Data of shape [n\_samples, n\_features]

**Returns** Predicted values

**Return type** *pd.Series*

**predict\_proba** (*self*, *X*)

Make probability estimates for labels.

**Parameters** *X* (*pd.DataFrame*, or *np.ndarray*) – Features

**Returns** Probability estimates

**Return type** *pd.Series*

**save** (*self*, *file\_path*, *pickle\_protocol=cloudpickle.DEFAULT\_PROTOCOL*)

Saves component at file path

**Parameters**

- **file\_path** (*str*) – Location to save file
- **pickle\_protocol** (*int*) – The pickle data stream format.

**Returns** None

```
class evalml.pipelines.DecisionTreeRegressor (criterion='mse', max_features='auto',  
                                              max_depth=6, min_samples_split=2,  
                                              min_weight_fraction_leaf=0.0, random_seed=0, **kwargs)
```

Decision Tree Regressor.

**Parameters**

- **criterion** (*{ "mse", "friedman\_mse", "mae", "poisson" }*) – The function to measure the quality of a split. Supported criteria are:
  - “mse” for the mean squared error, which is equal to variance reduction as feature selection criterion and minimizes the L2 loss using the mean of each terminal node
  - “friedman\_mse”, which uses mean squared error with Friedman’s improvement score for potential splits
  - “mae” for the mean absolute error, which minimizes the L1 loss using the median of each terminal node,
  - “poisson” which uses reduction in Poisson deviance to find splits.
- **max\_features** (*int, float or { "auto", "sqrt", "log2" }*) – The number of features to consider when looking for the best split:
  - If *int*, then consider *max\_features* features at each split.
  - If *float*, then *max\_features* is a fraction and *int(max\_features \* n\_features)* features are considered at each split.
  - If “auto”, then *max\_features=sqrt(n\_features)*.
  - If “sqrt”, then *max\_features=sqrt(n\_features)*.
  - If “log2”, then *max\_features=log2(n\_features)*.
  - If *None*, then *max\_features = n\_features*.

The search for a split does not stop until at least one valid partition of the node samples is found, even if it requires to effectively inspect more than `max_features` features.

- **max\_depth** (*int*) – The maximum depth of the tree. Defaults to 6.
- **min\_samples\_split** (*int or float*) – The minimum number of samples required to split an internal node:
  - If *int*, then consider `min_samples_split` as the minimum number.
  - If *float*, then `min_samples_split` is a fraction and `ceil(min_samples_split * n_samples)` are the minimum number of samples for each split.

Defaults to 2.

- **min\_weight\_fraction\_leaf** (*float*) – The minimum weighted fraction of the sum total of weights (of all the input samples) required to be at a leaf node. Defaults to 0.0.
- **random\_seed** (*int*) – Seed for the random number generator. Defaults to 0.

#### Attributes

<b>hyper-parameter_ranges</b>	{ "criterion": ["mse", "friedman_mse", "mae"], "max_features": ["auto", "sqrt", "log2"], "max_depth": Integer(4, 10), }
<b>model_family</b>	ModelFamily.DECISION_TREE
<b>modifies_features</b>	True
<b>modifies_target</b>	False
<b>name</b>	Decision Tree Regressor
<b>predict_uses_y</b>	False
<b>supported_problem_types</b>	[ ProblemTypes.REGRESSION, ProblemTypes.TIME_SERIES_REGRESSION, ]

#### Methods

<i>clone</i>	Constructs a new component with the same parameters and random state.
<i>default_parameters</i>	Returns the default parameters for this component.
<i>describe</i>	Describe a component and its parameters
<i>feature_importance</i>	Returns importance associated with each feature.
<i>fit</i>	Fits component to data
<i>load</i>	Loads component at file path
<i>needs_fitting</i>	Returns boolean determining if component needs fitting before
<i>parameters</i>	Returns the parameters which were used to initialize the component
<i>predict</i>	Make predictions using selected features.
<i>predict_proba</i>	Make probability estimates for labels.
<i>save</i>	Saves component at file path

#### **clone** (*self*)

Constructs a new component with the same parameters and random state.

**Returns** A new instance of this component with identical parameters and random state.

**default\_parameters** (*cls*)

Returns the default parameters for this component.

Our convention is that `Component.default_parameters == Component().parameters`.

**Returns** default parameters for this component.

**Return type** dict

**describe** (*self*, *print\_name=False*, *return\_dict=False*)

Describe a component and its parameters

**Parameters**

- **print\_name** (*bool*, *optional*) – whether to print name of component
- **return\_dict** (*bool*, *optional*) – whether to return description as dictionary in the format {"name": name, "parameters": parameters}

**Returns** prints and returns dictionary

**Return type** None or dict

**property feature\_importance** (*self*)

Returns importance associated with each feature.

**Returns** Importance associated with each feature

**Return type** np.ndarray

**fit** (*self*, *X*, *y=None*)

Fits component to data

**Parameters**

- **X** (*list*, *pd.DataFrame* or *np.ndarray*) – The input training data of shape [n\_samples, n\_features]
- **y** (*list*, *pd.Series*, *np.ndarray*, *optional*) – The target training data of length [n\_samples]

**Returns** self

**static load** (*file\_path*)

Loads component at file path

**Parameters** **file\_path** (*str*) – Location to load file

**Returns** ComponentBase object

**needs\_fitting** (*self*)

Returns boolean determining if component needs fitting before calling predict, predict\_proba, transform, or feature\_importances. This can be overridden to False for components that do not need to be fit or whose fit methods do nothing.

**property parameters** (*self*)

Returns the parameters which were used to initialize the component

**predict** (*self*, *X*)

Make predictions using selected features.

**Parameters** **X** (*pd.DataFrame*, *np.ndarray*) – Data of shape [n\_samples, n\_features]

**Returns** Predicted values

**Return type** pd.Series

**predict\_proba** (*self*, *X*)

Make probability estimates for labels.

**Parameters** *X* (*pd.DataFrame*, or *np.ndarray*) – Features

**Returns** Probability estimates

**Return type** *pd.Series*

**save** (*self*, *file\_path*, *pickle\_protocol=cloudpickle.DEFAULT\_PROTOCOL*)

Saves component at file path

**Parameters**

- **file\_path** (*str*) – Location to save file
- **pickle\_protocol** (*int*) – The pickle data stream format.

**Returns** None

**class** evalml.pipelines.DelayedFeatureTransformer (*date\_index=None*, *max\_delay=2*,  
*delay\_features=True*, *delay\_target=True*, *gap=1*, *random\_seed=0*, *\*\*kwargs*)

Transformer that delays input features and target variable for time series problems.

**Parameters**

- **date\_index** (*str*) – Name of the column containing the datetime information used to order the data. Ignored.
- **max\_delay** (*int*) – Maximum number of time units to delay each feature. Defaults to 2.
- **delay\_features** (*bool*) – Whether to delay the input features. Defaults to True.
- **delay\_target** (*bool*) – Whether to delay the target. Defaults to True.
- **gap** (*int*) – The number of time units between when the features are collected and when the target is collected. For example, if you are predicting the next time step’s target, *gap=1*. This is only needed because when *gap=0*, we need to be sure to start the lagging of the target variable at 1. Defaults to 1.
- **random\_seed** (*int*) – Seed for the random number generator. This transformer performs the same regardless of the random seed provided.

**Attributes**

<b>hyper-parameter_ranges</b>	{}
<b>model_family</b>	ModelFamily.NONE
<b>modifies_features</b>	True
<b>modifies_target</b>	False
<b>name</b>	Delayed Feature Transformer
<b>needs_fitting</b>	False

**Methods**

<code>clone</code>	Constructs a new component with the same parameters and random state.
<code>default_parameters</code>	Returns the default parameters for this component.
<code>describe</code>	Describe a component and its parameters
<code>fit</code>	Fits the DelayFeatureTransformer.
<code>fit_transform</code>	Fits on X and transforms X
<code>load</code>	Loads component at file path
<code>parameters</code>	Returns the parameters which were used to initialize the component
<code>save</code>	Saves component at file path
<code>transform</code>	Computes the delayed features for all features in X and y.

**clone** (*self*)

Constructs a new component with the same parameters and random state.

**Returns** A new instance of this component with identical parameters and random state.

**default\_parameters** (*cls*)

Returns the default parameters for this component.

Our convention is that `Component.default_parameters == Component().parameters`.

**Returns** default parameters for this component.

**Return type** dict

**describe** (*self*, *print\_name=False*, *return\_dict=False*)

Describe a component and its parameters

**Parameters**

- **print\_name** (*bool*, *optional*) – whether to print name of component
- **return\_dict** (*bool*, *optional*) – whether to return description as dictionary in the format {"name": name, "parameters": parameters}

**Returns** prints and returns dictionary

**Return type** None or dict

**fit** (*self*, *X*, *y=None*)

Fits the DelayFeatureTransformer.

**Parameters**

- **X** (*pd.DataFrame* or *np.ndarray*) – The input training data of shape [n\_samples, n\_features]
- **y** (*pd.Series*, *optional*) – The target training data of length [n\_samples]

**Returns** self

**fit\_transform** (*self*, *X*, *y*)

Fits on X and transforms X

**Parameters**

- **X** (*pd.DataFrame*) – Data to fit and transform
- **y** (*pd.Series*) – Target data

**Returns** Transformed X

**Return type** `pd.DataFrame`

**static load** (*file\_path*)

Loads component at file path

**Parameters** *file\_path* (*str*) – Location to load file

**Returns** `ComponentBase` object

**property parameters** (*self*)

Returns the parameters which were used to initialize the component

**save** (*self*, *file\_path*, *pickle\_protocol=cloudpickle.DEFAULT\_PROTOCOL*)

Saves component at file path

**Parameters**

- **file\_path** (*str*) – Location to save file
- **pickle\_protocol** (*int*) – The pickle data stream format.

**Returns** `None`

**transform** (*self*, *X*, *y=None*)

Computes the delayed features for all features in *X* and *y*.

For each feature in *X*, it will add a column to the output dataframe for each delay in the (inclusive) range `[1, max_delay]`. The values of each delayed feature are simply the original feature shifted forward in time by the delay amount. For example, a delay of 3 units means that the feature value at row *n* will be taken from the *n*-3rd row of that feature

If *y* is not `None`, it will also compute the delayed values for the target variable.

**Parameters**

- **X** (*pd.DataFrame* or *None*) – Data to transform. `None` is expected when only the target variable is being used.
- **y** (*pd.Series*, or *None*) – Target.

**Returns** Transformed *X*.

**Return type** `pd.DataFrame`

**class** `evalml.pipelines.DFSTransformer` (*index='index'*, *random\_seed=0*, *\*\*kwargs*)

Featuretools DFS component that generates features for the input features.

**Parameters**

- **index** (*string*) – The name of the column that contains the indices. If no column with this name exists, then `featuretools.EntitySet()` creates a column with this name to serve as the index column. Defaults to `'index'`.
- **random\_seed** (*int*) – Seed for the random number generator. Defaults to 0.

**Attributes**

<b>hyper-parameter_ranges</b>	{}
<b>model_family</b>	ModelFamily.NONE
<b>modifies_features</b>	True
<b>modifies_target</b>	False
<b>name</b>	DFS Transformer

## Methods

<i>clone</i>	Constructs a new component with the same parameters and random state.
<i>default_parameters</i>	Returns the default parameters for this component.
<i>describe</i>	Describe a component and its parameters
<i>fit</i>	Fits the DFSTransformer Transformer component.
<i>fit_transform</i>	Fits on X and transforms X
<i>load</i>	Loads component at file path
<i>needs_fitting</i>	Returns boolean determining if component needs fitting before
<i>parameters</i>	Returns the parameters which were used to initialize the component
<i>save</i>	Saves component at file path
<i>transform</i>	Computes the feature matrix for the input X using featuretools' dfs algorithm.

### **clone** (*self*)

Constructs a new component with the same parameters and random state.

**Returns** A new instance of this component with identical parameters and random state.

### **default\_parameters** (*cls*)

Returns the default parameters for this component.

Our convention is that `Component.default_parameters == Component().parameters`.

**Returns** default parameters for this component.

**Return type** dict

### **describe** (*self*, *print\_name=False*, *return\_dict=False*)

Describe a component and its parameters

#### Parameters

- **print\_name** (*bool*, *optional*) – whether to print name of component
- **return\_dict** (*bool*, *optional*) – whether to return description as dictionary in the format {"name": name, "parameters": parameters}

**Returns** prints and returns dictionary

**Return type** None or dict

### **fit** (*self*, *X*, *y=None*)

Fits the DFSTransformer Transformer component.



**Parameters**

- **X** (*pd.DataFrame, np.array*) – The input data to transform, of shape [n\_samples, n\_features]
- **y** (*pd.Series, np.ndarray, optional*) – The target training data of length [n\_samples]

**Returns** self**fit\_transform** (*self, X, y=None*)

Fits on X and transforms X

**Parameters**

- **X** (*pd.DataFrame*) – Data to fit and transform
- **y** (*pd.Series*) – Target data

**Returns** Transformed X**Return type** pd.DataFrame**static load** (*file\_path*)

Loads component at file path

**Parameters** **file\_path** (*str*) – Location to load file**Returns** ComponentBase object**needs\_fitting** (*self*)

Returns boolean determining if component needs fitting before calling predict, predict\_proba, transform, or feature\_importances. This can be overridden to False for components that do not need to be fit or whose fit methods do nothing.

**property parameters** (*self*)

Returns the parameters which were used to initialize the component

**save** (*self, file\_path, pickle\_protocol=cloudpickle.DEFAULT\_PROTOCOL*)

Saves component at file path

**Parameters**

- **file\_path** (*str*) – Location to save file
- **pickle\_protocol** (*int*) – The pickle data stream format.

**Returns** None**transform** (*self, X, y=None*)

Computes the feature matrix for the input X using featuretools' dfs algorithm.

**Parameters**

- **X** (*pd.DataFrame or np.ndarray*) – The input training data to transform. Has shape [n\_samples, n\_features]
- **y** (*pd.Series, optional*) – Ignored.

**Returns** Feature matrix**Return type** pd.DataFrame

**class** evalml.pipelines.**ElasticNetClassifier** (*penalty='elasticnet', C=1.0, l1\_ratio=0.15, multi\_class='auto', solver='saga', n\_jobs=-1, random\_seed=0, \*\*kwargs*)

Elastic Net Classifier. Uses Logistic Regression with elasticnet penalty as the base estimator.

## Parameters

- **penalty** (`{"l1", "l2", "elasticnet", "none"}`) – The norm used in penalization. Defaults to “elasticnet”.
- **C** (`float`) – Inverse of regularization strength. Must be a positive float. Defaults to 1.0.
- **l1\_ratio** (`float`) – The mixing parameter, with  $0 \leq \text{l1\_ratio} \leq 1$ . Only used if `penalty='elasticnet'`. Setting `l1_ratio=0` is equivalent to using `penalty='l2'`, while setting `l1_ratio=1` is equivalent to using `penalty='l1'`. For  $0 < \text{l1\_ratio} < 1$ , the penalty is a combination of L1 and L2. Defaults to 0.15.
- **multi\_class** (`{"auto", "ovr", "multinomial"}`) – If the option chosen is “ovr”, then a binary problem is fit for each label. For “multinomial” the loss minimised is the multinomial loss fit across the entire probability distribution, even when the data is binary. “multinomial” is unavailable when `solver='liblinear'`. “auto” selects “ovr” if the data is binary, or if `solver='liblinear'`, and otherwise selects “multinomial”. Defaults to “auto”.
- **solver** (`{"newton-cg", "lbfgs", "liblinear", "sag", "saga"}`) – Algorithm to use in the optimization problem. For small datasets, “liblinear” is a good choice, whereas “sag” and “saga” are faster for large ones. For multiclass problems, only “newton-cg”, “sag”, “saga” and “lbfgs” handle multinomial loss; “liblinear” is limited to one-versus-rest schemes.
  - “newton-cg”, “lbfgs”, “sag” and “saga” handle L2 or no penalty
  - “liblinear” and “saga” also handle L1 penalty
  - “saga” also supports “elasticnet” penalty
  - “liblinear” does not support setting `penalty='none'`
 Defaults to “saga”.
- **n\_jobs** (`int`) – Number of parallel threads used to run xgboost. Note that creating thread contention will significantly slow down the algorithm. Defaults to -1.
- **random\_seed** (`int`) – Seed for the random number generator. Defaults to 0.

## Attributes

<b>hyper-parameter_ranges</b>	{ “C”: Real(0.01, 10), “l1_ratio”: Real(0, 1)}
<b>model_family</b>	ModelFamily.LINEAR_MODEL
<b>modifies_features</b>	True
<b>modifies_target</b>	False
<b>name</b>	Elastic Net Classifier
<b>predict Uses y</b>	False
<b>supported_problem_types</b>	[ ProblemTypes.BINARY, ProblemTypes.MULTICLASS, ProblemTypes.TIME_SERIES_BINARY, ProblemTypes.TIME_SERIES_MULTICLASS,]

## Methods

<code>clone</code>	Constructs a new component with the same parameters and random state.
<code>default_parameters</code>	Returns the default parameters for this component.
<code>describe</code>	Describe a component and its parameters
<code>feature_importance</code>	Returns importance associated with each feature.
<code>fit</code>	Fits component to data
<code>load</code>	Loads component at file path
<code>needs_fitting</code>	Returns boolean determining if component needs fitting before
<code>parameters</code>	Returns the parameters which were used to initialize the component
<code>predict</code>	Make predictions using selected features.
<code>predict_proba</code>	Make probability estimates for labels.
<code>save</code>	Saves component at file path

**clone** (*self*)

Constructs a new component with the same parameters and random state.

**Returns** A new instance of this component with identical parameters and random state.

**default\_parameters** (*cls*)

Returns the default parameters for this component.

Our convention is that `Component.default_parameters == Component().parameters`.

**Returns** default parameters for this component.

**Return type** dict

**describe** (*self*, *print\_name=False*, *return\_dict=False*)

Describe a component and its parameters

**Parameters**

- **print\_name** (*bool*, *optional*) – whether to print name of component
- **return\_dict** (*bool*, *optional*) – whether to return description as dictionary in the format {"name": name, "parameters": parameters}

**Returns** prints and returns dictionary

**Return type** None or dict

**property feature\_importance** (*self*)

Returns importance associated with each feature.

**Returns** Importance associated with each feature

**Return type** np.ndarray

**fit** (*self*, *X*, *y*)

Fits component to data

**Parameters**

- **X** (*list*, *pd.DataFrame* or *np.ndarray*) – The input training data of shape [n\_samples, n\_features]
- **y** (*list*, *pd.Series*, *np.ndarray*, *optional*) – The target training data of length [n\_samples]

**Returns** self

**static load** (*file\_path*)

Loads component at file path

**Parameters** *file\_path* (*str*) – Location to load file

**Returns** ComponentBase object

**needs\_fitting** (*self*)

Returns boolean determining if component needs fitting before calling predict, predict\_proba, transform, or feature\_importances. This can be overridden to False for components that do not need to be fit or whose fit methods do nothing.

**property parameters** (*self*)

Returns the parameters which were used to initialize the component

**predict** (*self*, *X*)

Make predictions using selected features.

**Parameters** *X* (*pd.DataFrame*, *np.ndarray*) – Data of shape [n\_samples, n\_features]

**Returns** Predicted values

**Return type** *pd.Series*

**predict\_proba** (*self*, *X*)

Make probability estimates for labels.

**Parameters** *X* (*pd.DataFrame*, or *np.ndarray*) – Features

**Returns** Probability estimates

**Return type** *pd.Series*

**save** (*self*, *file\_path*, *pickle\_protocol=cloudpickle.DEFAULT\_PROTOCOL*)

Saves component at file path

**Parameters**

- **file\_path** (*str*) – Location to save file
- **pickle\_protocol** (*int*) – The pickle data stream format.

**Returns** None

**class** evalml.pipelines.ElasticNetRegressor (*alpha=0.0001*, *l1\_ratio=0.15*,  
*max\_iter=1000*, *normalize=False*, *random\_seed=0*, *\*\*kwargs*)

Elastic Net Regressor.

**Parameters**

- **alpha** (*float*) – Constant that multiplies the penalty terms. Defaults to 0.0001.
- **l1\_ratio** (*float*) – The mixing parameter, with  $0 \leq \text{l1\_ratio} \leq 1$ . Only used if *penalty='elasticnet'*. Setting *l1\_ratio=0* is equivalent to using *penalty='l2'*, while setting *l1\_ratio=1* is equivalent to using *penalty='l1'*. For  $0 < \text{l1\_ratio} < 1$ , the penalty is a combination of L1 and L2. Defaults to 0.15.
- **max\_iter** (*int*) – The maximum number of iterations. Defaults to 1000.
- **normalize** (*boolean*) – If True, the regressors will be normalized before regression by subtracting the mean and dividing by the l2-norm. Defaults to False.
- **random\_seed** (*int*) – Seed for the random number generator. Defaults to 0.

### Attributes

<b>hyper-parameter_ranges</b>	{ "alpha": Real(0, 1), "l1_ratio": Real(0, 1), }
<b>model_family</b>	ModelFamily.LINEAR_MODEL
<b>modifies_features</b>	True
<b>modifies_target</b>	False
<b>name</b>	Elastic Net Regressor
<b>pre_dict_uses_y</b>	False
<b>supported_problem_types</b>	[ ProblemTypes.REGRESSION, ProblemTypes.TIME_SERIES_REGRESSION, ]

### Methods

<i>clone</i>	Constructs a new component with the same parameters and random state.
<i>default_parameters</i>	Returns the default parameters for this component.
<i>describe</i>	Describe a component and its parameters
<i>feature_importance</i>	Returns importance associated with each feature.
<i>fit</i>	Fits component to data
<i>load</i>	Loads component at file path
<i>needs_fitting</i>	Returns boolean determining if component needs fitting before
<i>parameters</i>	Returns the parameters which were used to initialize the component
<i>predict</i>	Make predictions using selected features.
<i>predict_proba</i>	Make probability estimates for labels.
<i>save</i>	Saves component at file path

#### **clone** (*self*)

Constructs a new component with the same parameters and random state.

**Returns** A new instance of this component with identical parameters and random state.

#### **default\_parameters** (*cls*)

Returns the default parameters for this component.

Our convention is that `Component.default_parameters == Component().parameters`.

**Returns** default parameters for this component.

**Return type** dict

#### **describe** (*self*, *print\_name=False*, *return\_dict=False*)

Describe a component and its parameters

#### Parameters

- **print\_name** (*bool*, *optional*) – whether to print name of component
- **return\_dict** (*bool*, *optional*) – whether to return description as dictionary in the format {“name”: name, “parameters”: parameters}

**Returns** prints and returns dictionary

**Return type** None or dict

**property** `feature_importance` (*self*)

Returns importance associated with each feature.

**Returns** Importance associated with each feature

**Return type** np.ndarray

**fit** (*self*, *X*, *y=None*)

Fits component to data

**Parameters**

- **X** (*list*, *pd.DataFrame* or *np.ndarray*) – The input training data of shape [n\_samples, n\_features]
- **y** (*list*, *pd.Series*, *np.ndarray*, *optional*) – The target training data of length [n\_samples]

**Returns** self

**static** `load` (*file\_path*)

Loads component at file path

**Parameters** `file_path` (*str*) – Location to load file

**Returns** ComponentBase object

**needs\_fitting** (*self*)

Returns boolean determining if component needs fitting before calling predict, predict\_proba, transform, or feature\_importances. This can be overridden to False for components that do not need to be fit or whose fit methods do nothing.

**property** `parameters` (*self*)

Returns the parameters which were used to initialize the component

**predict** (*self*, *X*)

Make predictions using selected features.

**Parameters** **X** (*pd.DataFrame*, *np.ndarray*) – Data of shape [n\_samples, n\_features]

**Returns** Predicted values

**Return type** pd.Series

**predict\_proba** (*self*, *X*)

Make probability estimates for labels.

**Parameters** **X** (*pd.DataFrame*, or *np.ndarray*) – Features

**Returns** Probability estimates

**Return type** pd.Series

**save** (*self*, *file\_path*, *pickle\_protocol=cloudpickle.DEFAULT\_PROTOCOL*)

Saves component at file path

**Parameters**

- **file\_path** (*str*) – Location to save file
- **pickle\_protocol** (*int*) – The pickle data stream format.

**Returns** None

```
class evalml.pipelines.Estimator (parameters=None, component_obj=None, random_seed=0,
                                   **kwargs)
```

A component that fits and predicts given data.

To implement a new Estimator, define your own class which is a subclass of Estimator, including a name and a list of acceptable ranges for any parameters to be tuned during the automl search (hyperparameters). Define an `__init__` method which sets up any necessary state and objects. Make sure your `__init__` only uses standard keyword arguments and calls `super().__init__()` with a parameters dict. You may also override the `fit`, `transform`, `fit_transform` and other methods in this class if appropriate.

To see some examples, check out the definitions of any Estimator component.

### Parameters

- **parameters** (*dict*) – Dictionary of parameters for the component. Defaults to None.
- **component\_obj** (*obj*) – Third-party objects useful in component implementation. Defaults to None.
- **random\_seed** (*int*) – Seed for the random number generator. Defaults to 0.

### Attributes

<b>model_family</b>	ModelFamily.NONE
<b>modifies_features</b>	True
<b>modifies_target</b>	False
<b>pre-dict_uses_y</b>	False

### Methods

<i>clone</i>	Constructs a new component with the same parameters and random state.
<i>default_parameters</i>	Returns the default parameters for this component.
<i>describe</i>	Describe a component and its parameters
<i>feature_importance</i>	Returns importance associated with each feature.
<i>fit</i>	Fits component to data
<i>load</i>	Loads component at file path
<i>name</i>	Returns string name of this component
<i>needs_fitting</i>	Returns boolean determining if component needs fitting before
<i>parameters</i>	Returns the parameters which were used to initialize the component
<i>predict</i>	Make predictions using selected features.
<i>predict_proba</i>	Make probability estimates for labels.
<i>save</i>	Saves component at file path
<i>supported_problem_types</i>	Problem types this estimator supports

**clone** (*self*)

Constructs a new component with the same parameters and random state.

**Returns** A new instance of this component with identical parameters and random state.

**default\_parameters** (*cls*)

Returns the default parameters for this component.

Our convention is that `Component.default_parameters == Component().parameters`.

**Returns** default parameters for this component.

**Return type** dict

**describe** (*self*, *print\_name=False*, *return\_dict=False*)

Describe a component and its parameters

**Parameters**

- **print\_name** (*bool*, *optional*) – whether to print name of component
- **return\_dict** (*bool*, *optional*) – whether to return description as dictionary in the format {"name": name, "parameters": parameters}

**Returns** prints and returns dictionary

**Return type** None or dict

**property feature\_importance** (*self*)

Returns importance associated with each feature.

**Returns** Importance associated with each feature

**Return type** np.ndarray

**fit** (*self*, *X*, *y=None*)

Fits component to data

**Parameters**

- **X** (*list*, *pd.DataFrame* or *np.ndarray*) – The input training data of shape [n\_samples, n\_features]
- **y** (*list*, *pd.Series*, *np.ndarray*, *optional*) – The target training data of length [n\_samples]

**Returns** self

**static load** (*file\_path*)

Loads component at file path

**Parameters** **file\_path** (*str*) – Location to load file

**Returns** ComponentBase object

**property name** (*cls*)

Returns string name of this component

**needs\_fitting** (*self*)

Returns boolean determining if component needs fitting before calling predict, predict\_proba, transform, or feature\_importances. This can be overridden to False for components that do not need to be fit or whose fit methods do nothing.

**property parameters** (*self*)

Returns the parameters which were used to initialize the component

**predict** (*self*, *X*)

Make predictions using selected features.

**Parameters** **X** (*pd.DataFrame*, *np.ndarray*) – Data of shape [n\_samples, n\_features]

**Returns** Predicted values

**Return type** pd.Series



**predict\_proba** (*self*, *X*)

Make probability estimates for labels.

**Parameters** *X* (*pd.DataFrame*, or *np.ndarray*) – Features

**Returns** Probability estimates

**Return type** *pd.Series*

**save** (*self*, *file\_path*, *pickle\_protocol=cloudpickle.DEFAULT\_PROTOCOL*)

Saves component at file path

**Parameters**

- **file\_path** (*str*) – Location to save file
- **pickle\_protocol** (*int*) – The pickle data stream format.

**Returns** None

**property supported\_problem\_types** (*cls*)

Problem types this estimator supports

```
class evalml.pipelines.ExtraTreesClassifier (n_estimators=100,      max_features='auto',
                                           max_depth=6,          min_samples_split=2,
                                           min_weight_fraction_leaf=0.0, n_jobs=-1,
                                           random_seed=0, **kwargs)
```

Extra Trees Classifier.

**Parameters**

- **n\_estimators** (*float*) – The number of trees in the forest. Defaults to 100.
- **max\_features** (*int*, *float* or {"auto", "sqrt", "log2"}) – The number of features to consider when looking for the best split:
  - If *int*, then consider *max\_features* features at each split.
  - If *float*, then *max\_features* is a fraction and *int(max\_features \* n\_features)* features are considered at each split.
  - If “auto”, then *max\_features*=*sqrt*(*n\_features*).
  - If “sqrt”, then *max\_features*=*sqrt*(*n\_features*).
  - If “log2”, then *max\_features*=*log2*(*n\_features*).
  - If *None*, then *max\_features* = *n\_features*.

The search for a split does not stop until at least one valid partition of the node samples is found, even if it requires to effectively inspect more than *max\_features* features. Defaults to “auto”.

- **max\_depth** (*int*) – The maximum depth of the tree. Defaults to 6.
- **min\_samples\_split** (*int* or *float*) – The minimum number of samples required to split an internal node:
  - If *int*, then consider *min\_samples\_split* as the minimum number.
  - If *float*, then *min\_samples\_split* is a fraction and *ceil(min\_samples\_split \* n\_samples)* are the minimum number of samples for each split.
- **to 2.** (*Defaults*) –
- **min\_weight\_fraction\_leaf** (*float*) – The minimum weighted fraction of the sum total of weights (of all the input samples) required to be at a leaf node. Defaults to 0.0.

- **n\_jobs** (*int* or *None*) – Number of jobs to run in parallel. -1 uses all processes. Defaults to -1.
- **random\_seed** (*int*) – Seed for the random number generator. Defaults to 0.

#### Attributes

<b>hyper-parameter_ranges</b>	{ “n_estimators”: Integer(10, 1000), “max_features”: [“auto”, “sqrt”, “log2”], “max_depth”: Integer(4, 10),}
<b>model_family</b>	ModelFamily.EXTRA_TREES
<b>modifies_features</b>	True
<b>modifies_target</b>	False
<b>name</b>	Extra Trees Classifier
<b>pre_dict_uses_y</b>	False
<b>supported_problem_types</b>	[ ProblemTypes.BINARY, ProblemTypes.MULTICLASS, ProblemTypes.TIME_SERIES_BINARY, ProblemTypes.TIME_SERIES_MULTICLASS,]

#### Methods

<i>clone</i>	Constructs a new component with the same parameters and random state.
<i>default_parameters</i>	Returns the default parameters for this component.
<i>describe</i>	Describe a component and its parameters
<i>feature_importance</i>	Returns importance associated with each feature.
<i>fit</i>	Fits component to data
<i>load</i>	Loads component at file path
<i>needs_fitting</i>	Returns boolean determining if component needs fitting before
<i>parameters</i>	Returns the parameters which were used to initialize the component
<i>predict</i>	Make predictions using selected features.
<i>predict_proba</i>	Make probability estimates for labels.
<i>save</i>	Saves component at file path

#### **clone** (*self*)

Constructs a new component with the same parameters and random state.

**Returns** A new instance of this component with identical parameters and random state.

#### **default\_parameters** (*cls*)

Returns the default parameters for this component.

Our convention is that `Component.default_parameters == Component().parameters`.

**Returns** default parameters for this component.

**Return type** dict

#### **describe** (*self*, *print\_name=False*, *return\_dict=False*)

Describe a component and its parameters

**Parameters**

- **print\_name** (*bool, optional*) – whether to print name of component
- **return\_dict** (*bool, optional*) – whether to return description as dictionary in the format {"name": name, "parameters": parameters}

**Returns** prints and returns dictionary

**Return type** None or dict

**property feature\_importance** (*self*)

Returns importance associated with each feature.

**Returns** Importance associated with each feature

**Return type** np.ndarray

**fit** (*self, X, y=None*)

Fits component to data

**Parameters**

- **X** (*list, pd.DataFrame or np.ndarray*) – The input training data of shape [n\_samples, n\_features]
- **y** (*list, pd.Series, np.ndarray, optional*) – The target training data of length [n\_samples]

**Returns** self

**static load** (*file\_path*)

Loads component at file path

**Parameters** **file\_path** (*str*) – Location to load file

**Returns** ComponentBase object

**needs\_fitting** (*self*)

Returns boolean determining if component needs fitting before calling predict, predict\_proba, transform, or feature\_importances. This can be overridden to False for components that do not need to be fit or whose fit methods do nothing.

**property parameters** (*self*)

Returns the parameters which were used to initialize the component

**predict** (*self, X*)

Make predictions using selected features.

**Parameters** **X** (*pd.DataFrame, np.ndarray*) – Data of shape [n\_samples, n\_features]

**Returns** Predicted values

**Return type** pd.Series

**predict\_proba** (*self, X*)

Make probability estimates for labels.

**Parameters** **X** (*pd.DataFrame, or np.ndarray*) – Features

**Returns** Probability estimates

**Return type** pd.Series

**save** (*self, file\_path, pickle\_protocol=cloudpickle.DEFAULT\_PROTOCOL*)

Saves component at file path

**Parameters**

- **file\_path** (*str*) – Location to save file
- **pickle\_protocol** (*int*) – The pickle data stream format.

Returns None

```
class evalml.pipelines.ExtraTreesRegressor (n_estimators=100,      max_features='auto',
                                             max_depth=6,         min_samples_split=2,
                                             min_weight_fraction_leaf=0.0,  n_jobs=-1,
                                             random_seed=0, **kwargs)
```

Extra Trees Regressor.

#### Parameters

- **n\_estimators** (*float*) – The number of trees in the forest. Defaults to 100.
- **max\_features** (*int, float or {"auto", "sqrt", "log2"}*) – The number of features to consider when looking for the best split:
  - If int, then consider max\_features features at each split.
  - If float, then max\_features is a fraction and int(max\_features \* n\_features) features are considered at each split.
  - If “auto”, then max\_features=sqrt(n\_features).
  - If “sqrt”, then max\_features=sqrt(n\_features).
  - If “log2”, then max\_features=log2(n\_features).
  - If None, then max\_features = n\_features.

The search for a split does not stop until at least one valid partition of the node samples is found, even if it requires to effectively inspect more than max\_features features. Defaults to “auto”.

- **max\_depth** (*int*) – The maximum depth of the tree. Defaults to 6.
- **min\_samples\_split** (*int or float*) – The minimum number of samples required to split an internal node:
  - If int, then consider min\_samples\_split as the minimum number.
  - If float, then min\_samples\_split is a fraction and ceil(min\_samples\_split \* n\_samples) are the minimum number of samples for each split.
- **to 2. (Defaults)** –
- **min\_weight\_fraction\_leaf** (*float*) – The minimum weighted fraction of the sum total of weights (of all the input samples) required to be at a leaf node. Defaults to 0.0.
- **n\_jobs** (*int or None*) – Number of jobs to run in parallel. -1 uses all processes. Defaults to -1.
- **random\_seed** (*int*) – Seed for the random number generator. Defaults to 0.

#### Attributes

<b>hyper-parameter_ranges</b>	{ “n_estimators”: Integer(10, 1000), “max_features”: [“auto”, “sqrt”, “log2”], “max_depth”: Integer(4, 10),}
<b>model_family</b>	ModelFamily.EXTRA_TREES
<b>modifies_features</b>	True
<b>modifies_target</b>	False
<b>name</b>	Extra Trees Regressor
<b>pre-dict_uses_y</b>	False
<b>supported_problem_types</b>	[ ProblemTypes.REGRESSION, ProblemTypes.TIME_SERIES_REGRESSION,]

## Methods

<i>clone</i>	Constructs a new component with the same parameters and random state.
<i>default_parameters</i>	Returns the default parameters for this component.
<i>describe</i>	Describe a component and its parameters
<i>feature_importance</i>	Returns importance associated with each feature.
<i>fit</i>	Fits component to data
<i>load</i>	Loads component at file path
<i>needs_fitting</i>	Returns boolean determining if component needs fitting before
<i>parameters</i>	Returns the parameters which were used to initialize the component
<i>predict</i>	Make predictions using selected features.
<i>predict_proba</i>	Make probability estimates for labels.
<i>save</i>	Saves component at file path

### **clone** (*self*)

Constructs a new component with the same parameters and random state.

**Returns** A new instance of this component with identical parameters and random state.

### **default\_parameters** (*cls*)

Returns the default parameters for this component.

Our convention is that `Component.default_parameters == Component().parameters`.

**Returns** default parameters for this component.

**Return type** dict

### **describe** (*self*, *print\_name=False*, *return\_dict=False*)

Describe a component and its parameters

#### Parameters

- **print\_name** (*bool*, *optional*) – whether to print name of component
- **return\_dict** (*bool*, *optional*) – whether to return description as dictionary in the format {“name”: name, “parameters”: parameters}

**Returns** prints and returns dictionary

**Return type** None or dict

**property feature\_importance** (*self*)

Returns importance associated with each feature.

**Returns** Importance associated with each feature

**Return type** np.ndarray

**fit** (*self*, *X*, *y=None*)

Fits component to data

**Parameters**

- **X** (*list*, *pd.DataFrame* or *np.ndarray*) – The input training data of shape [n\_samples, n\_features]
- **y** (*list*, *pd.Series*, *np.ndarray*, *optional*) – The target training data of length [n\_samples]

**Returns** self

**static load** (*file\_path*)

Loads component at file path

**Parameters** **file\_path** (*str*) – Location to load file

**Returns** ComponentBase object

**needs\_fitting** (*self*)

Returns boolean determining if component needs fitting before calling predict, predict\_proba, transform, or feature\_importances. This can be overridden to False for components that do not need to be fit or whose fit methods do nothing.

**property parameters** (*self*)

Returns the parameters which were used to initialize the component

**predict** (*self*, *X*)

Make predictions using selected features.

**Parameters** **X** (*pd.DataFrame*, *np.ndarray*) – Data of shape [n\_samples, n\_features]

**Returns** Predicted values

**Return type** pd.Series

**predict\_proba** (*self*, *X*)

Make probability estimates for labels.

**Parameters** **X** (*pd.DataFrame*, or *np.ndarray*) – Features

**Returns** Probability estimates

**Return type** pd.Series

**save** (*self*, *file\_path*, *pickle\_protocol=cloudpickle.DEFAULT\_PROTOCOL*)

Saves component at file path

**Parameters**

- **file\_path** (*str*) – Location to save file
- **pickle\_protocol** (*int*) – The pickle data stream format.

**Returns** None

**class** evalml.pipelines.**FeatureSelector** (*parameters=None, component\_obj=None, random\_seed=0, \*\*kwargs*)

Selects top features based on importance weights.

#### Parameters

- **parameters** (*dict*) – Dictionary of parameters for the component. Defaults to None.
- **component\_obj** (*obj*) – Third-party objects useful in component implementation. Defaults to None.
- **random\_seed** (*int*) – Seed for the random number generator. Defaults to 0.

#### Attributes

<b>model_family</b>	ModelFamily.NONE
<b>modifies_features</b>	True
<b>modifies_target</b>	False

#### Methods

<i>clone</i>	Constructs a new component with the same parameters and random state.
<i>default_parameters</i>	Returns the default parameters for this component.
<i>describe</i>	Describe a component and its parameters
<i>fit</i>	Fits component to data
<i>fit_transform</i>	Fits on X and transforms X
<i>get_names</i>	Get names of selected features.
<i>load</i>	Loads component at file path
<i>name</i>	Returns string name of this component
<i>needs_fitting</i>	Returns boolean determining if component needs fitting before
<i>parameters</i>	Returns the parameters which were used to initialize the component
<i>save</i>	Saves component at file path
<i>transform</i>	Transforms input data by selecting features. If the component_obj does not have a transform method, will raise an MethodPropertyNotFoundError exception.

**clone** (*self*)

Constructs a new component with the same parameters and random state.

**Returns** A new instance of this component with identical parameters and random state.

**default\_parameters** (*cls*)

Returns the default parameters for this component.

Our convention is that Component.default\_parameters == Component().parameters.

**Returns** default parameters for this component.

**Return type** dict

**describe** (*self, print\_name=False, return\_dict=False*)

Describe a component and its parameters

**Parameters**

- **print\_name** (*bool, optional*) – whether to print name of component
- **return\_dict** (*bool, optional*) – whether to return description as dictionary in the format {"name": name, "parameters": parameters}

**Returns** prints and returns dictionary

**Return type** None or dict

**fit** (*self, X, y=None*)

Fits component to data

**Parameters**

- **X** (*list, pd.DataFrame or np.ndarray*) – The input training data of shape [n\_samples, n\_features]
- **y** (*list, pd.Series, np.ndarray, optional*) – The target training data of length [n\_samples]

**Returns** self

**fit\_transform** (*self, X, y=None*)

Fits on X and transforms X

**Parameters**

- **X** (*pd.DataFrame*) – Data to fit and transform
- **y** (*pd.Series*) – Target data

**Returns** Transformed X

**Return type** pd.DataFrame

**get\_names** (*self*)

Get names of selected features.

**Returns** List of the names of features selected

**Return type** list[str]

**static load** (*file\_path*)

Loads component at file path

**Parameters** **file\_path** (*str*) – Location to load file

**Returns** ComponentBase object

**property name** (*cls*)

Returns string name of this component

**needs\_fitting** (*self*)

Returns boolean determining if component needs fitting before calling predict, predict\_proba, transform, or feature\_importances. This can be overridden to False for components that do not need to be fit or whose fit methods do nothing.

**property parameters** (*self*)

Returns the parameters which were used to initialize the component

**save** (*self, file\_path, pickle\_protocol=cloudpickle.DEFAULT\_PROTOCOL*)

Saves component at file path

**Parameters**



- **file\_path** (*str*) – Location to save file
- **pickle\_protocol** (*int*) – The pickle data stream format.

**Returns** None

**transform** (*self*, *X*, *y=None*)

Transforms input data by selecting features. If the component\_obj does not have a transform method, will raise an `MethodPropertyNotFoundError` exception.

**Parameters**

- **X** (*pd.DataFrame*) – Data to transform.
- **y** (*pd.Series*, *optional*) – Target data. Ignored.

**Returns** Transformed X

**Return type** `pd.DataFrame`

```
class evalml.pipelines.KNeighborsClassifier(n_neighbors=5, weights='uniform', algorithm='auto', leaf_size=30, p=2, random_seed=0, **kwargs)
```

K-Nearest Neighbors Classifier.

**Parameters**

- **n\_neighbors** (*int*) – Number of neighbors to use by default. Defaults to 5.
- **weights** (*{ 'uniform', 'distance' } or callable*) – Weight function used in prediction. Can be:
  - ‘uniform’ : uniform weights. All points in each neighborhood are weighted equally.
  - ‘distance’ : weight points by the inverse of their distance. in this case, closer neighbors of a query point will have a greater influence than neighbors which are further away.
  - [callable] : a user-defined function which accepts an array of distances, and returns an array of the same shape containing the weights.

Defaults to “uniform”.

- **algorithm** (*{ 'auto', 'ball\_tree', 'kd\_tree', 'brute' }*) – Algorithm used to compute the nearest neighbors:
  - ‘ball\_tree’ will use BallTree
  - ‘kd\_tree’ will use KDTree
  - ‘brute’ will use a brute-force search.

‘auto’ will attempt to decide the most appropriate algorithm based on the values passed to fit method. Defaults to “auto”. Note: fitting on sparse input will override the setting of this parameter, using brute force.
- **leaf\_size** (*int*) – Leaf size passed to BallTree or KDTree. This can affect the speed of the construction and query, as well as the memory required to store the tree. The optimal value depends on the nature of the problem. Defaults to 30.
- **p** (*int*) – Power parameter for the Minkowski metric. When  $p = 1$ , this is equivalent to using `manhattan_distance` (l1), and `euclidean_distance` (l2) for  $p = 2$ . For arbitrary  $p$ , `minkowski_distance(l_p)` is used. Defaults to 2.
- **random\_seed** (*int*) – Seed for the random number generator. Defaults to 0.

## Attributes

<b>hyper-parameter_ranges</b>	{ "n_neighbors": Integer(2, 12), "weights": ["uniform", "distance"], "algorithm": ["auto", "ball_tree", "kd_tree", "brute"], "leaf_size": Integer(10, 30), "p": Integer(1, 5), }
<b>model_family</b>	ModelFamily.K_NEIGHBORS
<b>modifies_features</b>	True
<b>modifies_target</b>	False
<b>name</b>	KNN Classifier
<b>pre_dict_uses_y</b>	False
<b>supported_problem_types</b>	[ ProblemTypes.BINARY, ProblemTypes.MULTICLASS, ProblemTypes.TIME_SERIES_BINARY, ProblemTypes.TIME_SERIES_MULTICLASS, ]

## Methods

<i>clone</i>	Constructs a new component with the same parameters and random state.
<i>default_parameters</i>	Returns the default parameters for this component.
<i>describe</i>	Describe a component and its parameters
<i>feature_importance</i>	Returns array of 0's matching the input number of features as feature_importance is
<i>fit</i>	Fits component to data
<i>load</i>	Loads component at file path
<i>needs_fitting</i>	Returns boolean determining if component needs fitting before
<i>parameters</i>	Returns the parameters which were used to initialize the component
<i>predict</i>	Make predictions using selected features.
<i>predict_proba</i>	Make probability estimates for labels.
<i>save</i>	Saves component at file path

**clone** (*self*)

Constructs a new component with the same parameters and random state.

**Returns** A new instance of this component with identical parameters and random state.

**default\_parameters** (*cls*)

Returns the default parameters for this component.

Our convention is that `Component.default_parameters == Component().parameters`.

**Returns** default parameters for this component.

**Return type** dict

**describe** (*self*, *print\_name=False*, *return\_dict=False*)

Describe a component and its parameters

**Parameters**

- **print\_name** (*bool*, *optional*) – whether to print name of component
- **return\_dict** (*bool*, *optional*) – whether to return description as dictionary in the format {"name": name, "parameters": parameters}

**Returns** prints and returns dictionary

**Return type** None or dict

**property feature\_importance** (*self*)

Returns array of 0's matching the input number of features as feature\_importance is not defined for KNN classifiers.

**fit** (*self*, *X*, *y=None*)

Fits component to data

**Parameters**

- **X** (*list*, *pd.DataFrame* or *np.ndarray*) – The input training data of shape [n\_samples, n\_features]
- **y** (*list*, *pd.Series*, *np.ndarray*, *optional*) – The target training data of length [n\_samples]

**Returns** self

**static load** (*file\_path*)

Loads component at file path

**Parameters** **file\_path** (*str*) – Location to load file

**Returns** ComponentBase object

**needs\_fitting** (*self*)

Returns boolean determining if component needs fitting before calling predict, predict\_proba, transform, or feature\_importances. This can be overridden to False for components that do not need to be fit or whose fit methods do nothing.

**property parameters** (*self*)

Returns the parameters which were used to initialize the component

**predict** (*self*, *X*)

Make predictions using selected features.

**Parameters** **X** (*pd.DataFrame*, *np.ndarray*) – Data of shape [n\_samples, n\_features]

**Returns** Predicted values

**Return type** pd.Series

**predict\_proba** (*self*, *X*)

Make probability estimates for labels.

**Parameters** **X** (*pd.DataFrame*, or *np.ndarray*) – Features

**Returns** Probability estimates

**Return type** pd.Series

**save** (*self*, *file\_path*, *pickle\_protocol=cloudpickle.DEFAULT\_PROTOCOL*)

Saves component at file path

**Parameters**

- **file\_path** (*str*) – Location to save file
- **pickle\_protocol** (*int*) – The pickle data stream format.

**Returns** None

```
class evalml.pipelines.LightGBMClassifier(boosting_type='gbdt',      learning_rate=0.1,
                                         n_estimators=100,         max_depth=0,
                                         num_leaves=31, min_child_samples=20, bag-
                                         ging_fraction=0.9, bagging_freq=0, n_jobs=-
                                         1, random_seed=0, **kwargs)
```

LightGBM Classifier.

#### Parameters

- **boosting\_type** (*string*) – Type of boosting to use. Defaults to “gbdt”. - ‘gbdt’ uses traditional Gradient Boosting Decision Tree - “dart”, uses Dropouts meet Multiple Additive Regression Trees - “goss”, uses Gradient-based One-Side Sampling - “rf”, uses Random Forest
- **learning\_rate** (*float*) – Boosting learning rate. Defaults to 0.1.
- **n\_estimators** (*int*) – Number of boosted trees to fit. Defaults to 100.
- **max\_depth** (*int*) – Maximum tree depth for base learners, <=0 means no limit. Defaults to 0.
- **num\_leaves** (*int*) – Maximum tree leaves for base learners. Defaults to 31.
- **min\_child\_samples** (*int*) – Minimum number of data needed in a child (leaf). Defaults to 20.
- **bagging\_fraction** (*float*) – LightGBM will randomly select a subset of features on each iteration (tree) without resampling if this is smaller than 1.0. For example, if set to 0.8, LightGBM will select 80% of features before training each tree. This can be used to speed up training and deal with overfitting. Defaults to 0.9.
- **bagging\_freq** (*int*) – Frequency for bagging. 0 means bagging is disabled. k means perform bagging at every k iteration. Every k-th iteration, LightGBM will randomly select bagging\_fraction \* 100 % of the data to use for the next k iterations. Defaults to 0.
- **n\_jobs** (*int or None*) – Number of threads to run in parallel. -1 uses all threads. Defaults to -1.
- **random\_seed** (*int*) – Seed for the random number generator. Defaults to 0.

#### Attributes

<b>hyper-parameter_ranges</b>	{ “learning_rate”: Real(0.000001, 1), “boosting_type”: [“gbdt”, “dart”, “goss”, “rf”], “n_estimators”: Integer(10, 100), “max_depth”: Integer(0, 10), “num_leaves”: Integer(2, 100), “min_child_samples”: Integer(1, 100), “bagging_fraction”: Real(0.000001, 1), “bagging_freq”: Integer(0, 1),}
<b>model_family</b>	ModelFamily.LIGHTGBM
<b>modifies_features</b>	True
<b>modifies_target</b>	False
<b>name</b>	LightGBM Classifier
<b>pre-dict_uses_y</b>	False
<b>SEED_MAX</b>	SEED_BOUNDS.max_bound
<b>SEED_MIN</b>	0
<b>supported_problem_types</b>	[ ProblemTypes.BINARY, ProblemTypes.MULTICLASS, ProblemTypes.TIME_SERIES_BINARY, ProblemTypes.TIME_SERIES_MULTICLASS,]

## Methods

<code>clone</code>	Constructs a new component with the same parameters and random state.
<code>default_parameters</code>	Returns the default parameters for this component.
<code>describe</code>	Describe a component and its parameters
<code>feature_importance</code>	Returns importance associated with each feature.
<code>fit</code>	Fits component to data
<code>load</code>	Loads component at file path
<code>needs_fitting</code>	Returns boolean determining if component needs fitting before
<code>parameters</code>	Returns the parameters which were used to initialize the component
<code>predict</code>	Make predictions using selected features.
<code>predict_proba</code>	Make probability estimates for labels.
<code>save</code>	Saves component at file path

### **clone** (*self*)

Constructs a new component with the same parameters and random state.

**Returns** A new instance of this component with identical parameters and random state.

### **default\_parameters** (*cls*)

Returns the default parameters for this component.

Our convention is that `Component.default_parameters == Component().parameters`.

**Returns** default parameters for this component.

**Return type** dict

### **describe** (*self*, *print\_name=False*, *return\_dict=False*)

Describe a component and its parameters

#### Parameters

- **print\_name** (*bool*, *optional*) – whether to print name of component
- **return\_dict** (*bool*, *optional*) – whether to return description as dictionary in the format {"name": name, "parameters": parameters}

**Returns** prints and returns dictionary

**Return type** None or dict

### **property feature\_importance** (*self*)

Returns importance associated with each feature.

**Returns** Importance associated with each feature

**Return type** np.ndarray

### **fit** (*self*, *X*, *y=None*)

Fits component to data

#### Parameters

- **X** (*list*, *pd.DataFrame* or *np.ndarray*) – The input training data of shape [n\_samples, n\_features]
- **y** (*list*, *pd.Series*, *np.ndarray*, *optional*) – The target training data of length [n\_samples]

**Returns** self

**static load** (*file\_path*)

Loads component at file path

**Parameters** *file\_path* (*str*) – Location to load file

**Returns** ComponentBase object

**needs\_fitting** (*self*)

Returns boolean determining if component needs fitting before calling predict, predict\_proba, transform, or feature\_importances. This can be overridden to False for components that do not need to be fit or whose fit methods do nothing.

**property parameters** (*self*)

Returns the parameters which were used to initialize the component

**predict** (*self*, *X*)

Make predictions using selected features.

**Parameters** *X* (*pd.DataFrame*, *np.ndarray*) – Data of shape [n\_samples, n\_features]

**Returns** Predicted values

**Return type** *pd.Series*

**predict\_proba** (*self*, *X*)

Make probability estimates for labels.

**Parameters** *X* (*pd.DataFrame*, or *np.ndarray*) – Features

**Returns** Probability estimates

**Return type** *pd.Series*

**save** (*self*, *file\_path*, *pickle\_protocol=cloudpickle.DEFAULT\_PROTOCOL*)

Saves component at file path

**Parameters**

- **file\_path** (*str*) – Location to save file
- **pickle\_protocol** (*int*) – The pickle data stream format.

**Returns** None

```
class evalml.pipelines.LightGBMRegressor (boosting_type='gbdt', learning_rate=0.1,  
                                         n_estimators=20, max_depth=0,  
                                         num_leaves=31, min_child_samples=20, bag-  
                                         ging_fraction=0.9, bagging_freq=0, n_jobs=- 1,  
                                         random_seed=0, **kwargs)
```

LightGBM Regressor.

**Parameters**

- **boosting\_type** (*string*) – Type of boosting to use. Defaults to “gbdt”. - ‘gbdt’ uses traditional Gradient Boosting Decision Tree - “dart”, uses Dropouts meet Multiple Additive Regression Trees - “goss”, uses Gradient-based One-Side Sampling - “rf”, uses Random Forest
- **learning\_rate** (*float*) – Boosting learning rate. Defaults to 0.1.
- **n\_estimators** (*int*) – Number of boosted trees to fit. Defaults to 100.
- **max\_depth** (*int*) – Maximum tree depth for base learners, <=0 means no limit. Defaults to 0.

- **num\_leaves** (*int*) – Maximum tree leaves for base learners. Defaults to 31.
- **min\_child\_samples** (*int*) – Minimum number of data needed in a child (leaf). Defaults to 20.
- **bagging\_fraction** (*float*) – LightGBM will randomly select a subset of features on each iteration (tree) without resampling if this is smaller than 1.0. For example, if set to 0.8, LightGBM will select 80% of features before training each tree. This can be used to speed up training and deal with overfitting. Defaults to 0.9.
- **bagging\_freq** (*int*) – Frequency for bagging. 0 means bagging is disabled. k means perform bagging at every k iteration. Every k-th iteration, LightGBM will randomly select bagging\_fraction \* 100 % of the data to use for the next k iterations. Defaults to 0.
- **n\_jobs** (*int or None*) – Number of threads to run in parallel. -1 uses all threads. Defaults to -1.
- **random\_seed** (*int*) – Seed for the random number generator. Defaults to 0.

#### Attributes

<b>hyper-parameter_ranges</b>	{ "learning_rate": Real(0.000001, 1), "boosting_type": ["gbdt", "dart", "goss", "rf"], "n_estimators": Integer(10, 100), "max_depth": Integer(0, 10), "num_leaves": Integer(2, 100), "min_child_samples": Integer(1, 100), "bagging_fraction": Real(0.000001, 1), "bagging_freq": Integer(0, 1), }
<b>model_family</b>	ModelFamily.LIGHTGBM
<b>modifies_features</b>	True
<b>modifies_target</b>	False
<b>name</b>	LightGBM Regressor
<b>predict_uses_y</b>	False
<b>SEED_MAX</b>	SEED_BOUNDS.max_bound
<b>SEED_MIN</b>	0
<b>supported_problem_types</b>	[ProblemTypes.REGRESSION]

#### Methods

<i>clone</i>	Constructs a new component with the same parameters and random state.
<i>default_parameters</i>	Returns the default parameters for this component.
<i>describe</i>	Describe a component and its parameters
<i>feature_importance</i>	Returns importance associated with each feature.
<i>fit</i>	Fits component to data
<i>load</i>	Loads component at file path
<i>needs_fitting</i>	Returns boolean determining if component needs fitting before
<i>parameters</i>	Returns the parameters which were used to initialize the component
<i>predict</i>	Make predictions using selected features.
<i>predict_proba</i>	Make probability estimates for labels.
<i>save</i>	Saves component at file path

**clone** (*self*)

Constructs a new component with the same parameters and random state.

**Returns** A new instance of this component with identical parameters and random state.

**default\_parameters** (*cls*)

Returns the default parameters for this component.

Our convention is that `Component.default_parameters == Component().parameters`.

**Returns** default parameters for this component.

**Return type** dict

**describe** (*self*, *print\_name=False*, *return\_dict=False*)

Describe a component and its parameters

**Parameters**

- **print\_name** (*bool*, *optional*) – whether to print name of component
- **return\_dict** (*bool*, *optional*) – whether to return description as dictionary in the format {"name": name, "parameters": parameters}

**Returns** prints and returns dictionary

**Return type** None or dict

**property feature\_importance** (*self*)

Returns importance associated with each feature.

**Returns** Importance associated with each feature

**Return type** np.ndarray

**fit** (*self*, *X*, *y=None*)

Fits component to data

**Parameters**

- **X** (*list*, *pd.DataFrame* or *np.ndarray*) – The input training data of shape [n\_samples, n\_features]
- **y** (*list*, *pd.Series*, *np.ndarray*, *optional*) – The target training data of length [n\_samples]

**Returns** self

**static load** (*file\_path*)

Loads component at file path

**Parameters** **file\_path** (*str*) – Location to load file

**Returns** ComponentBase object

**needs\_fitting** (*self*)

Returns boolean determining if component needs fitting before calling predict, predict\_proba, transform, or feature\_importances. This can be overridden to False for components that do not need to be fit or whose fit methods do nothing.

**property parameters** (*self*)

Returns the parameters which were used to initialize the component

**predict** (*self*, *X*)

Make predictions using selected features.

**Parameters** **X** (*pd.DataFrame*, *np.ndarray*) – Data of shape [n\_samples, n\_features]



**Returns** Predicted values

**Return type** `pd.Series`

**predict\_proba** (*self*, *X*)

Make probability estimates for labels.

**Parameters** *X* (`pd.DataFrame`, or `np.ndarray`) – Features

**Returns** Probability estimates

**Return type** `pd.Series`

**save** (*self*, *file\_path*, *pickle\_protocol=cloudpickle.DEFAULT\_PROTOCOL*)

Saves component at file path

**Parameters**

- **file\_path** (*str*) – Location to save file
- **pickle\_protocol** (*int*) – The pickle data stream format.

**Returns** None

**class** `evalml.pipelines.LinearRegressor` (*fit\_intercept=True*, *normalize=False*, *n\_jobs=-1*, *random\_seed=0*, *\*\*kwargs*)

Linear Regressor.

**Parameters**

- **fit\_intercept** (*boolean*) – Whether to calculate the intercept for this model. If set to `False`, no intercept will be used in calculations (i.e. data is expected to be centered). Defaults to `True`.
- **normalize** (*boolean*) – If `True`, the regressors will be normalized before regression by subtracting the mean and dividing by the l2-norm. This parameter is ignored when `fit_intercept` is set to `False`. Defaults to `False`.
- **n\_jobs** (*int* or *None*) – Number of jobs to run in parallel. `-1` uses all threads. Defaults to `-1`.
- **random\_seed** (*int*) – Seed for the random number generator. Defaults to `0`.

**Attributes**

<b>hyper-parameter_ranges</b>	{ “fit_intercept”: [True, False], “normalize”: [True, False]}
<b>model_family</b>	ModelFamily.LINEAR_MODEL
<b>modifies_features</b>	True
<b>modifies_target</b>	False
<b>name</b>	Linear Regressor
<b>predict_uses_y</b>	False
<b>supported_problem_types</b>	[ ProblemTypes.REGRESSION, ProblemTypes.TIME_SERIES_REGRESSION,]

**Methods**

<code>clone</code>	Constructs a new component with the same parameters and random state.
<code>default_parameters</code>	Returns the default parameters for this component.
<code>describe</code>	Describe a component and its parameters
<code>feature_importance</code>	Returns importance associated with each feature.
<code>fit</code>	Fits component to data
<code>load</code>	Loads component at file path
<code>needs_fitting</code>	Returns boolean determining if component needs fitting before
<code>parameters</code>	Returns the parameters which were used to initialize the component
<code>predict</code>	Make predictions using selected features.
<code>predict_proba</code>	Make probability estimates for labels.
<code>save</code>	Saves component at file path

**clone** (*self*)

Constructs a new component with the same parameters and random state.

**Returns** A new instance of this component with identical parameters and random state.

**default\_parameters** (*cls*)

Returns the default parameters for this component.

Our convention is that `Component.default_parameters == Component().parameters`.

**Returns** default parameters for this component.

**Return type** dict

**describe** (*self*, *print\_name=False*, *return\_dict=False*)

Describe a component and its parameters

**Parameters**

- **print\_name** (*bool*, *optional*) – whether to print name of component
- **return\_dict** (*bool*, *optional*) – whether to return description as dictionary in the format {"name": name, "parameters": parameters}

**Returns** prints and returns dictionary

**Return type** None or dict

**property feature\_importance** (*self*)

Returns importance associated with each feature.

**Returns** Importance associated with each feature

**Return type** np.ndarray

**fit** (*self*, *X*, *y=None*)

Fits component to data

**Parameters**

- **X** (*list*, *pd.DataFrame* or *np.ndarray*) – The input training data of shape [n\_samples, n\_features]
- **y** (*list*, *pd.Series*, *np.ndarray*, *optional*) – The target training data of length [n\_samples]

**Returns** self

**static load** (*file\_path*)

Loads component at file path

**Parameters** *file\_path* (*str*) – Location to load file

**Returns** ComponentBase object

**needs\_fitting** (*self*)

Returns boolean determining if component needs fitting before calling predict, predict\_proba, transform, or feature\_importances. This can be overridden to False for components that do not need to be fit or whose fit methods do nothing.

**property parameters** (*self*)

Returns the parameters which were used to initialize the component

**predict** (*self*, *X*)

Make predictions using selected features.

**Parameters** *X* (*pd.DataFrame*, *np.ndarray*) – Data of shape [n\_samples, n\_features]

**Returns** Predicted values

**Return type** *pd.Series*

**predict\_proba** (*self*, *X*)

Make probability estimates for labels.

**Parameters** *X* (*pd.DataFrame*, *or np.ndarray*) – Features

**Returns** Probability estimates

**Return type** *pd.Series*

**save** (*self*, *file\_path*, *pickle\_protocol=cloudpickle.DEFAULT\_PROTOCOL*)

Saves component at file path

**Parameters**

- **file\_path** (*str*) – Location to save file
- **pickle\_protocol** (*int*) – The pickle data stream format.

**Returns** None

```
class evalml.pipelines.LogisticRegressionClassifier (penalty='l2', C=1.0,
                                                    multi_class='auto',
                                                    solver='lbfgs', n_jobs=-1,
                                                    random_seed=0, **kwargs)
```

Logistic Regression Classifier.

**Parameters**

- **penalty** (*{ "l1", "l2", "elasticnet", "none" }*) – The norm used in penalization. Defaults to “l2”.
- **C** (*float*) – Inverse of regularization strength. Must be a positive float. Defaults to 1.0.
- **multi\_class** (*{ "auto", "ovr", "multinomial" }*) – If the option chosen is “ovr”, then a binary problem is fit for each label. For “multinomial” the loss minimised is the multinomial loss fit across the entire probability distribution, even when the data is binary. “multinomial” is unavailable when solver=“liblinear”. “auto” selects “ovr” if the data is binary, or if solver=“liblinear”, and otherwise selects “multinomial”. Defaults to “auto”.
- **solver** (*{ "newton-cg", "lbfgs", "liblinear", "sag", "saga" }*) – Algorithm to use in the optimization problem. For small datasets, “liblinear” is a good choice, whereas “sag” and “saga” are faster for large ones. For multiclass problems, only

“newton-cg”, “sag”, “saga” and “lbfgs” handle multinomial loss; “liblinear” is limited to one-versus-rest schemes.

- “newton-cg”, “lbfgs”, “sag” and “saga” handle L2 or no penalty
- “liblinear” and “saga” also handle L1 penalty
- “saga” also supports “elasticnet” penalty
- “liblinear” does not support setting `penalty='none'`

Defaults to “lbfgs”.

- **n\_jobs** (*int*) – Number of parallel threads used to run xgboost. Note that creating thread contention will significantly slow down the algorithm. Defaults to -1.
- **random\_seed** (*int*) – Seed for the random number generator. Defaults to 0.

#### Attributes

<b>hyper-parameter_ranges</b>	{ “penalty”: [“l2”], “C”: Real(0.01, 10), }
<b>model_family</b>	ModelFamily.LINEAR_MODEL
<b>modifies_features</b>	True
<b>modifies_target</b>	False
<b>name</b>	Logistic Regression Classifier
<b>predict_uses_y</b>	False
<b>supported_problem_types</b>	[ ProblemTypes.BINARY, ProblemTypes.MULTICLASS, ProblemTypes.TIME_SERIES_BINARY, ProblemTypes.TIME_SERIES_MULTICLASS, ]

#### Methods

<i>clone</i>	Constructs a new component with the same parameters and random state.
<i>default_parameters</i>	Returns the default parameters for this component.
<i>describe</i>	Describe a component and its parameters
<i>feature_importance</i>	Returns importance associated with each feature.
<i>fit</i>	Fits component to data
<i>load</i>	Loads component at file path
<i>needs_fitting</i>	Returns boolean determining if component needs fitting before
<i>parameters</i>	Returns the parameters which were used to initialize the component
<i>predict</i>	Make predictions using selected features.
<i>predict_proba</i>	Make probability estimates for labels.
<i>save</i>	Saves component at file path

#### **clone** (*self*)

Constructs a new component with the same parameters and random state.

**Returns** A new instance of this component with identical parameters and random state.

#### **default\_parameters** (*cls*)

Returns the default parameters for this component.

Our convention is that `Component.default_parameters == Component().parameters`.

**Returns** default parameters for this component.

**Return type** dict

**describe** (*self*, *print\_name=False*, *return\_dict=False*)

Describe a component and its parameters

**Parameters**

- **print\_name** (*bool*, *optional*) – whether to print name of component
- **return\_dict** (*bool*, *optional*) – whether to return description as dictionary in the format {"name": name, "parameters": parameters}

**Returns** prints and returns dictionary

**Return type** None or dict

**property feature\_importance** (*self*)

Returns importance associated with each feature.

**Returns** Importance associated with each feature

**Return type** np.ndarray

**fit** (*self*, *X*, *y=None*)

Fits component to data

**Parameters**

- **X** (*list*, *pd.DataFrame* or *np.ndarray*) – The input training data of shape [n\_samples, n\_features]
- **y** (*list*, *pd.Series*, *np.ndarray*, *optional*) – The target training data of length [n\_samples]

**Returns** self

**static load** (*file\_path*)

Loads component at file path

**Parameters** **file\_path** (*str*) – Location to load file

**Returns** ComponentBase object

**needs\_fitting** (*self*)

Returns boolean determining if component needs fitting before calling predict, predict\_proba, transform, or feature\_importances. This can be overridden to False for components that do not need to be fit or whose fit methods do nothing.

**property parameters** (*self*)

Returns the parameters which were used to initialize the component

**predict** (*self*, *X*)

Make predictions using selected features.

**Parameters** **X** (*pd.DataFrame*, *np.ndarray*) – Data of shape [n\_samples, n\_features]

**Returns** Predicted values

**Return type** pd.Series

**predict\_proba** (*self*, *X*)

Make probability estimates for labels.

**Parameters** *X* (*pd.DataFrame*, or *np.ndarray*) – Features

**Returns** Probability estimates

**Return type** *pd.Series*

**save** (*self*, *file\_path*, *pickle\_protocol=cloudpickle.DEFAULT\_PROTOCOL*)

Saves component at file path

**Parameters**

- **file\_path** (*str*) – Location to save file
- **pickle\_protocol** (*int*) – The pickle data stream format.

**Returns** None

**class** evalml.pipelines.MulticlassClassificationPipeline (*component\_graph*, *parameters=None*, *custom\_name=None*, *random\_seed=0*)

Pipeline subclass for all multiclass classification pipelines.

**Parameters**

- **component\_graph** (*list* or *dict*) – List of components in order. Accepts strings or ComponentBase subclasses in the list. Note that when duplicate components are specified in a list, the duplicate component names will be modified with the component’s index in the list. For example, the component graph [Imputer, One Hot Encoder, Imputer, Logistic Regression Classifier] will have names [“Imputer”, “One Hot Encoder”, “Imputer\_2”, “Logistic Regression Classifier”]
- **parameters** (*dict*) – Dictionary with component names as keys and dictionary of that component’s parameters as values. An empty dictionary or None implies using all default values for component parameters. Defaults to None.
- **custom\_name** (*str*) – Custom name for the pipeline. Defaults to None.
- **random\_seed** (*int*) – Seed for the random number generator. Defaults to 0.

**Attributes**

<b>problem_type</b>	ProblemTypes.MULTICLASS
---------------------	-------------------------

**Methods**

<i>can_tune_threshold_with_objective</i>	Determine whether the threshold of a binary classification pipeline can be tuned.
<i>classes_</i>	Gets the class names for the problem.
<i>clone</i>	Constructs a new pipeline with the same components, parameters, and random state.
<i>compute_estimator_features</i>	Transforms the data by applying all pre-processing components.
<i>create_objectives</i>	
<i>custom_name</i>	Custom name of the pipeline.

continues on next page

Table 649 – continued from previous page

<i>describe</i>	Outputs pipeline details including component parameters
<i>feature_importance</i>	Importance associated with each feature. Features dropped by the feature selection are excluded.
<i>fit</i>	Build a classification model. For string and categorical targets, classes are sorted
<i>get_component</i>	Returns component by name
<i>get_hyperparameter_ranges</i>	Returns hyperparameter ranges from all components as a dictionary.
<i>graph</i>	Generate an image representing the pipeline graph.
<i>graph_feature_importance</i>	Generate a bar graph of the pipeline's feature importance
<i>inverse_transform</i>	Apply component <code>inverse_transform</code> methods to estimator predictions in reverse order.
<i>load</i>	Loads pipeline at file path
<i>model_family</i>	Returns model family of this pipeline.
<i>name</i>	Name of the pipeline.
<i>new</i>	Constructs a new instance of the pipeline with the same component graph but with a different set of parameters.
<i>parameters</i>	Parameter dictionary for this pipeline.
<i>predict</i>	Make predictions using selected features.
<i>predict_proba</i>	Make probability estimates for labels.
<i>save</i>	Saves pipeline at file path
<i>score</i>	Evaluate model performance on objectives
<i>summary</i>	A short summary of the pipeline structure, describing the list of components used.

**can\_tune\_threshold\_with\_objective** (*self*, *objective*)

Determine whether the threshold of a binary classification pipeline can be tuned.

#### Parameters

- **pipeline** (`PipelineBase`) – Binary classification pipeline.
- **objective** – Primary AutoMLSearch objective.

**property classes\_** (*self*)

Gets the class names for the problem.

**clone** (*self*)

Constructs a new pipeline with the same components, parameters, and random state.

**Returns** A new instance of this pipeline with identical components, parameters, and random state.

**compute\_estimator\_features** (*self*, *X*, *y=None*)

Transforms the data by applying all pre-processing components.

**Parameters** *X* (`pd.DataFrame`) – Input data to the pipeline to transform.

**Returns** New transformed features.

**Return type** `pd.DataFrame`

**static create\_objectives** (*objectives*)

**property** `custom_name` (*self*)

Custom name of the pipeline.

**describe** (*self*, *return\_dict=False*)

Outputs pipeline details including component parameters

**Parameters** `return_dict` (*bool*) – If True, return dictionary of information about pipeline.  
Defaults to False.

**Returns** Dictionary of all component parameters if `return_dict` is True, else None

**Return type** dict

**property** `feature_importance` (*self*)

Importance associated with each feature. Features dropped by the feature selection are excluded.

**Returns** `pd.DataFrame` including feature names and their corresponding importance

**fit** (*self*, *X*, *y*)

**Build a classification model. For string and categorical targets, classes are sorted** by `sorted(set(y))` and then are mapped to values between 0 and `n_classes-1`.

**Parameters**

- **X** (*pd.DataFrame* or *np.ndarray*) – The input training data of shape `[n_samples, n_features]`
- **y** (*pd.Series*, *np.ndarray*) – The target training labels of length `[n_samples]`

**Returns** *self*

**get\_component** (*self*, *name*)

Returns component by name

**Parameters** `name` (*str*) – Name of component

**Returns** Component to return

**Return type** Component

**get\_hyperparameter\_ranges** (*self*, *custom\_hyperparameters*)

Returns hyperparameter ranges from all components as a dictionary.

**Parameters** `custom_hyperparameters` (*dict*) – Custom hyperparameters for the pipeline.

**Returns** Dictionary of hyperparameter ranges for each component in the pipeline.

**Return type** dict

**graph** (*self*, *filepath=None*)

Generate an image representing the pipeline graph.

**Parameters** `filepath` (*str*, *optional*) – Path to where the graph should be saved. If set to None (as by default), the graph will not be saved.

**Returns** Graph object that can be directly displayed in Jupyter notebooks.

**Return type** `graphviz.Digraph`

**graph\_feature\_importance** (*self*, *importance\_threshold=0*)

Generate a bar graph of the pipeline's feature importance



**Parameters** `importance_threshold` (*float, optional*) – If provided, graph features with a permutation importance whose absolute value is larger than `importance_threshold`. Defaults to zero.

**Returns** `plotly.Figure`, a bar graph showing features and their corresponding importance

**inverse\_transform** (*self, y*)

Apply component `inverse_transform` methods to estimator predictions in reverse order.

Components that implement `inverse_transform` are `PolynomialDetrender`, `LabelEncoder` (tbd).

**Parameters** `y` (*pd.Series*) – Final component features

**static load** (*file\_path*)

Loads pipeline at file path

**Parameters** `file_path` (*str*) – location to load file

**Returns** `PipelineBase` object

**property model\_family** (*self*)

Returns model family of this pipeline.

**property name** (*self*)

Name of the pipeline.

**new** (*self, parameters, random\_seed=0*)

**Constructs a new instance of the pipeline with the same component graph but with a different set of parameters.**

Not to be confused with python's `__new__` method.

**Parameters**

- **parameters** (*dict*) – Dictionary with component names as keys and dictionary of that component's parameters as values. An empty dictionary or `None` implies using all default values for component parameters. Defaults to `None`.
- **random\_seed** (*int*) – Seed for the random number generator. Defaults to 0.

**Returns** A new instance of this pipeline with identical components.

**property parameters** (*self*)

Parameter dictionary for this pipeline.

**Returns** Dictionary of all component parameters.

**Return type** `dict`

**predict** (*self, X, objective=None*)

Make predictions using selected features.

**Parameters**

- **X** (*pd.DataFrame, or np.ndarray*) – Data of shape `[n_samples, n_features]`
- **objective** (*Object or string*) – The objective to use to make predictions

**Returns** Estimated labels

**Return type** `pd.Series`

**predict\_proba** (*self, X*)

Make probability estimates for labels.

**Parameters** `X` (*pd.DataFrame or np.ndarray*) – Data of shape `[n_samples, n_features]`

**Returns** Probability estimates

**Return type** `pd.DataFrame`

**save** (*self*, *file\_path*, *pickle\_protocol=cloudpickle.DEFAULT\_PROTOCOL*)

Saves pipeline at file path

**Parameters**

- **file\_path** (*str*) – location to save file
- **pickle\_protocol** (*int*) – the pickle data stream format.

**Returns** `None`

**score** (*self*, *X*, *y*, *objectives*)

Evaluate model performance on objectives

**Parameters**

- **X** (*pd.DataFrame* or *np.ndarray*) – Data of shape `[n_samples, n_features]`
- **y** (*pd.Series*, or *np.ndarray*) – True labels of length `[n_samples]`
- **objectives** (*list*) – List of objectives to score

**Returns** Ordered dictionary of objective scores

**Return type** `dict`

**property summary** (*self*)

A short summary of the pipeline structure, describing the list of components used. Example: Logistic Regression Classifier w/ Simple Imputer + One Hot Encoder

```
class evalml.pipelines.OneHotEncoder (top_n=10, features_to_encode=None, cat-  
                                     egories=None, drop='if_binary', han-  
                                     dle_unknown='ignore', handle_missing='error',  
                                     random_seed=0, **kwargs)
```

A transformer that encodes categorical features in a one-hot numeric array.

**Parameters**

- **top\_n** (*int*) – Number of categories per column to encode. If `None`, all categories will be encoded. Otherwise, the *n* most frequent will be encoded and all others will be dropped. Defaults to 10.
- **features\_to\_encode** (*list[str]*) – List of columns to encode. All other columns will remain untouched. If `None`, all appropriate columns will be encoded. Defaults to `None`.
- **categories** (*list*) – A two dimensional list of categories, where *categories[i]* is a list of the categories for the column at index *i*. This can also be `None`, or “auto” if *top\_n* is not `None`. Defaults to `None`.
- **drop** (*string*, *list*) – Method (“first” or “if\_binary”) to use to drop one category per feature. Can also be a list specifying which categories to drop for each feature. Defaults to “if\_binary”.
- **handle\_unknown** (*string*) – Whether to ignore or error for unknown categories for a feature encountered during *fit* or *transform*. If either *top\_n* or *categories* is used to limit the number of categories per column, this must be “ignore”. Defaults to “ignore”.
- **handle\_missing** (*string*) – Options for how to handle missing (NaN) values encountered during *fit* or *transform*. If this is set to “as\_category” and NaN values are within the *n* most frequent, “nan” values will be encoded as their own column. If this is set to “error”, any missing values encountered will raise an error. Defaults to “error”.

- **random\_seed** (*int*) – Seed for the random number generator. Defaults to 0.

#### Attributes

<b>hyper-parameter_ranges</b>	{}
<b>model_family</b>	ModelFamily.NONE
<b>modifies_features</b>	True
<b>modifies_target</b>	False
<b>name</b>	One Hot Encoder

#### Methods

<i>categories</i>	Returns a list of the unique categories to be encoded for the particular feature, in order.
<i>clone</i>	Constructs a new component with the same parameters and random state.
<i>default_parameters</i>	Returns the default parameters for this component.
<i>describe</i>	Describe a component and its parameters
<i>fit</i>	Fits component to data
<i>fit_transform</i>	Fits on X and transforms X
<i>get_feature_names</i>	Return feature names for the categorical features after fitting.
<i>load</i>	Loads component at file path
<i>needs_fitting</i>	Returns boolean determining if component needs fitting before
<i>parameters</i>	Returns the parameters which were used to initialize the component
<i>save</i>	Saves component at file path
<i>transform</i>	One-hot encode the input data.

#### **categories** (*self*, *feature\_name*)

Returns a list of the unique categories to be encoded for the particular feature, in order.

**Parameters** **feature\_name** (*str*) – the name of any feature provided to one-hot encoder during fit

**Returns** the unique categories, in the same dtype as they were provided during fit

**Return type** np.ndarray

#### **clone** (*self*)

Constructs a new component with the same parameters and random state.

**Returns** A new instance of this component with identical parameters and random state.

#### **default\_parameters** (*cls*)

Returns the default parameters for this component.

Our convention is that `Component.default_parameters == Component().parameters`.

**Returns** default parameters for this component.

**Return type** dict

**describe** (*self*, *print\_name=False*, *return\_dict=False*)

Describe a component and its parameters

**Parameters**

- **print\_name** (*bool*, *optional*) – whether to print name of component
- **return\_dict** (*bool*, *optional*) – whether to return description as dictionary in the format {"name": name, "parameters": parameters}

**Returns** prints and returns dictionary

**Return type** None or dict

**fit** (*self*, *X*, *y=None*)

Fits component to data

**Parameters**

- **X** (*list*, *pd.DataFrame* or *np.ndarray*) – The input training data of shape [n\_samples, n\_features]
- **y** (*list*, *pd.Series*, *np.ndarray*, *optional*) – The target training data of length [n\_samples]

**Returns** self

**fit\_transform** (*self*, *X*, *y=None*)

Fits on X and transforms X

**Parameters**

- **X** (*pd.DataFrame*) – Data to fit and transform
- **y** (*pd.Series*) – Target data

**Returns** Transformed X

**Return type** pd.DataFrame

**get\_feature\_names** (*self*)

Return feature names for the categorical features after fitting.

Feature names are formatted as {column name}\_{category name}. In the event of a duplicate name, an integer will be added at the end of the feature name to distinguish it.

For example, consider a dataframe with a column called “A” and category “x\_y” and another column called “A\_x” with “y”. In this example, the feature names would be “A\_x\_y” and “A\_x\_y\_1”.

**Returns** The feature names after encoding, provided in the same order as input\_features.

**Return type** np.ndarray

**static load** (*file\_path*)

Loads component at file path

**Parameters** **file\_path** (*str*) – Location to load file

**Returns** ComponentBase object

**needs\_fitting** (*self*)

Returns boolean determining if component needs fitting before calling predict, predict\_proba, transform, or feature\_importances. This can be overridden to False for components that do not need to be fit or whose fit methods do nothing.

**property parameters** (*self*)

Returns the parameters which were used to initialize the component

**save** (*self*, *file\_path*, *pickle\_protocol*=*cloudpickle.DEFAULT\_PROTOCOL*)

Saves component at file path

#### Parameters

- **file\_path** (*str*) – Location to save file
- **pickle\_protocol** (*int*) – The pickle data stream format.

**Returns** None

**transform** (*self*, *X*, *y=None*)

One-hot encode the input data.

#### Parameters

- **X** (*pd.DataFrame*) – Features to one-hot encode.
- **y** (*pd.Series*) – Ignored.

**Returns** Transformed data, where each categorical feature has been encoded into numerical columns using one-hot encoding.

**Return type** *pd.DataFrame*

**class** evalml.pipelines.**PerColumnImputer** (*impute\_strategies=None*, *de-*  
*fault\_impute\_strategy='most\_frequent'*, *de-*  
*random\_seed=0, \*\*kwargs*)

Imputes missing data according to a specified imputation strategy per column.

#### Parameters

- **impute\_strategies** (*dict*) – Column and {"impute\_strategy": strategy, "fill\_value":value} pairings. Valid values for impute strategy include "mean", "median", "most\_frequent", "constant" for numerical data, and "most\_frequent", "constant" for object data types. Defaults to None, which uses "most\_frequent" for all columns. When *impute\_strategy* == "constant", *fill\_value* is used to replace missing data. When None, uses 0 when imputing numerical data and "missing\_value" for strings or object data types.
- **default\_impute\_strategy** (*str*) – Impute strategy to fall back on when none is provided for a certain column. Valid values include "mean", "median", "most\_frequent", "constant" for numerical data, and "most\_frequent", "constant" for object data types. Defaults to "most\_frequent".
- **random\_seed** (*int*) – Seed for the random number generator. Defaults to 0.

#### Attributes

hyper-parameter_ranges	{}
model_family	ModelFamily.NONE
modifies_features	True
modifies_target	False
name	Per Column Imputer

#### Methods

<code>clone</code>	Constructs a new component with the same parameters and random state.
<code>default_parameters</code>	Returns the default parameters for this component.
<code>describe</code>	Describe a component and its parameters
<code>fit</code>	Fits imputers on input data
<code>fit_transform</code>	Fits on X and transforms X
<code>load</code>	Loads component at file path
<code>needs_fitting</code>	Returns boolean determining if component needs fitting before
<code>parameters</code>	Returns the parameters which were used to initialize the component
<code>save</code>	Saves component at file path
<code>transform</code>	Transforms input data by imputing missing values.

**clone** (*self*)

Constructs a new component with the same parameters and random state.

**Returns** A new instance of this component with identical parameters and random state.

**default\_parameters** (*cls*)

Returns the default parameters for this component.

Our convention is that `Component.default_parameters == Component().parameters`.

**Returns** default parameters for this component.

**Return type** dict

**describe** (*self*, *print\_name=False*, *return\_dict=False*)

Describe a component and its parameters

**Parameters**

- **print\_name** (*bool*, *optional*) – whether to print name of component
- **return\_dict** (*bool*, *optional*) – whether to return description as dictionary in the format {"name": name, "parameters": parameters}

**Returns** prints and returns dictionary

**Return type** None or dict

**fit** (*self*, *X*, *y=None*)

Fits imputers on input data

**Parameters**

- **X** (*pd.DataFrame* or *np.ndarray*) – The input training data of shape [*n\_samples*, *n\_features*] to fit.
- **y** (*pd.Series*, *optional*) – The target training data of length [*n\_samples*]. Ignored.

**Returns** self

**fit\_transform** (*self*, *X*, *y=None*)

Fits on X and transforms X

**Parameters**

- **X** (*pd.DataFrame*) – Data to fit and transform
- **y** (*pd.Series*) – Target data

**Returns** Transformed X

**Return type** `pd.DataFrame`

**static load** (*file\_path*)

Loads component at file path

**Parameters** `file_path` (*str*) – Location to load file

**Returns** `ComponentBase` object

**needs\_fitting** (*self*)

Returns boolean determining if component needs fitting before calling `predict`, `predict_proba`, `transform`, or `feature_importances`. This can be overridden to `False` for components that do not need to be fit or whose fit methods do nothing.

**property parameters** (*self*)

Returns the parameters which were used to initialize the component

**save** (*self*, *file\_path*, *pickle\_protocol=cloudpickle.DEFAULT\_PROTOCOL*)

Saves component at file path

**Parameters**

- `file_path` (*str*) – Location to save file
- `pickle_protocol` (*int*) – The pickle data stream format.

**Returns** `None`

**transform** (*self*, *X*, *y=None*)

Transforms input data by imputing missing values.

**Parameters**

- `X` (*pd.DataFrame* or *np.ndarray*) – The input training data of shape `[n_samples, n_features]` to transform.
- `y` (*pd.Series*, *optional*) – The target training data of length `[n_samples]`. Ignored.

**Returns** Transformed X

**Return type** `pd.DataFrame`

**class** `evalml.pipelines.PipelineBase` (*component\_graph*, *parameters=None*, *custom\_name=None*, *random\_seed=0*)

Machine learning pipeline made out of transformers and a estimator.

**Parameters**

- **component\_graph** (*list* or *dict*) – List of components in order. Accepts strings or `ComponentBase` subclasses in the list. Note that when duplicate components are specified in a list, the duplicate component names will be modified with the component’s index in the list. For example, the component graph `[Imputer, One Hot Encoder, Imputer, Logistic Regression Classifier]` will have names `["Imputer", "One Hot Encoder", "Imputer_2", "Logistic Regression Classifier"]`.
- **parameters** (*dict*) – Dictionary with component names as keys and dictionary of that component’s parameters as values. An empty dictionary or `None` implies using all default values for component parameters. Defaults to `None`.
- **custom\_name** (*str*) – Custom name for the pipeline. Defaults to `None`.
- **random\_seed** (*int*) – Seed for the random number generator. Defaults to 0.

**Attributes**

<b>problem_type</b>	None
---------------------	------

**Methods**

<i>can_tune_threshold_with_objective</i>	Determine whether the threshold of a binary classification pipeline can be tuned.
<i>clone</i>	Constructs a new pipeline with the same components, parameters, and random state.
<i>compute_estimator_features</i>	Transforms the data by applying all pre-processing components.
<i>create_objectives</i>	
<i>custom_name</i>	Custom name of the pipeline.
<i>describe</i>	Outputs pipeline details including component parameters
<i>feature_importance</i>	Importance associated with each feature. Features dropped by the feature selection are excluded.
<i>fit</i>	Build a model.
<i>get_component</i>	Returns component by name
<i>get_hyperparameter_ranges</i>	Returns hyperparameter ranges from all components as a dictionary.
<i>graph</i>	Generate an image representing the pipeline graph.
<i>graph_feature_importance</i>	Generate a bar graph of the pipeline's feature importance
<i>inverse_transform</i>	Apply component <code>inverse_transform</code> methods to estimator predictions in reverse order.
<i>load</i>	Loads pipeline at file path
<i>model_family</i>	Returns model family of this pipeline.
<i>name</i>	Name of the pipeline.
<i>new</i>	Constructs a new instance of the pipeline with the same component graph but with a different set of parameters.
<i>parameters</i>	Parameter dictionary for this pipeline.
<i>predict</i>	Make predictions using selected features.
<i>save</i>	Saves pipeline at file path
<i>score</i>	Evaluate model performance on current and additional objectives.
<i>summary</i>	A short summary of the pipeline structure, describing the list of components used.

**can\_tune\_threshold\_with\_objective** (*self*, *objective*)

Determine whether the threshold of a binary classification pipeline can be tuned.

**Parameters**

- **pipeline** (`PipelineBase`) – Binary classification pipeline.
- **objective** – Primary `AutoMLSearch` objective.

**clone** (*self*)

Constructs a new pipeline with the same components, parameters, and random state.



**Returns** A new instance of this pipeline with identical components, parameters, and random state.

**compute\_estimator\_features** (*self*, *X*, *y=None*)

Transforms the data by applying all pre-processing components.

**Parameters** *X* (*pd.DataFrame*) – Input data to the pipeline to transform.

**Returns** New transformed features.

**Return type** *pd.DataFrame*

**static create\_objectives** (*objectives*)

**property custom\_name** (*self*)

Custom name of the pipeline.

**describe** (*self*, *return\_dict=False*)

Outputs pipeline details including component parameters

**Parameters** *return\_dict* (*bool*) – If True, return dictionary of information about pipeline. Defaults to False.

**Returns** Dictionary of all component parameters if *return\_dict* is True, else None

**Return type** *dict*

**property feature\_importance** (*self*)

Importance associated with each feature. Features dropped by the feature selection are excluded.

**Returns** *pd.DataFrame* including feature names and their corresponding importance

**abstract fit** (*self*, *X*, *y*)

Build a model.

**Parameters**

- *X* (*pd.DataFrame* or *np.ndarray*) – The input training data of shape [n\_samples, n\_features].
- *y* (*pd.Series*, *np.ndarray*) – The target training data of length [n\_samples].

**Returns** *self*

**get\_component** (*self*, *name*)

Returns component by name

**Parameters** *name* (*str*) – Name of component

**Returns** Component to return

**Return type** *Component*

**get\_hyperparameter\_ranges** (*self*, *custom\_hyperparameters*)

Returns hyperparameter ranges from all components as a dictionary.

**Parameters** *custom\_hyperparameters* (*dict*) – Custom hyperparameters for the pipeline.

**Returns** Dictionary of hyperparameter ranges for each component in the pipeline.

**Return type** *dict*

**graph** (*self*, *filepath=None*)

Generate an image representing the pipeline graph.

**Parameters** `filepath` (*str*, *optional*) – Path to where the graph should be saved. If set to None (as by default), the graph will not be saved.

**Returns** Graph object that can be directly displayed in Jupyter notebooks.

**Return type** `graphviz.Digraph`

**graph\_feature\_importance** (*self*, *importance\_threshold=0*)

Generate a bar graph of the pipeline's feature importance

**Parameters** `importance_threshold` (*float*, *optional*) – If provided, graph features with a permutation importance whose absolute value is larger than `importance_threshold`. Defaults to zero.

**Returns** `plotly.Figure`, a bar graph showing features and their corresponding importance

**inverse\_transform** (*self*, *y*)

Apply component `inverse_transform` methods to estimator predictions in reverse order.

Components that implement `inverse_transform` are `PolynomialDetrender`, `LabelEncoder` (tbd).

**Parameters** `y` (*pd.Series*) – Final component features

**static load** (*file\_path*)

Loads pipeline at file path

**Parameters** `file_path` (*str*) – location to load file

**Returns** `PipelineBase` object

**property model\_family** (*self*)

Returns model family of this pipeline.

**property name** (*self*)

Name of the pipeline.

**new** (*self*, *parameters*, *random\_seed=0*)

**Constructs a new instance of the pipeline with the same component graph but with a different set of parameters.**

Not to be confused with python's `__new__` method.

**Parameters**

- **parameters** (*dict*) – Dictionary with component names as keys and dictionary of that component's parameters as values. An empty dictionary or None implies using all default values for component parameters. Defaults to None.
- **random\_seed** (*int*) – Seed for the random number generator. Defaults to 0.

**Returns** A new instance of this pipeline with identical components.

**property parameters** (*self*)

Parameter dictionary for this pipeline.

**Returns** Dictionary of all component parameters.

**Return type** `dict`

**predict** (*self*, *X*, *objective=None*)

Make predictions using selected features.

**Parameters**

- **X** (*pd.DataFrame*, or *np.ndarray*) – Data of shape `[n_samples, n_features]`.
- **objective** (*Object* or *string*) – The objective to use to make predictions.

**Returns** Predicted values.

**Return type** `pd.Series`

**save** (*self*, *file\_path*, *pickle\_protocol=cloudpickle.DEFAULT\_PROTOCOL*)

Saves pipeline at file path

**Parameters**

- **file\_path** (*str*) – location to save file
- **pickle\_protocol** (*int*) – the pickle data stream format.

**Returns** `None`

**abstract score** (*self*, *X*, *y*, *objectives*)

Evaluate model performance on current and additional objectives.

**Parameters**

- **X** (*pd.DataFrame* or *np.ndarray*) – Data of shape `[n_samples, n_features]`.
- **y** (*pd.Series*, *np.ndarray*) – True labels of length `[n_samples]`.
- **objectives** (*list*) – Non-empty list of objectives to score on.

**Returns** Ordered dictionary of objective scores.

**Return type** `dict`

**property summary** (*self*)

A short summary of the pipeline structure, describing the list of components used. Example: Logistic Regression Classifier w/ Simple Imputer + One Hot Encoder

```
class evalml.pipelines.ProphetRegressor (date_column='ds', changepoint_prior_scale=0.05,
                                         seasonality_prior_scale=10,             hol-
                                         idays_prior_scale=10,                 seasonal-
                                         ity_mode='additive',                random_seed=0,
                                         stan_backend='CMDSTANPY', **kwargs)
```

Prophet is a procedure for forecasting time series data based on an additive model where non-linear trends are fit with yearly, weekly, and daily seasonality, plus holiday effects. It works best with time series that have strong seasonal effects and several seasons of historical data. Prophet is robust to missing data and shifts in the trend, and typically handles outliers well.

More information here: <https://facebook.github.io/prophet/>

**Attributes**

<b>hyper-parameter_ranges</b>	{ "changepoint_prior_scale": Real(0.001, 0.5), "seasonality_prior_scale": Real(0.01, 10), "holidays_prior_scale": Real(0.01, 10), "seasonality_mode": ["additive", "multiplicative"], }
<b>model_family</b>	ModelFamily.PROPHET
<b>modifies_features</b>	True
<b>modifies_target</b>	False
<b>name</b>	Prophet Regressor
<b>pre-dict_uses_y</b>	False
<b>supported_problem_types</b>	[ProblemTypes.TIME_SERIES_REGRESSION]

## Methods

<code>build_prophet_df</code>	
<code>clone</code>	Constructs a new component with the same parameters and random state.
<code>default_parameters</code>	Returns the default parameters for this component.
<code>describe</code>	Describe a component and its parameters
<code>feature_importance</code>	Returns array of 0's with len(1) as feature_importance is not defined for Prophet regressor.
<code>fit</code>	Fits component to data
<code>get_params</code>	
<code>load</code>	Loads component at file path
<code>needs_fitting</code>	Returns boolean determining if component needs fitting before
<code>parameters</code>	Returns the parameters which were used to initialize the component
<code>predict</code>	Make predictions using selected features.
<code>predict_proba</code>	Make probability estimates for labels.
<code>save</code>	Saves component at file path

**static build\_prophet\_df** (*X*, *y=None*, *date\_column='ds'*)

**clone** (*self*)

Constructs a new component with the same parameters and random state.

**Returns** A new instance of this component with identical parameters and random state.

**default\_parameters** (*cls*)

Returns the default parameters for this component.

Our convention is that `Component.default_parameters == Component().parameters`.

**Returns** default parameters for this component.

**Return type** dict

**describe** (*self*, *print\_name=False*, *return\_dict=False*)

Describe a component and its parameters

**Parameters**

- **print\_name** (*bool*, *optional*) – whether to print name of component
- **return\_dict** (*bool*, *optional*) – whether to return description as dictionary in the format {"name": name, "parameters": parameters}

**Returns** prints and returns dictionary

**Return type** None or dict

**property feature\_importance** (*self*)

Returns array of 0's with len(1) as feature\_importance is not defined for Prophet regressor.

**fit** (*self*, *X*, *y=None*)

Fits component to data

**Parameters**

- **X** (*list*, *pd.DataFrame* or *np.ndarray*) – The input training data of shape [n\_samples, n\_features]
- **y** (*list*, *pd.Series*, *np.ndarray*, *optional*) – The target training data of length [n\_samples]

**Returns** self

**get\_params** (*self*)

**static load** (*file\_path*)

Loads component at file path

**Parameters** **file\_path** (*str*) – Location to load file

**Returns** ComponentBase object

**needs\_fitting** (*self*)

Returns boolean determining if component needs fitting before calling predict, predict\_proba, transform, or feature\_importances. This can be overridden to False for components that do not need to be fit or whose fit methods do nothing.

**property parameters** (*self*)

Returns the parameters which were used to initialize the component

**predict** (*self*, *X*, *y=None*)

Make predictions using selected features.

**Parameters** **X** (*pd.DataFrame*, *np.ndarray*) – Data of shape [n\_samples, n\_features]

**Returns** Predicted values

**Return type** pd.Series

**predict\_proba** (*self*, *X*)

Make probability estimates for labels.

**Parameters** **X** (*pd.DataFrame*, or *np.ndarray*) – Features

**Returns** Probability estimates

**Return type** pd.Series

**save** (*self*, *file\_path*, *pickle\_protocol=cloudpickle.DEFAULT\_PROTOCOL*)

Saves component at file path

**Parameters**

- **file\_path** (*str*) – Location to save file
- **pickle\_protocol** (*int*) – The pickle data stream format.

**Returns** None

**class** evalml.pipelines.**RandomForestClassifier** (*n\_estimators=100*, *max\_depth=6*,  
*n\_jobs=-1*, *random\_seed=0*, *\*\*kwargs*)

Random Forest Classifier.

**Parameters**

- **n\_estimators** (*float*) – The number of trees in the forest. Defaults to 100.
- **max\_depth** (*int*) – Maximum tree depth for base learners. Defaults to 6.
- **n\_jobs** (*int* or *None*) – Number of jobs to run in parallel. -1 uses all processes. Defaults to -1.
- **random\_seed** (*int*) – Seed for the random number generator. Defaults to 0.

## Attributes

<b>hyper-parameter_ranges</b>	{ "n_estimators": Integer(10, 1000), "max_depth": Integer(1, 10), }
<b>model_family</b>	ModelFamily.RANDOM_FOREST
<b>modifies_features</b>	True
<b>modifies_target</b>	False
<b>name</b>	Random Forest Classifier
<b>pre_dict_uses_y</b>	False
<b>supported_problem_types</b>	[ ProblemTypes.BINARY, ProblemTypes.MULTICLASS, ProblemTypes.TIME_SERIES_BINARY, ProblemTypes.TIME_SERIES_MULTICLASS, ]

## Methods

<i>clone</i>	Constructs a new component with the same parameters and random state.
<i>default_parameters</i>	Returns the default parameters for this component.
<i>describe</i>	Describe a component and its parameters
<i>feature_importance</i>	Returns importance associated with each feature.
<i>fit</i>	Fits component to data
<i>load</i>	Loads component at file path
<i>needs_fitting</i>	Returns boolean determining if component needs fitting before
<i>parameters</i>	Returns the parameters which were used to initialize the component
<i>predict</i>	Make predictions using selected features.
<i>predict_proba</i>	Make probability estimates for labels.
<i>save</i>	Saves component at file path

**clone** (*self*)

Constructs a new component with the same parameters and random state.

**Returns** A new instance of this component with identical parameters and random state.

**default\_parameters** (*cls*)

Returns the default parameters for this component.

Our convention is that `Component.default_parameters == Component().parameters`.

**Returns** default parameters for this component.

**Return type** dict

**describe** (*self*, *print\_name=False*, *return\_dict=False*)

Describe a component and its parameters

**Parameters**

- **print\_name** (*bool*, *optional*) – whether to print name of component
- **return\_dict** (*bool*, *optional*) – whether to return description as dictionary in the format {"name": name, "parameters": parameters}

**Returns** prints and returns dictionary

**Return type** None or dict

**property feature\_importance** (*self*)

Returns importance associated with each feature.

**Returns** Importance associated with each feature

**Return type** np.ndarray

**fit** (*self*, *X*, *y=None*)

Fits component to data

**Parameters**

- **X** (*list*, *pd.DataFrame* or *np.ndarray*) – The input training data of shape [n\_samples, n\_features]
- **y** (*list*, *pd.Series*, *np.ndarray*, *optional*) – The target training data of length [n\_samples]

**Returns** self

**static load** (*file\_path*)

Loads component at file path

**Parameters** **file\_path** (*str*) – Location to load file

**Returns** ComponentBase object

**needs\_fitting** (*self*)

Returns boolean determining if component needs fitting before calling predict, predict\_proba, transform, or feature\_importances. This can be overridden to False for components that do not need to be fit or whose fit methods do nothing.

**property parameters** (*self*)

Returns the parameters which were used to initialize the component

**predict** (*self*, *X*)

Make predictions using selected features.

**Parameters** **X** (*pd.DataFrame*, *np.ndarray*) – Data of shape [n\_samples, n\_features]

**Returns** Predicted values

**Return type** pd.Series

**predict\_proba** (*self*, *X*)

Make probability estimates for labels.

**Parameters** **X** (*pd.DataFrame*, or *np.ndarray*) – Features

**Returns** Probability estimates

**Return type** pd.Series

**save** (*self*, *file\_path*, *pickle\_protocol=cloudpickle.DEFAULT\_PROTOCOL*)

Saves component at file path

**Parameters**

- **file\_path** (*str*) – Location to save file
- **pickle\_protocol** (*int*) – The pickle data stream format.

**Returns** None

**class** evalml.pipelines.RandomForestRegressor (*n\_estimators=100, max\_depth=6, n\_jobs=-1, random\_seed=0, \*\*kwargs*)

Random Forest Regressor.

#### Parameters

- **n\_estimators** (*float*) – The number of trees in the forest. Defaults to 100.
- **max\_depth** (*int*) – Maximum tree depth for base learners. Defaults to 6.
- **n\_jobs** (*int or None*) – Number of jobs to run in parallel. -1 uses all processes. Defaults to -1.
- **random\_seed** (*int*) – Seed for the random number generator. Defaults to 0.

#### Attributes

<b>hyper-parameter_ranges</b>	{ "n_estimators": Integer(10, 1000), "max_depth": Integer(1, 32), }
<b>model_family</b>	ModelFamily.RANDOM_FOREST
<b>modifies_features</b>	True
<b>modifies_target</b>	False
<b>name</b>	Random Forest Regressor
<b>predict_uses_y</b>	False
<b>supported_problem_types</b>	[ ProblemTypes.REGRESSION, ProblemTypes.TIME_SERIES_REGRESSION, ]

#### Methods

<i>clone</i>	Constructs a new component with the same parameters and random state.
<i>default_parameters</i>	Returns the default parameters for this component.
<i>describe</i>	Describe a component and its parameters
<i>feature_importance</i>	Returns importance associated with each feature.
<i>fit</i>	Fits component to data
<i>load</i>	Loads component at file path
<i>needs_fitting</i>	Returns boolean determining if component needs fitting before
<i>parameters</i>	Returns the parameters which were used to initialize the component
<i>predict</i>	Make predictions using selected features.
<i>predict_proba</i>	Make probability estimates for labels.
<i>save</i>	Saves component at file path

**clone** (*self*)

Constructs a new component with the same parameters and random state.

**Returns** A new instance of this component with identical parameters and random state.

**default\_parameters** (*cls*)

Returns the default parameters for this component.

Our convention is that `Component.default_parameters == Component().parameters`.



**Returns** default parameters for this component.

**Return type** dict

**describe** (*self*, *print\_name=False*, *return\_dict=False*)

Describe a component and its parameters

**Parameters**

- **print\_name** (*bool*, *optional*) – whether to print name of component
- **return\_dict** (*bool*, *optional*) – whether to return description as dictionary in the format {"name": name, "parameters": parameters}

**Returns** prints and returns dictionary

**Return type** None or dict

**property feature\_importance** (*self*)

Returns importance associated with each feature.

**Returns** Importance associated with each feature

**Return type** np.ndarray

**fit** (*self*, *X*, *y=None*)

Fits component to data

**Parameters**

- **X** (*list*, *pd.DataFrame* or *np.ndarray*) – The input training data of shape [n\_samples, n\_features]
- **y** (*list*, *pd.Series*, *np.ndarray*, *optional*) – The target training data of length [n\_samples]

**Returns** self

**static load** (*file\_path*)

Loads component at file path

**Parameters** **file\_path** (*str*) – Location to load file

**Returns** ComponentBase object

**needs\_fitting** (*self*)

Returns boolean determining if component needs fitting before calling predict, predict\_proba, transform, or feature\_importances. This can be overridden to False for components that do not need to be fit or whose fit methods do nothing.

**property parameters** (*self*)

Returns the parameters which were used to initialize the component

**predict** (*self*, *X*)

Make predictions using selected features.

**Parameters** **X** (*pd.DataFrame*, *np.ndarray*) – Data of shape [n\_samples, n\_features]

**Returns** Predicted values

**Return type** pd.Series

**predict\_proba** (*self*, *X*)

Make probability estimates for labels.

**Parameters** **X** (*pd.DataFrame*, or *np.ndarray*) – Features

**Returns** Probability estimates

**Return type** `pd.Series`

**save** (*self*, *file\_path*, *pickle\_protocol=cloudpickle.DEFAULT\_PROTOCOL*)

Saves component at file path

**Parameters**

- **file\_path** (*str*) – Location to save file
- **pickle\_protocol** (*int*) – The pickle data stream format.

**Returns** None

**class** `evalml.pipelines.RegressionPipeline` (*component\_graph*, *parameters=None*, *custom\_name=None*, *random\_seed=0*)

Pipeline subclass for all regression pipelines.

**Parameters**

- **component\_graph** (*list or dict*) – List of components in order. Accepts strings or `ComponentBase` subclasses in the list. Note that when duplicate components are specified in a list, the duplicate component names will be modified with the component’s index in the list. For example, the component graph [Imputer, One Hot Encoder, Imputer, Logistic Regression Classifier] will have names [“Imputer”, “One Hot Encoder”, “Imputer\_2”, “Logistic Regression Classifier”]
- **parameters** (*dict*) – Dictionary with component names as keys and dictionary of that component’s parameters as values. An empty dictionary or None implies using all default values for component parameters. Defaults to None.
- **custom\_name** (*str*) – Custom name for the pipeline. Defaults to None.
- **random\_seed** (*int*) – Seed for the random number generator. Defaults to 0.

**Attributes**

<b>problem_type</b>	<code>ProblemTypes.REGRESSION</code>
---------------------	--------------------------------------

**Methods**

<code>can_tune_threshold_with_objective</code>	Determine whether the threshold of a binary classification pipeline can be tuned.
<code>clone</code>	Constructs a new pipeline with the same components, parameters, and random state.
<code>compute_estimator_features</code>	Transforms the data by applying all pre-processing components.
<code>create_objectives</code>	
<code>custom_name</code>	Custom name of the pipeline.
<code>describe</code>	Outputs pipeline details including component parameters
<code>feature_importance</code>	Importance associated with each feature. Features dropped by the feature selection are excluded.
<code>fit</code>	Build a regression model.
<code>get_component</code>	Returns component by name

continues on next page

Table 656 – continued from previous page

<code>get_hyperparameter_ranges</code>	Returns hyperparameter ranges from all components as a dictionary.
<code>graph</code>	Generate an image representing the pipeline graph.
<code>graph_feature_importance</code>	Generate a bar graph of the pipeline’s feature importance
<code>inverse_transform</code>	Apply component <code>inverse_transform</code> methods to estimator predictions in reverse order.
<code>load</code>	Loads pipeline at file path
<code>model_family</code>	Returns model family of this pipeline.
<code>name</code>	Name of the pipeline.
<code>new</code>	Constructs a new instance of the pipeline with the same component graph but with a different set of parameters.
<code>parameters</code>	Parameter dictionary for this pipeline.
<code>predict</code>	Make predictions using selected features.
<code>save</code>	Saves pipeline at file path
<code>score</code>	Evaluate model performance on current and additional objectives
<code>summary</code>	A short summary of the pipeline structure, describing the list of components used.

**`can_tune_threshold_with_objective`** (*self*, *objective*)

Determine whether the threshold of a binary classification pipeline can be tuned.

**Parameters**

- **`pipeline`** (`PipelineBase`) – Binary classification pipeline.
- **`objective`** – Primary AutoMLSearch objective.

**`clone`** (*self*)

Constructs a new pipeline with the same components, parameters, and random state.

**Returns** A new instance of this pipeline with identical components, parameters, and random state.

**`compute_estimator_features`** (*self*, *X*, *y=None*)

Transforms the data by applying all pre-processing components.

**Parameters** ***X*** (`pd.DataFrame`) – Input data to the pipeline to transform.

**Returns** New transformed features.

**Return type** `pd.DataFrame`

**`static create_objectives`** (*objectives*)

**`property custom_name`** (*self*)

Custom name of the pipeline.

**`describe`** (*self*, *return\_dict=False*)

Outputs pipeline details including component parameters

**Parameters** **`return_dict`** (*bool*) – If True, return dictionary of information about pipeline. Defaults to False.

**Returns** Dictionary of all component parameters if `return_dict` is True, else None

**Return type** `dict`

**property** `feature_importance` (*self*)

Importance associated with each feature. Features dropped by the feature selection are excluded.

**Returns** `pd.DataFrame` including feature names and their corresponding importance

**fit** (*self*, *X*, *y*)

Build a regression model.

**Parameters**

- **X** (`pd.DataFrame` or `np.ndarray`) – The input training data of shape `[n_samples, n_features]`
- **y** (`pd.Series`, `np.ndarray`) – The target training data of length `[n_samples]`

**Returns** `self`

**get\_component** (*self*, *name*)

Returns component by name

**Parameters** **name** (*str*) – Name of component

**Returns** Component to return

**Return type** Component

**get\_hyperparameter\_ranges** (*self*, *custom\_hyperparameters*)

Returns hyperparameter ranges from all components as a dictionary.

**Parameters** **custom\_hyperparameters** (*dict*) – Custom hyperparameters for the pipeline.

**Returns** Dictionary of hyperparameter ranges for each component in the pipeline.

**Return type** `dict`

**graph** (*self*, *filepath=None*)

Generate an image representing the pipeline graph.

**Parameters** **filepath** (*str*, *optional*) – Path to where the graph should be saved. If set to `None` (as by default), the graph will not be saved.

**Returns** Graph object that can be directly displayed in Jupyter notebooks.

**Return type** `graphviz.Digraph`

**graph\_feature\_importance** (*self*, *importance\_threshold=0*)

Generate a bar graph of the pipeline's feature importance

**Parameters** **importance\_threshold** (*float*, *optional*) – If provided, graph features with a permutation importance whose absolute value is larger than `importance_threshold`. Defaults to zero.

**Returns** `plotly.Figure`, a bar graph showing features and their corresponding importance

**inverse\_transform** (*self*, *y*)

Apply component `inverse_transform` methods to estimator predictions in reverse order.

Components that implement `inverse_transform` are `PolynomialDetrender`, `LabelEncoder` (tbd).

**Parameters** **y** (`pd.Series`) – Final component features

**static load** (*file\_path*)

Loads pipeline at file path

**Parameters** **file\_path** (*str*) – location to load file

**Returns** PipelineBase object

**property model\_family** (*self*)  
Returns model family of this pipeline.

**property name** (*self*)  
Name of the pipeline.

**new** (*self*, *parameters*, *random\_seed=0*)

**Constructs a new instance of the pipeline with the same component graph but with a different set of parameters.**  
Not to be confused with python's `__new__` method.

#### Parameters

- **parameters** (*dict*) – Dictionary with component names as keys and dictionary of that component's parameters as values. An empty dictionary or None implies using all default values for component parameters. Defaults to None.
- **random\_seed** (*int*) – Seed for the random number generator. Defaults to 0.

**Returns** A new instance of this pipeline with identical components.

**property parameters** (*self*)  
Parameter dictionary for this pipeline.

**Returns** Dictionary of all component parameters.

**Return type** dict

**predict** (*self*, *X*, *objective=None*)  
Make predictions using selected features.

#### Parameters

- **X** (*pd.DataFrame*, or *np.ndarray*) – Data of shape [n\_samples, n\_features].
- **objective** (*Object* or *string*) – The objective to use to make predictions.

**Returns** Predicted values.

**Return type** pd.Series

**save** (*self*, *file\_path*, *pickle\_protocol=cloudpickle.DEFAULT\_PROTOCOL*)  
Saves pipeline at file path

#### Parameters

- **file\_path** (*str*) – location to save file
- **pickle\_protocol** (*int*) – the pickle data stream format.

**Returns** None

**score** (*self*, *X*, *y*, *objectives*)  
Evaluate model performance on current and additional objectives

#### Parameters

- **X** (*pd.DataFrame*, or *np.ndarray*) – Data of shape [n\_samples, n\_features]
- **y** (*pd.Series*, or *np.ndarray*) – True values of length [n\_samples]
- **objectives** (*list*) – Non-empty list of objectives to score on

**Returns** Ordered dictionary of objective scores

**Return type** dict

**property summary** (*self*)

A short summary of the pipeline structure, describing the list of components used. Example: Logistic Regression Classifier w/ Simple Imputer + One Hot Encoder

```
class evalml.pipelines.RFClassifierSelectFromModel (number_features=None,
                                                    n_estimators=10,
                                                    max_depth=None,           per-
                                                    cent_features=0.5,   threshold=-
                                                    np.inf,   n_jobs=-1,   ran-
                                                    dom_seed=0, **kwargs)
```

Selects top features based on importance weights using a Random Forest classifier.

#### Parameters

- **number\_features** (*int*) – The maximum number of features to select. If both *percent\_features* and *number\_features* are specified, take the greater number of features. Defaults to 0.5. Defaults to None.
- **n\_estimators** (*float*) – The number of trees in the forest. Defaults to 100.
- **max\_depth** (*int*) – Maximum tree depth for base learners. Defaults to 6.
- **percent\_features** (*float*) – Percentage of features to use. If both *percent\_features* and *number\_features* are specified, take the greater number of features. Defaults to 0.5.
- **threshold** (*string or float*) – The threshold value to use for feature selection. Features whose importance is greater or equal are kept while the others are discarded. If “median”, then the threshold value is the median of the feature importances. A scaling factor (e.g., “1.25\*mean”) may also be used. Defaults to -np.inf.
- **n\_jobs** (*int or None*) – Number of jobs to run in parallel. -1 uses all processes. Defaults to -1.
- **random\_seed** (*int*) – Seed for the random number generator. Defaults to 0.

#### Attributes

<b>hyper-parameter_ranges</b>	{ “percent_features”: Real(0.01, 1), “threshold”: [“mean”, -np.inf], }
<b>model_family</b>	ModelFamily.NONE
<b>modifies_features</b>	True
<b>modifies_target</b>	False
<b>name</b>	RF Classifier Select From Model

#### Methods

<i>clone</i>	Constructs a new component with the same parameters and random state.
<i>default_parameters</i>	Returns the default parameters for this component.
<i>describe</i>	Describe a component and its parameters
<i>fit</i>	Fits component to data
<i>fit_transform</i>	Fits on X and transforms X

continues on next page

Table 657 – continued from previous page

<code>get_names</code>	Get names of selected features.
<code>load</code>	Loads component at file path
<code>needs_fitting</code>	Returns boolean determining if component needs fitting before
<code>parameters</code>	Returns the parameters which were used to initialize the component
<code>save</code>	Saves component at file path
<code>transform</code>	Transforms input data by selecting features. If the component_obj does not have a transform method, will raise an <code>MethodPropertyNotFoundError</code> exception.

**clone** (*self*)

Constructs a new component with the same parameters and random state.

**Returns** A new instance of this component with identical parameters and random state.

**default\_parameters** (*cls*)

Returns the default parameters for this component.

Our convention is that `Component.default_parameters == Component().parameters`.

**Returns** default parameters for this component.

**Return type** dict

**describe** (*self*, *print\_name=False*, *return\_dict=False*)

Describe a component and its parameters

**Parameters**

- **print\_name** (*bool*, *optional*) – whether to print name of component
- **return\_dict** (*bool*, *optional*) – whether to return description as dictionary in the format {"name": name, "parameters": parameters}

**Returns** prints and returns dictionary

**Return type** None or dict

**fit** (*self*, *X*, *y=None*)

Fits component to data

**Parameters**

- **X** (*list*, *pd.DataFrame* or *np.ndarray*) – The input training data of shape [n\_samples, n\_features]
- **y** (*list*, *pd.Series*, *np.ndarray*, *optional*) – The target training data of length [n\_samples]

**Returns** self

**fit\_transform** (*self*, *X*, *y=None*)

Fits on X and transforms X

**Parameters**

- **X** (*pd.DataFrame*) – Data to fit and transform
- **y** (*pd.Series*) – Target data

**Returns** Transformed X

**Return type** `pd.DataFrame`

**get\_names** (*self*)

Get names of selected features.

**Returns** List of the names of features selected

**Return type** `list[str]`

**static load** (*file\_path*)

Loads component at file path

**Parameters** **file\_path** (*str*) – Location to load file

**Returns** `ComponentBase` object

**needs\_fitting** (*self*)

Returns boolean determining if component needs fitting before calling `predict`, `predict_proba`, `transform`, or `feature_importances`. This can be overridden to `False` for components that do not need to be fit or whose fit methods do nothing.

**property parameters** (*self*)

Returns the parameters which were used to initialize the component

**save** (*self*, *file\_path*, *pickle\_protocol=cloudpickle.DEFAULT\_PROTOCOL*)

Saves component at file path

**Parameters**

- **file\_path** (*str*) – Location to save file
- **pickle\_protocol** (*int*) – The pickle data stream format.

**Returns** `None`

**transform** (*self*, *X*, *y=None*)

Transforms input data by selecting features. If the component\_obj does not have a transform method, will raise an `MethodPropertyNotFoundError` exception.

**Parameters**

- **X** (*pd.DataFrame*) – Data to transform.
- **y** (*pd.Series*, *optional*) – Target data. Ignored.

**Returns** Transformed X

**Return type** `pd.DataFrame`

```
class evalml.pipelines.RFRegressorSelectFromModel (number_features=None,  
                                                  n_estimators=10,  
                                                  max_depth=None,           per-  
                                                  cent_features=0.5,       threshold=-  
                                                  np.inf, n_jobs=- 1, random_seed=0,  
                                                  **kwargs)
```

Selects top features based on importance weights using a Random Forest regressor.

**Parameters**

- **number\_features** (*int*) – The maximum number of features to select. If both `percent_features` and `number_features` are specified, take the greater number of features. Defaults to 0.5. Defaults to `None`.
- **n\_estimators** (*float*) – The number of trees in the forest. Defaults to 100.
- **max\_depth** (*int*) – Maximum tree depth for base learners. Defaults to 6.



- **percent\_features** (*float*) – Percentage of features to use. If both percent\_features and number\_features are specified, take the greater number of features. Defaults to 0.5.
- **threshold** (*string or float*) – The threshold value to use for feature selection. Features whose importance is greater or equal are kept while the others are discarded. If “median”, then the threshold value is the median of the feature importances. A scaling factor (e.g., “1.25\*mean”) may also be used. Defaults to -np.inf.
- **n\_jobs** (*int or None*) – Number of jobs to run in parallel. -1 uses all processes. Defaults to -1.
- **random\_seed** (*int*) – Seed for the random number generator. Defaults to 0.

#### Attributes

<b>hyper-parameter_ranges</b>	{ “percent_features”: Real(0.01, 1), “threshold”: [“mean”, -np.inf], }
<b>model_family</b>	ModelFamily.NONE
<b>modifies_features</b>	True
<b>modifies_target</b>	False
<b>name</b>	RF Regressor Select From Model

#### Methods

<i>clone</i>	Constructs a new component with the same parameters and random state.
<i>default_parameters</i>	Returns the default parameters for this component.
<i>describe</i>	Describe a component and its parameters
<i>fit</i>	Fits component to data
<i>fit_transform</i>	Fits on X and transforms X
<i>get_names</i>	Get names of selected features.
<i>load</i>	Loads component at file path
<i>needs_fitting</i>	Returns boolean determining if component needs fitting before
<i>parameters</i>	Returns the parameters which were used to initialize the component
<i>save</i>	Saves component at file path
<i>transform</i>	Transforms input data by selecting features. If the component_obj does not have a transform method, will raise an MethodPropertyNotFoundError exception.

#### **clone** (*self*)

Constructs a new component with the same parameters and random state.

**Returns** A new instance of this component with identical parameters and random state.

#### **default\_parameters** (*cls*)

Returns the default parameters for this component.

Our convention is that `Component.default_parameters == Component().parameters`.

**Returns** default parameters for this component.

**Return type** dict

**describe** (*self*, *print\_name=False*, *return\_dict=False*)

Describe a component and its parameters

**Parameters**

- **print\_name** (*bool*, *optional*) – whether to print name of component
- **return\_dict** (*bool*, *optional*) – whether to return description as dictionary in the format {"name": name, "parameters": parameters}

**Returns** prints and returns dictionary

**Return type** None or dict

**fit** (*self*, *X*, *y=None*)

Fits component to data

**Parameters**

- **X** (*list*, *pd.DataFrame* or *np.ndarray*) – The input training data of shape [n\_samples, n\_features]
- **y** (*list*, *pd.Series*, *np.ndarray*, *optional*) – The target training data of length [n\_samples]

**Returns** self

**fit\_transform** (*self*, *X*, *y=None*)

Fits on X and transforms X

**Parameters**

- **X** (*pd.DataFrame*) – Data to fit and transform
- **y** (*pd.Series*) – Target data

**Returns** Transformed X

**Return type** pd.DataFrame

**get\_names** (*self*)

Get names of selected features.

**Returns** List of the names of features selected

**Return type** list[str]

**static load** (*file\_path*)

Loads component at file path

**Parameters** **file\_path** (*str*) – Location to load file

**Returns** ComponentBase object

**needs\_fitting** (*self*)

Returns boolean determining if component needs fitting before calling predict, predict\_proba, transform, or feature\_importances. This can be overridden to False for components that do not need to be fit or whose fit methods do nothing.

**property parameters** (*self*)

Returns the parameters which were used to initialize the component

**save** (*self*, *file\_path*, *pickle\_protocol=cloudpickle.DEFAULT\_PROTOCOL*)

Saves component at file path

**Parameters**

- **file\_path** (*str*) – Location to save file
- **pickle\_protocol** (*int*) – The pickle data stream format.

**Returns** None**transform** (*self*, *X*, *y=None*)

Transforms input data by selecting features. If the component\_obj does not have a transform method, will raise an `MethodPropertyNotFoundError` exception.

**Parameters**

- **X** (*pd.DataFrame*) – Data to transform.
- **y** (*pd.Series*, *optional*) – Target data. Ignored.

**Returns** Transformed X**Return type** `pd.DataFrame`

**class** `evalml.pipelines.SimpleImputer` (*impute\_strategy='most\_frequent'*, *fill\_value=None*, *random\_seed=0*, *\*\*kwargs*)

Imputes missing data according to a specified imputation strategy.

**Parameters**

- **impute\_strategy** (*string*) – Impute strategy to use. Valid values include “mean”, “median”, “most\_frequent”, “constant” for numerical data, and “most\_frequent”, “constant” for object data types.
- **fill\_value** (*string*) – When `impute_strategy == “constant”`, `fill_value` is used to replace missing data. Defaults to 0 when imputing numerical data and “missing\_value” for strings or object data types.
- **random\_seed** (*int*) – Seed for the random number generator. Defaults to 0.

**Attributes**

<b>hyper-parameter_ranges</b>	{ “impute_strategy”: [“mean”, “median”, “most_frequent”]}
<b>model_family</b>	ModelFamily.NONE
<b>modifies_features</b>	True
<b>modifies_target</b>	False
<b>name</b>	Simple Imputer

**Methods**

<i>clone</i>	Constructs a new component with the same parameters and random state.
<i>default_parameters</i>	Returns the default parameters for this component.
<i>describe</i>	Describe a component and its parameters
<i>fit</i>	Fits imputer to data. ‘None’ values are converted to <code>np.nan</code> before imputation and are
<i>fit_transform</i>	Fits on X and transforms X

continues on next page

Table 659 – continued from previous page

<code>load</code>	Loads component at file path
<code>needs_fitting</code>	Returns boolean determining if component needs fitting before
<code>parameters</code>	Returns the parameters which were used to initialize the component
<code>save</code>	Saves component at file path
<code>transform</code>	Transforms input by imputing missing values. ‘None’ and np.nan values are treated as the same.

**clone** (*self*)

Constructs a new component with the same parameters and random state.

**Returns** A new instance of this component with identical parameters and random state.

**default\_parameters** (*cls*)

Returns the default parameters for this component.

Our convention is that `Component.default_parameters == Component().parameters`.

**Returns** default parameters for this component.

**Return type** dict

**describe** (*self*, *print\_name=False*, *return\_dict=False*)

Describe a component and its parameters

**Parameters**

- **print\_name** (*bool*, *optional*) – whether to print name of component
- **return\_dict** (*bool*, *optional*) – whether to return description as dictionary in the format {“name”: name, “parameters”: parameters}

**Returns** prints and returns dictionary

**Return type** None or dict

**fit** (*self*, *X*, *y=None*)

**Fits imputer to data. ‘None’ values are converted to np.nan before imputation and are** treated as the same.

**Parameters**

- **X** (*pd.DataFrame* or *np.ndarray*) – the input training data of shape [n\_samples, n\_features]
- **y** (*pd.Series*, *optional*) – the target training data of length [n\_samples]

**Returns** self

**fit\_transform** (*self*, *X*, *y=None*)

Fits on X and transforms X

**Parameters**

- **X** (*pd.DataFrame*) – Data to fit and transform
- **y** (*pd.Series*, *optional*) – Target data.

**Returns** Transformed X

**Return type** pd.DataFrame

**static load** (*file\_path*)

Loads component at file path

**Parameters** *file\_path* (*str*) – Location to load file

**Returns** ComponentBase object

**needs\_fitting** (*self*)

Returns boolean determining if component needs fitting before calling predict, predict\_proba, transform, or feature\_importances. This can be overridden to False for components that do not need to be fit or whose fit methods do nothing.

**property parameters** (*self*)

Returns the parameters which were used to initialize the component

**save** (*self*, *file\_path*, *pickle\_protocol*=cloudpickle.DEFAULT\_PROTOCOL)

Saves component at file path

**Parameters**

- **file\_path** (*str*) – Location to save file
- **pickle\_protocol** (*int*) – The pickle data stream format.

**Returns** None

**transform** (*self*, *X*, *y*=None)

Transforms input by imputing missing values. ‘None’ and np.nan values are treated as the same.

**Parameters**

- **X** (*pd.DataFrame*) – Data to transform
- **y** (*pd.Series*, *optional*) – Ignored.

**Returns** Transformed X

**Return type** pd.DataFrame

**class** evalml.pipelines.**SklearnStackedEnsembleClassifier** (*input\_pipelines*=None, *final\_estimator*=None, *cv*=None, *n\_jobs*=-1, *random\_seed*=0, *\*\*kwargs*)

Scikit-learn Stacked Ensemble Classifier.

**Parameters**

- **input\_pipelines** (*list* (*PipelineBase* or *subclass obj*)) – List of pipeline instances to use as the base estimators. This must not be None or an empty list or else EnsembleMissingPipelinesError will be raised.
- **final\_estimator** (*Estimator* or *subclass*) – The classifier used to combine the base estimators. If None, uses LogisticRegressionClassifier.
- **cv** (*int*, *cross-validation generator* or *an iterable*) – Determines the cross-validation splitting strategy used to train final\_estimator. For int/None inputs, if the estimator is a classifier and y is either binary or multiclass, StratifiedKFold is used. Defaults to None. Possible inputs for cv are:
  - None: 3-fold cross validation
  - int: the number of folds in a (Stratified) KFold
  - An scikit-learn cross-validation generator object
  - An iterable yielding (train, test) splits

- **n\_jobs** (*int* or *None*) – Non-negative integer describing level of parallelism used for pipelines. None and 1 are equivalent. If set to -1, all CPUs are used. For n\_jobs below -1, (n\_cpus + 1 + n\_jobs) are used. Defaults to -1. - Note: there could be some multi-process errors thrown for values of *n\_jobs*  $\neq 1$ . If this is the case, please use *n\_jobs* = 1.
- **random\_seed** (*int*) – Seed for the random number generator. Defaults to 0.

#### Attributes

<b>hyper-parameter_ranges</b>	{}
<b>model_family</b>	ModelFamily.ENSEMBLE
<b>modifies_features</b>	True
<b>modifies_target</b>	False
<b>name</b>	Sklearn Stacked Ensemble Classifier
<b>predict_uses_y</b>	False
<b>supported_problem_types</b>	[ ProblemTypes.BINARY, ProblemTypes.MULTICLASS, ProblemTypes.TIME_SERIES_BINARY, ProblemTypes.TIME_SERIES_MULTICLASS,]

#### Methods

<i>clone</i>	Constructs a new component with the same parameters and random state.
<i>default_parameters</i>	Returns the default parameters for stacked ensemble classes.
<i>describe</i>	Describe a component and its parameters
<i>feature_importance</i>	Not implemented for SklearnStackedEnsembleClassifier and SklearnStackedEnsembleRegressor
<i>fit</i>	Fits component to data
<i>load</i>	Loads component at file path
<i>needs_fitting</i>	Returns boolean determining if component needs fitting before
<i>parameters</i>	Returns the parameters which were used to initialize the component
<i>predict</i>	Make predictions using selected features.
<i>predict_proba</i>	Make probability estimates for labels.
<i>save</i>	Saves component at file path

#### **clone** (*self*)

Constructs a new component with the same parameters and random state.

**Returns** A new instance of this component with identical parameters and random state.

#### **default\_parameters** (*cls*)

Returns the default parameters for stacked ensemble classes.

**Returns** default parameters for this component.

**Return type** dict

#### **describe** (*self*, *print\_name=False*, *return\_dict=False*)

Describe a component and its parameters

**Parameters**

- **print\_name** (*bool, optional*) – whether to print name of component
- **return\_dict** (*bool, optional*) – whether to return description as dictionary in the format {"name": name, "parameters": parameters}

**Returns** prints and returns dictionary

**Return type** None or dict

**property feature\_importance** (*self*)

Not implemented for SklearnStackedEnsembleClassifier and SklearnStackedEnsembleRegressor

**fit** (*self, X, y=None*)

Fits component to data

**Parameters**

- **X** (*list, pd.DataFrame or np.ndarray*) – The input training data of shape [n\_samples, n\_features]
- **y** (*list, pd.Series, np.ndarray, optional*) – The target training data of length [n\_samples]

**Returns** self

**static load** (*file\_path*)

Loads component at file path

**Parameters** **file\_path** (*str*) – Location to load file

**Returns** ComponentBase object

**needs\_fitting** (*self*)

Returns boolean determining if component needs fitting before calling predict, predict\_proba, transform, or feature\_importances. This can be overridden to False for components that do not need to be fit or whose fit methods do nothing.

**property parameters** (*self*)

Returns the parameters which were used to initialize the component

**predict** (*self, X*)

Make predictions using selected features.

**Parameters** **X** (*pd.DataFrame, np.ndarray*) – Data of shape [n\_samples, n\_features]

**Returns** Predicted values

**Return type** pd.Series

**predict\_proba** (*self, X*)

Make probability estimates for labels.

**Parameters** **X** (*pd.DataFrame, or np.ndarray*) – Features

**Returns** Probability estimates

**Return type** pd.Series

**save** (*self, file\_path, pickle\_protocol=cloudpickle.DEFAULT\_PROTOCOL*)

Saves component at file path

**Parameters**

- **file\_path** (*str*) – Location to save file

- **pickle\_protocol** (*int*) – The pickle data stream format.

**Returns** None

```
class evalml.pipelines.SklearnStackedEnsembleRegressor (input_pipelines=None,  
                                                    final_estimator=None,  
                                                    cv=None, n_jobs=- 1, ran-  
                                                    dom_seed=0, **kwargs)
```

Scikit-learn Stacked Ensemble Regressor.

#### Parameters

- **input\_pipelines** (*list(PipelineBase or subclass obj)*) – List of pipeline instances to use as the base estimators. This must not be None or an empty list or else EnsembleMissingPipelinesError will be raised.
- **final\_estimator** (*Estimator or subclass*) – The regressor used to combine the base estimators. If None, uses LinearRegressor.
- **cv** (*int, cross-validation generator or an iterable*) – Determines the cross-validation splitting strategy used to train final\_estimator. For int/None inputs, KFold is used. Defaults to None. Possible inputs for cv are:
  - None: 3-fold cross validation
  - int: the number of folds in a (Stratified) KFold
  - An scikit-learn cross-validation generator object
  - An iterable yielding (train, test) splits
- **n\_jobs** (*int or None*) – Non-negative integer describing level of parallelism used for pipelines. None and 1 are equivalent. If set to -1, all CPUs are used. For n\_jobs below -1, (n\_cpus + 1 + n\_jobs) are used. Defaults to -1. - Note: there could be some multi-process errors thrown for values of *n\_jobs* != 1. If this is the case, please use *n\_jobs* = 1.
- **random\_seed** (*int*) – Seed for the random number generator. Defaults to 0.

#### Attributes

<b>hyper-parameter_ranges</b>	{}
<b>model_family</b>	ModelFamily.ENSEMBLE
<b>modifies_features</b>	True
<b>modifies_target</b>	False
<b>name</b>	Sklearn Stacked Ensemble Regressor
<b>pre-dict_uses_y</b>	False
<b>supported_problem_types</b>	[ ProblemTypes.REGRESSION, ProblemTypes.TIME_SERIES_REGRESSION,]

#### Methods

---

<i>clone</i>	Constructs a new component with the same parameters and random state.
--------------	---

---

continues on next page



Table 661 – continued from previous page

<code>default_parameters</code>	Returns the default parameters for stacked ensemble classes.
<code>describe</code>	Describe a component and its parameters
<code>feature_importance</code>	Not implemented for SklearnStackedEnsembleClassifier and SklearnStackedEnsembleRegressor
<code>fit</code>	Fits component to data
<code>load</code>	Loads component at file path
<code>needs_fitting</code>	Returns boolean determining if component needs fitting before
<code>parameters</code>	Returns the parameters which were used to initialize the component
<code>predict</code>	Make predictions using selected features.
<code>predict_proba</code>	Make probability estimates for labels.
<code>save</code>	Saves component at file path

**clone** (*self*)

Constructs a new component with the same parameters and random state.

**Returns** A new instance of this component with identical parameters and random state.

**default\_parameters** (*cls*)

Returns the default parameters for stacked ensemble classes.

**Returns** default parameters for this component.

**Return type** dict

**describe** (*self*, *print\_name=False*, *return\_dict=False*)

Describe a component and its parameters

**Parameters**

- **print\_name** (*bool*, *optional*) – whether to print name of component
- **return\_dict** (*bool*, *optional*) – whether to return description as dictionary in the format {"name": name, "parameters": parameters}

**Returns** prints and returns dictionary

**Return type** None or dict

**property feature\_importance** (*self*)

Not implemented for SklearnStackedEnsembleClassifier and SklearnStackedEnsembleRegressor

**fit** (*self*, *X*, *y=None*)

Fits component to data

**Parameters**

- **X** (*list*, *pd.DataFrame* or *np.ndarray*) – The input training data of shape [n\_samples, n\_features]
- **y** (*list*, *pd.Series*, *np.ndarray*, *optional*) – The target training data of length [n\_samples]

**Returns** self

**static load** (*file\_path*)

Loads component at file path

**Parameters** **file\_path** (*str*) – Location to load file

**Returns** ComponentBase object

**needs\_fitting** (*self*)

Returns boolean determining if component needs fitting before calling predict, predict\_proba, transform, or feature\_importances. This can be overridden to False for components that do not need to be fit or whose fit methods do nothing.

**property parameters** (*self*)

Returns the parameters which were used to initialize the component

**predict** (*self*, *X*)

Make predictions using selected features.

**Parameters** *X* (*pd.DataFrame*, *np.ndarray*) – Data of shape [n\_samples, n\_features]

**Returns** Predicted values

**Return type** *pd.Series*

**predict\_proba** (*self*, *X*)

Make probability estimates for labels.

**Parameters** *X* (*pd.DataFrame*, or *np.ndarray*) – Features

**Returns** Probability estimates

**Return type** *pd.Series*

**save** (*self*, *file\_path*, *pickle\_protocol=cloudpickle.DEFAULT\_PROTOCOL*)

Saves component at file path

**Parameters**

- **file\_path** (*str*) – Location to save file
- **pickle\_protocol** (*int*) – The pickle data stream format.

**Returns** None

**class** evalml.pipelines.**StandardScaler** (*random\_seed=0*, *\*\*kwargs*)

A transformer that standardizes input features by removing the mean and scaling to unit variance.

**Parameters** **random\_seed** (*int*) – Seed for the random number generator. Defaults to 0.

**Attributes**

hyper-parameter_ranges	{}
model_family	ModelFamily.NONE
modifies_features	True
modifies_target	False
name	Standard Scaler

**Methods**

<i>clone</i>	Constructs a new component with the same parameters and random state.
<i>default_parameters</i>	Returns the default parameters for this component.

continues on next page

Table 662 – continued from previous page

<i>describe</i>	Describe a component and its parameters
<i>fit</i>	Fits component to data
<i>fit_transform</i>	Fits on X and transforms X
<i>load</i>	Loads component at file path
<i>needs_fitting</i>	Returns boolean determining if component needs fitting before
<i>parameters</i>	Returns the parameters which were used to initialize the component
<i>save</i>	Saves component at file path
<i>transform</i>	Transforms data X.

**clone** (*self*)

Constructs a new component with the same parameters and random state.

**Returns** A new instance of this component with identical parameters and random state.

**default\_parameters** (*cls*)

Returns the default parameters for this component.

Our convention is that `Component.default_parameters == Component().parameters`.

**Returns** default parameters for this component.

**Return type** dict

**describe** (*self*, *print\_name=False*, *return\_dict=False*)

Describe a component and its parameters

**Parameters**

- **print\_name** (*bool*, *optional*) – whether to print name of component
- **return\_dict** (*bool*, *optional*) – whether to return description as dictionary in the format {"name": name, "parameters": parameters}

**Returns** prints and returns dictionary

**Return type** None or dict

**fit** (*self*, *X*, *y=None*)

Fits component to data

**Parameters**

- **X** (*list*, *pd.DataFrame* or *np.ndarray*) – The input training data of shape [n\_samples, n\_features]
- **y** (*list*, *pd.Series*, *np.ndarray*, *optional*) – The target training data of length [n\_samples]

**Returns** self

**fit\_transform** (*self*, *X*, *y=None*)

Fits on X and transforms X

**Parameters**

- **X** (*pd.DataFrame*) – Data to fit and transform
- **y** (*pd.Series*) – Target data

**Returns** Transformed X

**Return type** `pd.DataFrame`

**static load** (*file\_path*)

Loads component at file path

**Parameters** `file_path` (*str*) – Location to load file

**Returns** `ComponentBase` object

**needs\_fitting** (*self*)

Returns boolean determining if component needs fitting before calling `predict`, `predict_proba`, `transform`, or `feature_importances`. This can be overridden to `False` for components that do not need to be fit or whose fit methods do nothing.

**property parameters** (*self*)

Returns the parameters which were used to initialize the component

**save** (*self*, *file\_path*, *pickle\_protocol=cloudpickle.DEFAULT\_PROTOCOL*)

Saves component at file path

**Parameters**

- `file_path` (*str*) – Location to save file
- `pickle_protocol` (*int*) – The pickle data stream format.

**Returns** `None`

**transform** (*self*, *X*, *y=None*)

Transforms data *X*.

**Parameters**

- `X` (*pd.DataFrame*) – Data to transform.
- `y` (*pd.Series*, *optional*) – Target data.

**Returns** Transformed *X*

**Return type** `pd.DataFrame`

**class** `evalml.pipelines.SVMClassifier` (*C=1.0*, *kernel='rbf'*, *gamma='scale'*, *probability=True*, *random\_seed=0*, *\*\*kwargs*)

Support Vector Machine Classifier.

**Parameters**

- `C` (*float*) – The regularization parameter. The strength of the regularization is inversely proportional to *C*. Must be strictly positive. The penalty is a squared l2 penalty. Defaults to 1.0.
- `kernel` (*{ "linear", "poly", "rbf", "sigmoid", "precomputed" }*) – Specifies the kernel type to be used in the algorithm. Defaults to “rbf”.
- `gamma` (*{ "scale", "auto" } or float*) – Kernel coefficient for “rbf”, “poly” and “sigmoid”. Defaults to “scale”. - If `gamma='scale'` (default) is passed then it uses  $1 / (n\_features * X.var())$  as value of `gamma` - if “auto”, uses  $1 / n\_features$
- `probability` (*boolean*) – Whether to enable probability estimates. Defaults to `True`.
- `random_seed` (*int*) – Seed for the random number generator. Defaults to 0.

**Attributes**

<b>hyper-parameter_ranges</b>	{ "C": Real(0, 10), "kernel": ["linear", "poly", "rbf", "sigmoid", "precomputed"], "gamma": ["scale", "auto"], }
<b>model_family</b>	ModelFamily.SVM
<b>modifies_features</b>	True
<b>modifies_target</b>	False
<b>name</b>	SVM Classifier
<b>pre-dict_uses_y</b>	False
<b>supported_problem_types</b>	[ ProblemTypes.BINARY, ProblemTypes.MULTICLASS, ProblemTypes.TIME_SERIES_BINARY, ProblemTypes.TIME_SERIES_MULTICLASS, ]

## Methods

<i>clone</i>	Constructs a new component with the same parameters and random state.
<i>default_parameters</i>	Returns the default parameters for this component.
<i>describe</i>	Describe a component and its parameters
<i>feature_importance</i>	Feature importance only works with linear kernels.
<i>fit</i>	Fits component to data
<i>load</i>	Loads component at file path
<i>needs_fitting</i>	Returns boolean determining if component needs fitting before
<i>parameters</i>	Returns the parameters which were used to initialize the component
<i>predict</i>	Make predictions using selected features.
<i>predict_proba</i>	Make probability estimates for labels.
<i>save</i>	Saves component at file path

### **clone** (*self*)

Constructs a new component with the same parameters and random state.

**Returns** A new instance of this component with identical parameters and random state.

### **default\_parameters** (*cls*)

Returns the default parameters for this component.

Our convention is that `Component.default_parameters == Component().parameters`.

**Returns** default parameters for this component.

**Return type** dict

### **describe** (*self*, *print\_name=False*, *return\_dict=False*)

Describe a component and its parameters

#### Parameters

- **print\_name** (*bool*, *optional*) – whether to print name of component
- **return\_dict** (*bool*, *optional*) – whether to return description as dictionary in the format {"name": name, "parameters": parameters}

**Returns** prints and returns dictionary

**Return type** None or dict

**property feature\_importance** (*self*)

Feature importance only works with linear kernels. If the kernel isn't linear, we return a numpy array of zeros

**fit** (*self*, *X*, *y=None*)

Fits component to data

**Parameters**

- **X** (*list*, *pd.DataFrame* or *np.ndarray*) – The input training data of shape [n\_samples, n\_features]
- **y** (*list*, *pd.Series*, *np.ndarray*, *optional*) – The target training data of length [n\_samples]

**Returns** *self*

**static load** (*file\_path*)

Loads component at file path

**Parameters** **file\_path** (*str*) – Location to load file

**Returns** ComponentBase object

**needs\_fitting** (*self*)

Returns boolean determining if component needs fitting before calling predict, predict\_proba, transform, or feature\_importances. This can be overridden to False for components that do not need to be fit or whose fit methods do nothing.

**property parameters** (*self*)

Returns the parameters which were used to initialize the component

**predict** (*self*, *X*)

Make predictions using selected features.

**Parameters** **X** (*pd.DataFrame*, *np.ndarray*) – Data of shape [n\_samples, n\_features]

**Returns** Predicted values

**Return type** *pd.Series*

**predict\_proba** (*self*, *X*)

Make probability estimates for labels.

**Parameters** **X** (*pd.DataFrame*, or *np.ndarray*) – Features

**Returns** Probability estimates

**Return type** *pd.Series*

**save** (*self*, *file\_path*, *pickle\_protocol=cloudpickle.DEFAULT\_PROTOCOL*)

Saves component at file path

**Parameters**

- **file\_path** (*str*) – Location to save file
- **pickle\_protocol** (*int*) – The pickle data stream format.

**Returns** None

**class** evalml.pipelines.**SVMRegressor** (*C=1.0*, *kernel='rbf'*, *gamma='scale'*, *random\_seed=0*,  
\*\**kwargs*)

Support Vector Machine Regressor.

**Parameters**

- **C** (*float*) – The regularization parameter. The strength of the regularization is inversely proportional to C. Must be strictly positive. The penalty is a squared l2 penalty. Defaults to 1.0.
- **kernel** (*{ "linear", "poly", "rbf", "sigmoid", "precomputed" }*) – Specifies the kernel type to be used in the algorithm. Defaults to “rbf”.
- **gamma** (*{ "scale", "auto" } or float*) – Kernel coefficient for “rbf”, “poly” and “sigmoid”. Defaults to “scale”. - If gamma=’scale’ (default) is passed then it uses  $1 / (n\_features * X.var())$  as value of gamma - if “auto”, uses  $1 / n\_features$
- **random\_seed** (*int*) – Seed for the random number generator. Defaults to 0.

**Attributes**

<b>hyper-parameter_ranges</b>	{ “C”: Real(0, 10), “kernel”: [“linear”, “poly”, “rbf”, “sigmoid”, “precomputed”], “gamma”: [“scale”, “auto”], }
<b>model_family</b>	ModelFamily.SVM
<b>modifies_features</b>	True
<b>modifies_target</b>	False
<b>name</b>	SVM Regressor
<b>pre-dict_uses_y</b>	False
<b>supported_problem_types</b>	[ ProblemTypes.REGRESSION, ProblemTypes.TIME_SERIES_REGRESSION, ]

**Methods**

<i>clone</i>	Constructs a new component with the same parameters and random state.
<i>default_parameters</i>	Returns the default parameters for this component.
<i>describe</i>	Describe a component and its parameters
<i>feature_importance</i>	Feature importance only works with linear kernels.
<i>fit</i>	Fits component to data
<i>load</i>	Loads component at file path
<i>needs_fitting</i>	Returns boolean determining if component needs fitting before
<i>parameters</i>	Returns the parameters which were used to initialize the component
<i>predict</i>	Make predictions using selected features.
<i>predict_proba</i>	Make probability estimates for labels.
<i>save</i>	Saves component at file path

**clone** (*self*)

Constructs a new component with the same parameters and random state.

**Returns** A new instance of this component with identical parameters and random state.

**default\_parameters** (*cls*)

Returns the default parameters for this component.

Our convention is that `Component.default_parameters == Component().parameters`.

**Returns** default parameters for this component.

**Return type** dict

**describe** (*self*, *print\_name=False*, *return\_dict=False*)

Describe a component and its parameters

**Parameters**

- **print\_name** (*bool*, *optional*) – whether to print name of component
- **return\_dict** (*bool*, *optional*) – whether to return description as dictionary in the format {"name": name, "parameters": parameters}

**Returns** prints and returns dictionary

**Return type** None or dict

**property feature\_importance** (*self*)

Feature importance only works with linear kernels. If the kernel isn't linear, we return a numpy array of zeros

**fit** (*self*, *X*, *y=None*)

Fits component to data

**Parameters**

- **X** (*list*, *pd.DataFrame* or *np.ndarray*) – The input training data of shape [n\_samples, n\_features]
- **y** (*list*, *pd.Series*, *np.ndarray*, *optional*) – The target training data of length [n\_samples]

**Returns** self

**static load** (*file\_path*)

Loads component at file path

**Parameters** **file\_path** (*str*) – Location to load file

**Returns** ComponentBase object

**needs\_fitting** (*self*)

Returns boolean determining if component needs fitting before calling predict, predict\_proba, transform, or feature\_importances. This can be overridden to False for components that do not need to be fit or whose fit methods do nothing.

**property parameters** (*self*)

Returns the parameters which were used to initialize the component

**predict** (*self*, *X*)

Make predictions using selected features.

**Parameters** **X** (*pd.DataFrame*, *np.ndarray*) – Data of shape [n\_samples, n\_features]

**Returns** Predicted values

**Return type** pd.Series

**predict\_proba** (*self*, *X*)

Make probability estimates for labels.

**Parameters** **X** (*pd.DataFrame*, or *np.ndarray*) – Features

**Returns** Probability estimates



**Return type** `pd.Series`

**save** (*self*, *file\_path*, *pickle\_protocol*=`cloudpickle.DEFAULT_PROTOCOL`)  
Saves component at file path

**Parameters**

- **file\_path** (*str*) – Location to save file
- **pickle\_protocol** (*int*) – The pickle data stream format.

**Returns** `None`

**class** `evalml.pipelines.TargetEncoder` (*cols*=`None`, *smoothing*=`1.0`, *handle\_unknown*='value',  
*handle\_missing*='value', *random\_seed*=`0`, *\*\*kwargs*)

A transformer that encodes categorical features into target encodings.

**Parameters**

- **cols** (*list*) – Columns to encode. If `None`, all string columns will be encoded, otherwise only the columns provided will be encoded. Defaults to `None`
- **smoothing** (*float*) – The smoothing factor to apply. The larger this value is, the more influence the expected target value has on the resulting target encodings. Must be strictly larger than 0. Defaults to `1.0`
- **handle\_unknown** (*string*) – Determines how to handle unknown categories for a feature encountered. Options are 'value', 'error', and 'return\_nan'. Defaults to 'value', which replaces with the target mean
- **handle\_missing** (*string*) – Determines how to handle missing values encountered during *fit* or *transform*. Options are 'value', 'error', and 'return\_nan'. Defaults to 'value', which replaces with the target mean
- **random\_seed** (*int*) – Seed for the random number generator. Defaults to `0`.

**Attributes**

<b>hyper-parameter_ranges</b>	<code>{}</code>
<b>model_family</b>	<code>ModelFamily.NONE</code>
<b>modifies_features</b>	<code>True</code>
<b>modifies_target</b>	<code>False</code>
<b>name</b>	Target Encoder

**Methods**

<code>clone</code>	Constructs a new component with the same parameters and random state.
<code>default_parameters</code>	Returns the default parameters for this component.
<code>describe</code>	Describe a component and its parameters
<code>fit</code>	Fits component to data
<code>fit_transform</code>	Fits on X and transforms X
<code>get_feature_names</code>	Return feature names for the input features after fitting.

continues on next page

Table 665 – continued from previous page

<code>load</code>	Loads component at file path
<code>needs_fitting</code>	Returns boolean determining if component needs fitting before
<code>parameters</code>	Returns the parameters which were used to initialize the component
<code>save</code>	Saves component at file path
<code>transform</code>	Transforms data X.

**clone** (*self*)

Constructs a new component with the same parameters and random state.

**Returns** A new instance of this component with identical parameters and random state.

**default\_parameters** (*cls*)

Returns the default parameters for this component.

Our convention is that `Component.default_parameters == Component().parameters`.

**Returns** default parameters for this component.

**Return type** dict

**describe** (*self*, *print\_name=False*, *return\_dict=False*)

Describe a component and its parameters

**Parameters**

- **print\_name** (*bool*, *optional*) – whether to print name of component
- **return\_dict** (*bool*, *optional*) – whether to return description as dictionary in the format {"name": name, "parameters": parameters}

**Returns** prints and returns dictionary

**Return type** None or dict

**fit** (*self*, *X*, *y*)

Fits component to data

**Parameters**

- **X** (*list*, *pd.DataFrame* or *np.ndarray*) – The input training data of shape [n\_samples, n\_features]
- **y** (*list*, *pd.Series*, *np.ndarray*, *optional*) – The target training data of length [n\_samples]

**Returns** self

**fit\_transform** (*self*, *X*, *y*)

Fits on X and transforms X

**Parameters**

- **X** (*pd.DataFrame*) – Data to fit and transform
- **y** (*pd.Series*) – Target data

**Returns** Transformed X

**Return type** pd.DataFrame

**get\_feature\_names** (*self*)

Return feature names for the input features after fitting.

**Returns** The feature names after encoding

**Return type** np.array

**static load** (*file\_path*)

Loads component at file path

**Parameters** *file\_path* (*str*) – Location to load file

**Returns** ComponentBase object

**needs\_fitting** (*self*)

Returns boolean determining if component needs fitting before calling predict, predict\_proba, transform, or feature\_importances. This can be overridden to False for components that do not need to be fit or whose fit methods do nothing.

**property parameters** (*self*)

Returns the parameters which were used to initialize the component

**save** (*self*, *file\_path*, *pickle\_protocol*=cloudpickle.DEFAULT\_PROTOCOL)

Saves component at file path

**Parameters**

- **file\_path** (*str*) – Location to save file
- **pickle\_protocol** (*int*) – The pickle data stream format.

**Returns** None

**transform** (*self*, *X*, *y*=None)

Transforms data X.

**Parameters**

- **X** (*pd.DataFrame*) – Data to transform.
- **y** (*pd.Series*, *optional*) – Target data.

**Returns** Transformed X

**Return type** pd.DataFrame

```
class evalml.pipelines.TimeSeriesBinaryClassificationPipeline(component_graph,
                                                             param-
                                                             eters=None, cus-
                                                             tom_name=None,
                                                             random_seed=0)
```

Pipeline base class for time series binary classification problems.

**Parameters**

- **component\_graph** (*list or dict*) – List of components in order. Accepts strings or ComponentBase subclasses in the list. Note that when duplicate components are specified in a list, the duplicate component names will be modified with the component’s index in the list. For example, the component graph [Imputer, One Hot Encoder, Imputer, Logistic Regression Classifier] will have names [“Imputer”, “One Hot Encoder”, “Imputer\_2”, “Logistic Regression Classifier”]
- **parameters** (*dict*) – Dictionary with component names as keys and dictionary of that component’s parameters as values. An empty dictionary {} implies using all default values for component parameters. Pipeline-level parameters such as date\_index, gap, and max\_delay must be specified with the “pipeline” key. For example: Pipeline(parameters={“pipeline”: {“date\_index”: “Date”, “max\_delay”: 4, “gap”: 2}}).

- **random\_seed** (*int*) – Seed for the random number generator. Defaults to 0.

#### Attributes

<b>problem_type</b>	None
---------------------	------

#### Methods

<i>can_tune_threshold_with_objective</i>	Determine whether the threshold of a binary classification pipeline can be tuned.
<i>classes_</i>	Gets the class names for the problem.
<i>clone</i>	Constructs a new pipeline with the same components, parameters, and random state.
<i>compute_estimator_features</i>	Transforms the data by applying all pre-processing components.
<i>create_objectives</i>	
<i>custom_name</i>	Custom name of the pipeline.
<i>describe</i>	Outputs pipeline details including component parameters
<i>feature_importance</i>	Importance associated with each feature. Features dropped by the feature selection are excluded.
<i>fit</i>	Fit a time series classification pipeline.
<i>get_component</i>	Returns component by name
<i>get_hyperparameter_ranges</i>	Returns hyperparameter ranges from all components as a dictionary.
<i>graph</i>	Generate an image representing the pipeline graph.
<i>graph_feature_importance</i>	Generate a bar graph of the pipeline's feature importance
<i>inverse_transform</i>	Apply component <i>inverse_transform</i> methods to estimator predictions in reverse order.
<i>load</i>	Loads pipeline at file path
<i>model_family</i>	Returns model family of this pipeline.
<i>name</i>	Name of the pipeline.
<i>new</i>	Constructs a new instance of the pipeline with the same component graph but with a different set of parameters.
<i>optimize_threshold</i>	Optimize the pipeline threshold given the objective to use. Only used for binary problems with objectives whose thresholds can be tuned.
<i>parameters</i>	Parameter dictionary for this pipeline.
<i>predict</i>	Make predictions using selected features.
<i>predict_proba</i>	Make probability estimates for labels.
<i>save</i>	Saves pipeline at file path
<i>score</i>	Evaluate model performance on current and additional objectives.
<i>summary</i>	A short summary of the pipeline structure, describing the list of components used.
<i>threshold</i>	Threshold used to make a prediction. Defaults to None.

**can\_tune\_threshold\_with\_objective** (*self*, *objective*)

Determine whether the threshold of a binary classification pipeline can be tuned.

**Parameters**

- **pipeline** (*PipelineBase*) – Binary classification pipeline.
- **objective** – Primary AutoMLSearch objective.

**property classes\_** (*self*)

Gets the class names for the problem.

**clone** (*self*)

Constructs a new pipeline with the same components, parameters, and random state.

**Returns** A new instance of this pipeline with identical components, parameters, and random state.

**compute\_estimator\_features** (*self*, *X*, *y=None*)

Transforms the data by applying all pre-processing components.

**Parameters** **X** (*pd.DataFrame*) – Input data to the pipeline to transform.

**Returns** New transformed features.

**Return type** *pd.DataFrame*

**static create\_objectives** (*objectives*)

**property custom\_name** (*self*)

Custom name of the pipeline.

**describe** (*self*, *return\_dict=False*)

Outputs pipeline details including component parameters

**Parameters** **return\_dict** (*bool*) – If True, return dictionary of information about pipeline. Defaults to False.

**Returns** Dictionary of all component parameters if *return\_dict* is True, else None

**Return type** *dict*

**property feature\_importance** (*self*)

Importance associated with each feature. Features dropped by the feature selection are excluded.

**Returns** *pd.DataFrame* including feature names and their corresponding importance

**fit** (*self*, *X*, *y*)

Fit a time series classification pipeline.

**Parameters**

- **X** (*pd.DataFrame* or *np.ndarray*) – The input training data of shape [n\_samples, n\_features]
- **y** (*pd.Series*, *np.ndarray*) – The target training targets of length [n\_samples]

**Returns** *self*

**get\_component** (*self*, *name*)

Returns component by name

**Parameters** **name** (*str*) – Name of component

**Returns** Component to return

**Return type** *Component*

**get\_hyperparameter\_ranges** (*self*, *custom\_hyperparameters*)

Returns hyperparameter ranges from all components as a dictionary.

**Parameters** **custom\_hyperparameters** (*dict*) – Custom hyperparameters for the pipeline.

**Returns** Dictionary of hyperparameter ranges for each component in the pipeline.

**Return type** dict

**graph** (*self*, *filepath=None*)

Generate an image representing the pipeline graph.

**Parameters** **filepath** (*str*, *optional*) – Path to where the graph should be saved. If set to None (as by default), the graph will not be saved.

**Returns** Graph object that can be directly displayed in Jupyter notebooks.

**Return type** graphviz.Digraph

**graph\_feature\_importance** (*self*, *importance\_threshold=0*)

Generate a bar graph of the pipeline's feature importance

**Parameters** **importance\_threshold** (*float*, *optional*) – If provided, graph features with a permutation importance whose absolute value is larger than importance\_threshold. Defaults to zero.

**Returns** plotly.Figure, a bar graph showing features and their corresponding importance

**inverse\_transform** (*self*, *y*)

Apply component inverse\_transform methods to estimator predictions in reverse order.

Components that implement inverse\_transform are PolynomialDetrender, LabelEncoder (tbd).

**Parameters** **y** (*pd.Series*) – Final component features

**static load** (*file\_path*)

Loads pipeline at file path

**Parameters** **file\_path** (*str*) – location to load file

**Returns** PipelineBase object

**property model\_family** (*self*)

Returns model family of this pipeline.

**property name** (*self*)

Name of the pipeline.

**new** (*self*, *parameters*, *random\_seed=0*)

**Constructs a new instance of the pipeline with the same component graph but with a different set of parameters.**

Not to be confused with python's \_\_new\_\_ method.

**Parameters**

- **parameters** (*dict*) – Dictionary with component names as keys and dictionary of that component's parameters as values. An empty dictionary or None implies using all default values for component parameters. Defaults to None.
- **random\_seed** (*int*) – Seed for the random number generator. Defaults to 0.

**Returns** A new instance of this pipeline with identical components.

**optimize\_threshold** (*self*, *X*, *y*, *y\_pred\_proba*, *objective*)

Optimize the pipeline threshold given the objective to use. Only used for binary problems with objectives whose thresholds can be tuned.

**Parameters**

- **X** (*pd.DataFrame*) – Input features
- **y** (*pd.Series*) – Input target values
- **y\_pred\_proba** (*pd.Series*) – The predicted probabilities of the target outputted by the pipeline
- **objective** (*ObjectiveBase*) – The objective to threshold with. Must have a tunable threshold.

**property parameters** (*self*)

Parameter dictionary for this pipeline.

**Returns** Dictionary of all component parameters.

**Return type** dict

**predict** (*self*, *X*, *y=None*, *objective=None*)

Make predictions using selected features.

**Parameters**

- **X** (*pd.DataFrame*, or *np.ndarray*) – Data of shape [n\_samples, n\_features]
- **y** (*pd.Series*, *np.ndarray*, *None*) – The target training targets of length [n\_samples]
- **objective** (*Object* or *string*) – The objective to use to make predictions

**Returns** Predicted values.

**Return type** *pd.Series*

**predict\_proba** (*self*, *X*, *y=None*)

Make probability estimates for labels.

**Parameters** **X** (*pd.DataFrame* or *np.ndarray*) – Data of shape [n\_samples, n\_features]

**Returns** Probability estimates

**Return type** *pd.DataFrame*

**save** (*self*, *file\_path*, *pickle\_protocol=cloudpickle.DEFAULT\_PROTOCOL*)

Saves pipeline at file path

**Parameters**

- **file\_path** (*str*) – location to save file
- **pickle\_protocol** (*int*) – the pickle data stream format.

**Returns** None

**score** (*self*, *X*, *y*, *objectives*)

Evaluate model performance on current and additional objectives.

**Parameters**

- **X** (*pd.DataFrame* or *np.ndarray*) – Data of shape [n\_samples, n\_features]
- **y** (*pd.Series*) – True labels of length [n\_samples]

- **objectives** (*list*) – Non-empty list of objectives to score on

**Returns** Ordered dictionary of objective scores

**Return type** dict

**property summary** (*self*)

A short summary of the pipeline structure, describing the list of components used. Example: Logistic Regression Classifier w/ Simple Imputer + One Hot Encoder

**property threshold** (*self*)

Threshold used to make a prediction. Defaults to None.

```
class evalml.pipelines.TimeSeriesClassificationPipeline(component_graph, pa-  
                                                    rameters=None, cus-  
                                                    tom_name=None, ran-  
                                                    dom_seed=0)
```

Pipeline base class for time series classification problems.

#### Parameters

- **component\_graph** (*list or dict*) – List of components in order. Accepts strings or ComponentBase subclasses in the list. Note that when duplicate components are specified in a list, the duplicate component names will be modified with the component’s index in the list. For example, the component graph [Imputer, One Hot Encoder, Imputer, Logistic Regression Classifier] will have names [“Imputer”, “One Hot Encoder”, “Imputer\_2”, “Logistic Regression Classifier”]
- **parameters** (*dict*) – Dictionary with component names as keys and dictionary of that component’s parameters as values. An empty dictionary {} implies using all default values for component parameters. Pipeline-level parameters such as date\_index, gap, and max\_delay must be specified with the “pipeline” key. For example: Pipeline(parameters={“pipeline”: {“date\_index”: “Date”, “max\_delay”: 4, “gap”: 2}}).
- **random\_seed** (*int*) – Seed for the random number generator. Defaults to 0.

#### Attributes

<b>problem_type</b>	None
---------------------	------

#### Methods

<i>can_tune_threshold_with_objective</i>	Determine whether the threshold of a binary classification pipeline can be tuned.
<i>classes_</i>	Gets the class names for the problem.
<i>clone</i>	Constructs a new pipeline with the same components, parameters, and random state.
<i>compute_estimator_features</i>	Transforms the data by applying all pre-processing components.
<i>create_objectives</i>	
<i>custom_name</i>	Custom name of the pipeline.
<i>describe</i>	Outputs pipeline details including component parameters
<i>feature_importance</i>	Importance associated with each feature. Features dropped by the feature selection are excluded.

continues on next page



Table 667 – continued from previous page

<code>fit</code>	Fit a time series classification pipeline.
<code>get_component</code>	Returns component by name
<code>get_hyperparameter_ranges</code>	Returns hyperparameter ranges from all components as a dictionary.
<code>graph</code>	Generate an image representing the pipeline graph.
<code>graph_feature_importance</code>	Generate a bar graph of the pipeline’s feature importance
<code>inverse_transform</code>	Apply component <code>inverse_transform</code> methods to estimator predictions in reverse order.
<code>load</code>	Loads pipeline at file path
<code>model_family</code>	Returns model family of this pipeline.
<code>name</code>	Name of the pipeline.
<code>new</code>	Constructs a new instance of the pipeline with the same component graph but with a different set of parameters.
<code>parameters</code>	Parameter dictionary for this pipeline.
<code>predict</code>	Make predictions using selected features.
<code>predict_proba</code>	Make probability estimates for labels.
<code>save</code>	Saves pipeline at file path
<code>score</code>	Evaluate model performance on current and additional objectives.
<code>summary</code>	A short summary of the pipeline structure, describing the list of components used.

**can\_tune\_threshold\_with\_objective** (*self*, *objective*)

Determine whether the threshold of a binary classification pipeline can be tuned.

**Parameters**

- **pipeline** (`PipelineBase`) – Binary classification pipeline.
- **objective** – Primary `AutoMLSearch` objective.

**property classes\_** (*self*)

Gets the class names for the problem.

**clone** (*self*)

Constructs a new pipeline with the same components, parameters, and random state.

**Returns** A new instance of this pipeline with identical components, parameters, and random state.

**compute\_estimator\_features** (*self*, *X*, *y=None*)

Transforms the data by applying all pre-processing components.

**Parameters** **X** (`pd.DataFrame`) – Input data to the pipeline to transform.

**Returns** New transformed features.

**Return type** `pd.DataFrame`

**static create\_objectives** (*objectives*)

**property custom\_name** (*self*)

Custom name of the pipeline.

**describe** (*self*, *return\_dict=False*)

Outputs pipeline details including component parameters

**Parameters** `return_dict` (*bool*) – If True, return dictionary of information about pipeline. Defaults to False.

**Returns** Dictionary of all component parameters if `return_dict` is True, else None

**Return type** dict

**property** `feature_importance` (*self*)

Importance associated with each feature. Features dropped by the feature selection are excluded.

**Returns** `pd.DataFrame` including feature names and their corresponding importance

**fit** (*self*, *X*, *y*)

Fit a time series classification pipeline.

**Parameters**

- **X** (*pd.DataFrame* or *np.ndarray*) – The input training data of shape [n\_samples, n\_features]
- **y** (*pd.Series*, *np.ndarray*) – The target training targets of length [n\_samples]

**Returns** *self*

**get\_component** (*self*, *name*)

Returns component by name

**Parameters** **name** (*str*) – Name of component

**Returns** Component to return

**Return type** Component

**get\_hyperparameter\_ranges** (*self*, *custom\_hyperparameters*)

Returns hyperparameter ranges from all components as a dictionary.

**Parameters** **custom\_hyperparameters** (*dict*) – Custom hyperparameters for the pipeline.

**Returns** Dictionary of hyperparameter ranges for each component in the pipeline.

**Return type** dict

**graph** (*self*, *filepath=None*)

Generate an image representing the pipeline graph.

**Parameters** **filepath** (*str*, *optional*) – Path to where the graph should be saved. If set to None (as by default), the graph will not be saved.

**Returns** Graph object that can be directly displayed in Jupyter notebooks.

**Return type** `graphviz.Digraph`

**graph\_feature\_importance** (*self*, *importance\_threshold=0*)

Generate a bar graph of the pipeline's feature importance

**Parameters** **importance\_threshold** (*float*, *optional*) – If provided, graph features with a permutation importance whose absolute value is larger than `importance_threshold`. Defaults to zero.

**Returns** `plotly.Figure`, a bar graph showing features and their corresponding importance

**inverse\_transform** (*self*, *y*)

Apply component `inverse_transform` methods to estimator predictions in reverse order.

Components that implement `inverse_transform` are `PolynomialDetrender`, `LabelEncoder` (`tbd`).

**Parameters** *y* (*pd.Series*) – Final component features

**static load** (*file\_path*)

Loads pipeline at file path

**Parameters** *file\_path* (*str*) – location to load file

**Returns** PipelineBase object

**property model\_family** (*self*)

Returns model family of this pipeline.

**property name** (*self*)

Name of the pipeline.

**new** (*self*, *parameters*, *random\_seed=0*)

**Constructs a new instance of the pipeline with the same component graph but with a different set of parameters.**

Not to be confused with python's `__new__` method.

**Parameters**

- **parameters** (*dict*) – Dictionary with component names as keys and dictionary of that component's parameters as values. An empty dictionary or None implies using all default values for component parameters. Defaults to None.
- **random\_seed** (*int*) – Seed for the random number generator. Defaults to 0.

**Returns** A new instance of this pipeline with identical components.

**property parameters** (*self*)

Parameter dictionary for this pipeline.

**Returns** Dictionary of all component parameters.

**Return type** dict

**predict** (*self*, *X*, *y=None*, *objective=None*)

Make predictions using selected features.

**Parameters**

- **X** (*pd.DataFrame*, or *np.ndarray*) – Data of shape [n\_samples, n\_features]
- **y** (*pd.Series*, *np.ndarray*, *None*) – The target training targets of length [n\_samples]
- **objective** (*Object* or *string*) – The objective to use to make predictions

**Returns** Predicted values.

**Return type** pd.Series

**predict\_proba** (*self*, *X*, *y=None*)

Make probability estimates for labels.

**Parameters** *X* (*pd.DataFrame* or *np.ndarray*) – Data of shape [n\_samples, n\_features]

**Returns** Probability estimates

**Return type** pd.DataFrame

**save** (*self*, *file\_path*, *pickle\_protocol=cloudpickle.DEFAULT\_PROTOCOL*)

Saves pipeline at file path

**Parameters**

- **file\_path** (*str*) – location to save file
- **pickle\_protocol** (*int*) – the pickle data stream format.

**Returns** None**score** (*self, X, y, objectives*)

Evaluate model performance on current and additional objectives.

**Parameters**

- **X** (*pd.DataFrame or np.ndarray*) – Data of shape [n\_samples, n\_features]
- **y** (*pd.Series*) – True labels of length [n\_samples]
- **objectives** (*list*) – Non-empty list of objectives to score on

**Returns** Ordered dictionary of objective scores**Return type** dict**property summary** (*self*)

A short summary of the pipeline structure, describing the list of components used. Example: Logistic Regression Classifier w/ Simple Imputer + One Hot Encoder

```
class evalml.pipelines.TimeSeriesMulticlassClassificationPipeline(component_graph,
                                                                param-
                                                                eters=None,
                                                                cus-
                                                                tom_name=None,
                                                                ran-
                                                                dom_seed=0)
```

Pipeline base class for time series multiclass classification problems.

**Parameters**

- **component\_graph** (*list or dict*) – List of components in order. Accepts strings or ComponentBase subclasses in the list. Note that when duplicate components are specified in a list, the duplicate component names will be modified with the component’s index in the list. For example, the component graph [Imputer, One Hot Encoder, Imputer, Logistic Regression Classifier] will have names [“Imputer”, “One Hot Encoder”, “Imputer\_2”, “Logistic Regression Classifier”]
- **parameters** (*dict*) – Dictionary with component names as keys and dictionary of that component’s parameters as values. An empty dictionary {} implies using all default values for component parameters. Pipeline-level parameters such as date\_index, gap, and max\_delay must be specified with the “pipeline” key. For example: Pipeline(parameters={“pipeline”: {“date\_index”: “Date”, “max\_delay”: 4, “gap”: 2}}).
- **random\_seed** (*int*) – Seed for the random number generator. Defaults to 0.

**Attributes**

<b>problem_type</b>	ProblemTypes.TIME_SERIES_MULTICLASS
---------------------	-------------------------------------

**Methods**

<code>can_tune_threshold_with_objective</code>	Determine whether the threshold of a binary classification pipeline can be tuned.
<code>classes_</code>	Gets the class names for the problem.
<code>clone</code>	Constructs a new pipeline with the same components, parameters, and random state.
<code>compute_estimator_features</code>	Transforms the data by applying all pre-processing components.
<code>create_objectives</code>	
<code>custom_name</code>	Custom name of the pipeline.
<code>describe</code>	Outputs pipeline details including component parameters
<code>feature_importance</code>	Importance associated with each feature. Features dropped by the feature selection are excluded.
<code>fit</code>	Fit a time series classification pipeline.
<code>get_component</code>	Returns component by name
<code>get_hyperparameter_ranges</code>	Returns hyperparameter ranges from all components as a dictionary.
<code>graph</code>	Generate an image representing the pipeline graph.
<code>graph_feature_importance</code>	Generate a bar graph of the pipeline's feature importance
<code>inverse_transform</code>	Apply component <code>inverse_transform</code> methods to estimator predictions in reverse order.
<code>load</code>	Loads pipeline at file path
<code>model_family</code>	Returns model family of this pipeline.
<code>name</code>	Name of the pipeline.
<code>new</code>	Constructs a new instance of the pipeline with the same component graph but with a different set of parameters.
<code>parameters</code>	Parameter dictionary for this pipeline.
<code>predict</code>	Make predictions using selected features.
<code>predict_proba</code>	Make probability estimates for labels.
<code>save</code>	Saves pipeline at file path
<code>score</code>	Evaluate model performance on current and additional objectives.
<code>summary</code>	A short summary of the pipeline structure, describing the list of components used.

**`can_tune_threshold_with_objective`** (*self*, *objective*)

Determine whether the threshold of a binary classification pipeline can be tuned.

#### Parameters

- **`pipeline`** (`PipelineBase`) – Binary classification pipeline.
- **`objective`** – Primary `AutoMLSearch` objective.

**`property classes_`** (*self*)

Gets the class names for the problem.

**`clone`** (*self*)

Constructs a new pipeline with the same components, parameters, and random state.

**Returns** A new instance of this pipeline with identical components, parameters, and random state.

**compute\_estimator\_features** (*self*, *X*, *y=None*)

Transforms the data by applying all pre-processing components.

**Parameters** **X** (*pd.DataFrame*) – Input data to the pipeline to transform.

**Returns** New transformed features.

**Return type** *pd.DataFrame*

**static create\_objectives** (*objectives*)

**property custom\_name** (*self*)

Custom name of the pipeline.

**describe** (*self*, *return\_dict=False*)

Outputs pipeline details including component parameters

**Parameters** **return\_dict** (*bool*) – If True, return dictionary of information about pipeline.  
Defaults to False.

**Returns** Dictionary of all component parameters if *return\_dict* is True, else None

**Return type** *dict*

**property feature\_importance** (*self*)

Importance associated with each feature. Features dropped by the feature selection are excluded.

**Returns** *pd.DataFrame* including feature names and their corresponding importance

**fit** (*self*, *X*, *y*)

Fit a time series classification pipeline.

**Parameters**

- **X** (*pd.DataFrame* or *np.ndarray*) – The input training data of shape [n\_samples, n\_features]
- **y** (*pd.Series*, *np.ndarray*) – The target training targets of length [n\_samples]

**Returns** *self*

**get\_component** (*self*, *name*)

Returns component by name

**Parameters** **name** (*str*) – Name of component

**Returns** Component to return

**Return type** Component

**get\_hyperparameter\_ranges** (*self*, *custom\_hyperparameters*)

Returns hyperparameter ranges from all components as a dictionary.

**Parameters** **custom\_hyperparameters** (*dict*) – Custom hyperparameters for the pipeline.

**Returns** Dictionary of hyperparameter ranges for each component in the pipeline.

**Return type** *dict*

**graph** (*self*, *filepath=None*)

Generate an image representing the pipeline graph.

**Parameters** **filepath** (*str*, *optional*) – Path to where the graph should be saved. If set to None (as by default), the graph will not be saved.

**Returns** Graph object that can be directly displayed in Jupyter notebooks.

**Return type** graphviz.Digraph

**graph\_feature\_importance** (*self*, *importance\_threshold=0*)

Generate a bar graph of the pipeline's feature importance

**Parameters** **importance\_threshold** (*float*, *optional*) – If provided, graph features with a permutation importance whose absolute value is larger than importance\_threshold. Defaults to zero.

**Returns** plotly.Figure, a bar graph showing features and their corresponding importance

**inverse\_transform** (*self*, *y*)

Apply component inverse\_transform methods to estimator predictions in reverse order.

Components that implement inverse\_transform are PolynomialDetrender, LabelEncoder (tbd).

**Parameters** **y** (*pd.Series*) – Final component features

**static load** (*file\_path*)

Loads pipeline at file path

**Parameters** **file\_path** (*str*) – location to load file

**Returns** PipelineBase object

**property model\_family** (*self*)

Returns model family of this pipeline.

**property name** (*self*)

Name of the pipeline.

**new** (*self*, *parameters*, *random\_seed=0*)

**Constructs a new instance of the pipeline with the same component graph but with a different set of parameters.**

Not to be confused with python's \_\_new\_\_ method.

**Parameters**

- **parameters** (*dict*) – Dictionary with component names as keys and dictionary of that component's parameters as values. An empty dictionary or None implies using all default values for component parameters. Defaults to None.
- **random\_seed** (*int*) – Seed for the random number generator. Defaults to 0.

**Returns** A new instance of this pipeline with identical components.

**property parameters** (*self*)

Parameter dictionary for this pipeline.

**Returns** Dictionary of all component parameters.

**Return type** dict

**predict** (*self*, *X*, *y=None*, *objective=None*)

Make predictions using selected features.

**Parameters**

- **X** (*pd.DataFrame*, or *np.ndarray*) – Data of shape [n\_samples, n\_features]
- **y** (*pd.Series*, *np.ndarray*, *None*) – The target training targets of length [n\_samples]
- **objective** (*Object* or *string*) – The objective to use to make predictions

**Returns** Predicted values.

**Return type** `pd.Series`

**predict\_proba** (*self*, *X*, *y=None*)

Make probability estimates for labels.

**Parameters** *X* (`pd.DataFrame` or `np.ndarray`) – Data of shape `[n_samples, n_features]`

**Returns** Probability estimates

**Return type** `pd.DataFrame`

**save** (*self*, *file\_path*, *pickle\_protocol=cloudpickle.DEFAULT\_PROTOCOL*)

Saves pipeline at file path

**Parameters**

- **file\_path** (*str*) – location to save file
- **pickle\_protocol** (*int*) – the pickle data stream format.

**Returns** `None`

**score** (*self*, *X*, *y*, *objectives*)

Evaluate model performance on current and additional objectives.

**Parameters**

- **X** (`pd.DataFrame` or `np.ndarray`) – Data of shape `[n_samples, n_features]`
- **y** (`pd.Series`) – True labels of length `[n_samples]`
- **objectives** (*list*) – Non-empty list of objectives to score on

**Returns** Ordered dictionary of objective scores

**Return type** `dict`

**property summary** (*self*)

A short summary of the pipeline structure, describing the list of components used. Example: Logistic Regression Classifier w/ Simple Imputer + One Hot Encoder

**class** `evalml.pipelines.TimeSeriesRegressionPipeline` (*component\_graph*, *parameters=None*, *custom\_name=None*, *random\_seed=0*)

Pipeline base class for time series regression problems.

**Parameters**

- **component\_graph** (*list* or *dict*) – List of components in order. Accepts strings or `ComponentBase` subclasses in the list. Note that when duplicate components are specified in a list, the duplicate component names will be modified with the component’s index in the list. For example, the component graph [Imputer, One Hot Encoder, Imputer, Logistic Regression Classifier] will have names [“Imputer”, “One Hot Encoder”, “Imputer\_2”, “Logistic Regression Classifier”]
- **parameters** (*dict*) – Dictionary with component names as keys and dictionary of that component’s parameters as values. An empty dictionary `{}` implies using all default values for component parameters. Pipeline-level parameters such as `date_index`, `gap`, and `max_delay` must be specified with the “pipeline” key. For example: `Pipeline(parameters={"pipeline": {"date_index": "Date", "max_delay": 4, "gap": 2}})`.
- **random\_seed** (*int*) – Seed for the random number generator. Defaults to 0.



**Attributes**

<b>problem_type</b>	ProblemTypes.TIME_SERIES_REGRESSIO
---------------------	------------------------------------

**Methods**

<i>can_tune_threshold_with_objective</i>	Determine whether the threshold of a binary classification pipeline can be tuned.
<i>clone</i>	Constructs a new pipeline with the same components, parameters, and random state.
<i>compute_estimator_features</i>	Transforms the data by applying all pre-processing components.
<i>create_objectives</i>	
<i>custom_name</i>	Custom name of the pipeline.
<i>describe</i>	Outputs pipeline details including component parameters
<i>feature_importance</i>	Importance associated with each feature. Features dropped by the feature selection are excluded.
<i>fit</i>	Fit a time series regression pipeline.
<i>get_component</i>	Returns component by name
<i>get_hyperparameter_ranges</i>	Returns hyperparameter ranges from all components as a dictionary.
<i>graph</i>	Generate an image representing the pipeline graph.
<i>graph_feature_importance</i>	Generate a bar graph of the pipeline's feature importance
<i>inverse_transform</i>	Apply component <code>inverse_transform</code> methods to estimator predictions in reverse order.
<i>load</i>	Loads pipeline at file path
<i>model_family</i>	Returns model family of this pipeline.
<i>name</i>	Name of the pipeline.
<i>new</i>	Constructs a new instance of the pipeline with the same component graph but with a different set of parameters.
<i>parameters</i>	Parameter dictionary for this pipeline.
<i>predict</i>	Make predictions using selected features.
<i>save</i>	Saves pipeline at file path
<i>score</i>	Evaluate model performance on current and additional objectives.
<i>summary</i>	A short summary of the pipeline structure, describing the list of components used.

**can\_tune\_threshold\_with\_objective** (*self*, *objective*)

Determine whether the threshold of a binary classification pipeline can be tuned.

**Parameters**

- **pipeline** (`PipelineBase`) – Binary classification pipeline.
- **objective** – Primary `AutoMLSearch` objective.

**clone** (*self*)

Constructs a new pipeline with the same components, parameters, and random state.

**Returns** A new instance of this pipeline with identical components, parameters, and random state.

**compute\_estimator\_features** (*self*, *X*, *y=None*)

Transforms the data by applying all pre-processing components.

**Parameters** *X* (*pd.DataFrame*) – Input data to the pipeline to transform.

**Returns** New transformed features.

**Return type** *pd.DataFrame*

**static create\_objectives** (*objectives*)

**property custom\_name** (*self*)

Custom name of the pipeline.

**describe** (*self*, *return\_dict=False*)

Outputs pipeline details including component parameters

**Parameters** *return\_dict* (*bool*) – If True, return dictionary of information about pipeline. Defaults to False.

**Returns** Dictionary of all component parameters if *return\_dict* is True, else None

**Return type** *dict*

**property feature\_importance** (*self*)

Importance associated with each feature. Features dropped by the feature selection are excluded.

**Returns** *pd.DataFrame* including feature names and their corresponding importance

**fit** (*self*, *X*, *y*)

Fit a time series regression pipeline.

**Parameters**

- *X* (*pd.DataFrame* or *np.ndarray*) – The input training data of shape [n\_samples, n\_features]
- *y* (*pd.Series*, *np.ndarray*) – The target training targets of length [n\_samples]

**Returns** *self*

**get\_component** (*self*, *name*)

Returns component by name

**Parameters** *name* (*str*) – Name of component

**Returns** Component to return

**Return type** *Component*

**get\_hyperparameter\_ranges** (*self*, *custom\_hyperparameters*)

Returns hyperparameter ranges from all components as a dictionary.

**Parameters** *custom\_hyperparameters* (*dict*) – Custom hyperparameters for the pipeline.

**Returns** Dictionary of hyperparameter ranges for each component in the pipeline.

**Return type** *dict*

**graph** (*self*, *filepath=None*)

Generate an image representing the pipeline graph.

**Parameters** `filepath` (*str*, *optional*) – Path to where the graph should be saved. If set to None (as by default), the graph will not be saved.

**Returns** Graph object that can be directly displayed in Jupyter notebooks.

**Return type** `graphviz.Digraph`

**graph\_feature\_importance** (*self*, *importance\_threshold=0*)

Generate a bar graph of the pipeline's feature importance

**Parameters** `importance_threshold` (*float*, *optional*) – If provided, graph features with a permutation importance whose absolute value is larger than `importance_threshold`. Defaults to zero.

**Returns** `plotly.Figure`, a bar graph showing features and their corresponding importance

**inverse\_transform** (*self*, *y*)

Apply component `inverse_transform` methods to estimator predictions in reverse order.

Components that implement `inverse_transform` are `PolynomialDetrender`, `LabelEncoder` (tbd).

**Parameters** `y` (*pd.Series*) – Final component features

**static load** (*file\_path*)

Loads pipeline at file path

**Parameters** `file_path` (*str*) – location to load file

**Returns** `PipelineBase` object

**property model\_family** (*self*)

Returns model family of this pipeline.

**property name** (*self*)

Name of the pipeline.

**new** (*self*, *parameters*, *random\_seed=0*)

**Constructs a new instance of the pipeline with the same component graph but with a different set of parameters.**

Not to be confused with python's `__new__` method.

**Parameters**

- **parameters** (*dict*) – Dictionary with component names as keys and dictionary of that component's parameters as values. An empty dictionary or None implies using all default values for component parameters. Defaults to None.
- **random\_seed** (*int*) – Seed for the random number generator. Defaults to 0.

**Returns** A new instance of this pipeline with identical components.

**property parameters** (*self*)

Parameter dictionary for this pipeline.

**Returns** Dictionary of all component parameters.

**Return type** `dict`

**predict** (*self*, *X*, *y=None*, *objective=None*)

Make predictions using selected features.

**Parameters**

- **X** (*pd.DataFrame*, or *np.ndarray*) – Data of shape `[n_samples, n_features]`

- **y** (*pd.Series, np.ndarray, None*) – The target training targets of length [n\_samples]
- **objective** (*Object or string*) – The objective to use to make predictions

**Returns** Predicted values.

**Return type** `pd.Series`

**save** (*self, file\_path, pickle\_protocol=cloudpickle.DEFAULT\_PROTOCOL*)

Saves pipeline at file path

**Parameters**

- **file\_path** (*str*) – location to save file
- **pickle\_protocol** (*int*) – the pickle data stream format.

**Returns** `None`

**score** (*self, X, y, objectives*)

Evaluate model performance on current and additional objectives.

**Parameters**

- **X** (*pd.DataFrame or np.ndarray*) – Data of shape [n\_samples, n\_features]
- **y** (*pd.Series*) – True labels of length [n\_samples]
- **objectives** (*list*) – Non-empty list of objectives to score on

**Returns** Ordered dictionary of objective scores

**Return type** `dict`

**property summary** (*self*)

A short summary of the pipeline structure, describing the list of components used. Example: Logistic Regression Classifier w/ Simple Imputer + One Hot Encoder

**class** `evalml.pipelines.Transformer` (*parameters=None, component\_obj=None, random\_seed=0, \*\*kwargs*)

A component that may or may not need fitting that transforms data. These components are used before an estimator.

To implement a new Transformer, define your own class which is a subclass of Transformer, including a name and a list of acceptable ranges for any parameters to be tuned during the automl search (hyperparameters). Define an `__init__` method which sets up any necessary state and objects. Make sure your `__init__` only uses standard keyword arguments and calls `super().__init__()` with a parameters dict. You may also override the `fit`, `transform`, `fit_transform` and other methods in this class if appropriate.

To see some examples, check out the definitions of any Transformer component.

**Parameters**

- **parameters** (*dict*) – Dictionary of parameters for the component. Defaults to `None`.
- **component\_obj** (*obj*) – Third-party objects useful in component implementation. Defaults to `None`.
- **random\_seed** (*int*) – Seed for the random number generator. Defaults to 0.

**Attributes**

<b>model_family</b>	ModelFamily.NONE
<b>modifies_features</b>	True
<b>modifies_target</b>	False

## Methods

<i>clone</i>	Constructs a new component with the same parameters and random state.
<i>default_parameters</i>	Returns the default parameters for this component.
<i>describe</i>	Describe a component and its parameters
<i>fit</i>	Fits component to data
<i>fit_transform</i>	Fits on X and transforms X
<i>load</i>	Loads component at file path
<i>name</i>	Returns string name of this component
<i>needs_fitting</i>	Returns boolean determining if component needs fitting before
<i>parameters</i>	Returns the parameters which were used to initialize the component
<i>save</i>	Saves component at file path
<i>transform</i>	Transforms data X.

### **clone** (*self*)

Constructs a new component with the same parameters and random state.

**Returns** A new instance of this component with identical parameters and random state.

### **default\_parameters** (*cls*)

Returns the default parameters for this component.

Our convention is that `Component.default_parameters == Component().parameters`.

**Returns** default parameters for this component.

**Return type** dict

### **describe** (*self*, *print\_name=False*, *return\_dict=False*)

Describe a component and its parameters

#### Parameters

- **print\_name** (*bool*, *optional*) – whether to print name of component
- **return\_dict** (*bool*, *optional*) – whether to return description as dictionary in the format {"name": name, "parameters": parameters}

**Returns** prints and returns dictionary

**Return type** None or dict

### **fit** (*self*, *X*, *y=None*)

Fits component to data

#### Parameters

- **X** (*list*, *pd.DataFrame* or *np.ndarray*) – The input training data of shape [n\_samples, n\_features]

- **y** (*list, pd.Series, np.ndarray, optional*) – The target training data of length [n\_samples]

**Returns** self

**fit\_transform** (*self, X, y=None*)

Fits on X and transforms X

**Parameters**

- **X** (*pd.DataFrame*) – Data to fit and transform
- **y** (*pd.Series*) – Target data

**Returns** Transformed X

**Return type** pd.DataFrame

**static load** (*file\_path*)

Loads component at file path

**Parameters** **file\_path** (*str*) – Location to load file

**Returns** ComponentBase object

**property name** (*cls*)

Returns string name of this component

**needs\_fitting** (*self*)

Returns boolean determining if component needs fitting before calling predict, predict\_proba, transform, or feature\_importances. This can be overridden to False for components that do not need to be fit or whose fit methods do nothing.

**property parameters** (*self*)

Returns the parameters which were used to initialize the component

**save** (*self, file\_path, pickle\_protocol=cloudpickle.DEFAULT\_PROTOCOL*)

Saves component at file path

**Parameters**

- **file\_path** (*str*) – Location to save file
- **pickle\_protocol** (*int*) – The pickle data stream format.

**Returns** None

**transform** (*self, X, y=None*)

Transforms data X.

**Parameters**

- **X** (*pd.DataFrame*) – Data to transform.
- **y** (*pd.Series, optional*) – Target data.

**Returns** Transformed X

**Return type** pd.DataFrame

**class** evalml.pipelines.XGBoostClassifier (*eta=0.1, max\_depth=6, min\_child\_weight=1, n\_estimators=100, random\_seed=0, n\_jobs=-1, \*\*kwargs*)

XGBoost Classifier.

**Parameters**

- **eta** (*float*) – Boosting learning rate. Defaults to 0.1.

- **max\_depth** (*int*) – Maximum tree depth for base learners. Defaults to 6.
- **min\_child\_weight** (*float*) – Minimum sum of instance weight (hessian) needed in a child. Defaults to 1.0
- **n\_estimators** (*int*) – Number of gradient boosted trees. Equivalent to number of boosting rounds. Defaults to 100.
- **random\_seed** (*int*) – Seed for the random number generator. Defaults to 0.
- **n\_jobs** (*int*) – Number of parallel threads used to run xgboost. Note that creating thread contention will significantly slow down the algorithm. Defaults to -1.

#### Attributes

<b>hyper-parameter_ranges</b>	{ “eta”: Real(0.000001, 1), “max_depth”: Integer(1, 10), “min_child_weight”: Real(1, 10), “n_estimators”: Integer(1, 1000), }
<b>model_family</b>	ModelFamily.XGBOOST
<b>modifies_features</b>	True
<b>modifies_target</b>	False
<b>name</b>	XGBoost Classifier
<b>predict_uses_y</b>	False
<b>SEED_MAX</b>	None
<b>SEED_MIN</b>	None
<b>supported_problem_types</b>	[ ProblemTypes.BINARY, ProblemTypes.MULTICLASS, ProblemTypes.TIME_SERIES_BINARY, ProblemTypes.TIME_SERIES_MULTICLASS, ]

#### Methods

<i>clone</i>	Constructs a new component with the same parameters and random state.
<i>default_parameters</i>	Returns the default parameters for this component.
<i>describe</i>	Describe a component and its parameters
<i>feature_importance</i>	Returns importance associated with each feature.
<i>fit</i>	Fits component to data
<i>load</i>	Loads component at file path
<i>needs_fitting</i>	Returns boolean determining if component needs fitting before
<i>parameters</i>	Returns the parameters which were used to initialize the component
<i>predict</i>	Make predictions using selected features.
<i>predict_proba</i>	Make probability estimates for labels.
<i>save</i>	Saves component at file path

#### **clone** (*self*)

Constructs a new component with the same parameters and random state.

**Returns** A new instance of this component with identical parameters and random state.

#### **default\_parameters** (*cls*)

Returns the default parameters for this component.

Our convention is that `Component.default_parameters == Component().parameters`.

**Returns** default parameters for this component.

**Return type** dict

**describe** (*self*, *print\_name=False*, *return\_dict=False*)

Describe a component and its parameters

**Parameters**

- **print\_name** (*bool*, *optional*) – whether to print name of component
- **return\_dict** (*bool*, *optional*) – whether to return description as dictionary in the format {"name": name, "parameters": parameters}

**Returns** prints and returns dictionary

**Return type** None or dict

**property feature\_importance** (*self*)

Returns importance associated with each feature.

**Returns** Importance associated with each feature

**Return type** np.ndarray

**fit** (*self*, *X*, *y=None*)

Fits component to data

**Parameters**

- **X** (*list*, *pd.DataFrame* or *np.ndarray*) – The input training data of shape [n\_samples, n\_features]
- **y** (*list*, *pd.Series*, *np.ndarray*, *optional*) – The target training data of length [n\_samples]

**Returns** self

**static load** (*file\_path*)

Loads component at file path

**Parameters** **file\_path** (*str*) – Location to load file

**Returns** ComponentBase object

**needs\_fitting** (*self*)

Returns boolean determining if component needs fitting before calling predict, predict\_proba, transform, or feature\_importances. This can be overridden to False for components that do not need to be fit or whose fit methods do nothing.

**property parameters** (*self*)

Returns the parameters which were used to initialize the component

**predict** (*self*, *X*)

Make predictions using selected features.

**Parameters** **X** (*pd.DataFrame*, *np.ndarray*) – Data of shape [n\_samples, n\_features]

**Returns** Predicted values

**Return type** pd.Series

**predict\_proba** (*self*, *X*)

Make probability estimates for labels.



**Parameters** **X** (*pd.DataFrame*, or *np.ndarray*) – Features

**Returns** Probability estimates

**Return type** *pd.Series*

**save** (*self*, *file\_path*, *pickle\_protocol=cloudpickle.DEFAULT\_PROTOCOL*)

Saves component at file path

**Parameters**

- **file\_path** (*str*) – Location to save file
- **pickle\_protocol** (*int*) – The pickle data stream format.

**Returns** None

```
class evalml.pipelines.XGBoostRegressor (eta=0.1, max_depth=6, min_child_weight=1,  
                                         n_estimators=100, random_seed=0, n_jobs=-1,  
                                         **kwargs)
```

XGBoost Regressor.

**Parameters**

- **eta** (*float*) – Boosting learning rate. Defaults to 0.1.
- **max\_depth** (*int*) – Maximum tree depth for base learners. Defaults to 6.
- **min\_child\_weight** (*float*) – Minimum sum of instance weight (hessian) needed in a child. Defaults to 1.0
- **n\_estimators** (*int*) – Number of gradient boosted trees. Equivalent to number of boosting rounds. Defaults to 100.
- **random\_seed** (*int*) – Seed for the random number generator. Defaults to 0.
- **n\_jobs** (*int*) – Number of parallel threads used to run xgboost. Note that creating thread contention will significantly slow down the algorithm. Defaults to -1.

**Attributes**

<b>hyper-parameter_ranges</b>	{ "eta": Real(0.000001, 1), "max_depth": Integer(1, 20), "min_child_weight": Real(1, 10), "n_estimators": Integer(1, 1000), }
<b>model_family</b>	ModelFamily.XGBOOST
<b>modifies_features</b>	True
<b>modifies_target</b>	False
<b>name</b>	XGBoost Regressor
<b>pre-dict_uses_y</b>	False
<b>SEED_MAX</b>	None
<b>SEED_MIN</b>	None
<b>supported_problem_types</b>	[ ProblemTypes.REGRESSION, ProblemTypes.TIME_SERIES_REGRESSION, ]

**Methods**

<code>clone</code>	Constructs a new component with the same parameters and random state.
<code>default_parameters</code>	Returns the default parameters for this component.
<code>describe</code>	Describe a component and its parameters
<code>feature_importance</code>	Returns importance associated with each feature.
<code>fit</code>	Fits component to data
<code>load</code>	Loads component at file path
<code>needs_fitting</code>	Returns boolean determining if component needs fitting before
<code>parameters</code>	Returns the parameters which were used to initialize the component
<code>predict</code>	Make predictions using selected features.
<code>predict_proba</code>	Make probability estimates for labels.
<code>save</code>	Saves component at file path

**clone** (*self*)

Constructs a new component with the same parameters and random state.

**Returns** A new instance of this component with identical parameters and random state.

**default\_parameters** (*cls*)

Returns the default parameters for this component.

Our convention is that `Component.default_parameters == Component().parameters`.

**Returns** default parameters for this component.

**Return type** dict

**describe** (*self*, *print\_name=False*, *return\_dict=False*)

Describe a component and its parameters

**Parameters**

- **print\_name** (*bool*, *optional*) – whether to print name of component
- **return\_dict** (*bool*, *optional*) – whether to return description as dictionary in the format {"name": name, "parameters": parameters}

**Returns** prints and returns dictionary

**Return type** None or dict

**property feature\_importance** (*self*)

Returns importance associated with each feature.

**Returns** Importance associated with each feature

**Return type** np.ndarray

**fit** (*self*, *X*, *y=None*)

Fits component to data

**Parameters**

- **X** (*list*, *pd.DataFrame* or *np.ndarray*) – The input training data of shape [n\_samples, n\_features]
- **y** (*list*, *pd.Series*, *np.ndarray*, *optional*) – The target training data of length [n\_samples]

**Returns** self

**static load** (*file\_path*)

Loads component at file path

**Parameters** *file\_path* (*str*) – Location to load file

**Returns** ComponentBase object

**needs\_fitting** (*self*)

Returns boolean determining if component needs fitting before calling predict, predict\_proba, transform, or feature\_importances. This can be overridden to False for components that do not need to be fit or whose fit methods do nothing.

**property parameters** (*self*)

Returns the parameters which were used to initialize the component

**predict** (*self*, *X*)

Make predictions using selected features.

**Parameters** *X* (*pd.DataFrame*, *np.ndarray*) – Data of shape [n\_samples, n\_features]

**Returns** Predicted values

**Return type** *pd.Series*

**predict\_proba** (*self*, *X*)

Make probability estimates for labels.

**Parameters** *X* (*pd.DataFrame*, *or np.ndarray*) – Features

**Returns** Probability estimates

**Return type** *pd.Series*

**save** (*self*, *file\_path*, *pickle\_protocol=cloudpickle.DEFAULT\_PROTOCOL*)

Saves component at file path

**Parameters**

- **file\_path** (*str*) – Location to save file
- **pickle\_protocol** (*int*) – The pickle data stream format.

**Returns** None

## Preprocessing

## Subpackages

## data\_splitters

## Submodules

## balanced\_classification\_sampler

## Module Contents

## Classes Summary

---

*BalancedClassificationSampler*Class for balanced classification downsampler.

---

## Contents

**class** evalml.preprocessing.data\_splitters.balanced\_classification\_sampler.**BalancedClassifi**

Class for balanced classification downsampler.

### Parameters

- **sampling\_ratio** (*float*) – The smallest minority:majority ratio that is accepted as ‘balanced’. For instance, a 1:4 ratio would be represented as 0.25, while a 1:1 ratio is 1.0. Must be between 0 and 1, inclusive. Defaults to 0.25.
- **sampling\_ratio\_dict** (*dict*) – A dictionary specifying the desired balanced ratio for each target value. Overrides sampling\_ratio if provided. Defaults to None.
- **min\_samples** (*int*) – The minimum number of samples that we must have for any class, pre or post sampling. If a class must be downsampled, it will not be downsampled past this value. To determine severe imbalance, the minority class must occur less often than this and must have a class ratio below min\_percentage. Must be greater than 0. Defaults to 100.
- **min\_percentage** (*float*) – The minimum percentage of the minimum class to total dataset that we tolerate, as long as it is above min\_samples. To determine severe imbalance, the minority class must have a class ratio below this and must occur less often than min\_samples. Must be between 0 and 0.5, inclusive. Defaults to 0.1.
- **random\_seed** (*int*) – The seed to use for random sampling. Defaults to 0.

### Methods

---

*fit\_resample*Resampling technique for this sampler.

---

**fit\_resample** (*self*, *X*, *y*)

Resampling technique for this sampler.

### Parameters

- **X** (*pd.DataFrame*) – Training data to fit and resample
- **y** (*pd.Series*) – Training data targets to fit and resample

**Returns** Indices to keep for training data

**Return type** list

## sampler\_base

### Module Contents

### Classes Summary

<i>SamplerBase</i>	Base class for all custom samplers.
--------------------	-------------------------------------

### Contents

**class** evalml.preprocessing.data\_splitters.sampler\_base.**SamplerBase** (*random\_seed=0*)  
Base class for all custom samplers.

**Parameters** **random\_seed** (*int*) – The seed to use for random sampling. Defaults to 0.

#### Methods

<i>fit_resample</i>	Resample the input data with this sampling strategy.
---------------------	--

**abstract fit\_resample** (*self, X, y*)  
Resample the input data with this sampling strategy.

#### Parameters

- **X** (*pd.DataFrame*) – Training data to fit and resample.
- **y** (*pd.Series*) – Training data targets to fit and resample.

**Returns** resampled X and y data for oversampling or indices to keep for undersampling.

**Return type** Tuple(pd.DataFrame, pd.Series) or list

## time\_series\_split

### Module Contents

### Classes Summary

<i>TimeSeriesSplit</i>	Rolling Origin Cross Validation for time series problems.
------------------------	---

## Contents

**class** evalml.preprocessing.data\_splitters.time\_series\_split.**TimeSeriesSplit** (*max\_delay=0*,  
*gap=0*,  
*date\_index=None*,  
*n\_splits=3*)

Rolling Origin Cross Validation for time series problems.

This class uses `max_delay` and `gap` values to take into account that evalml time series pipelines perform some feature and target engineering, e.g delaying input features and shifting the target variable by the desired amount. If the data that will be split already has all the features and appropriate target values, and then set `max_delay` and `gap` to 0.

### Parameters

- **max\_delay** (*int*) – Max delay value for feature engineering. Time series pipelines create delayed features from existing features. This process will introduce NaNs into the first `max_delay` number of rows. The splitter uses the last `max_delay` number of rows from the previous split as the first `max_delay` number of rows of the current split to avoid “throwing out” more data than is necessary. Defaults to 0.
- **gap** (*int*) – Gap used in time series problem. Time series pipelines shift the target variable by `gap` rows. Defaults to 0.
- **date\_index** (*str*) – Name of the column containing the datetime information used to order the data. Defaults to `None`.
- **n\_splits** (*int*) – number of data splits to make. Defaults to 3.

### Methods

<code>get_n_splits</code>	Get the number of data splits.
<code>split</code>	Get the time series splits.

**get\_n\_splits** (*self*, *X=None*, *y=None*, *groups=None*)  
Get the number of data splits.

**split** (*self*, *X*, *y=None*, *groups=None*)  
Get the time series splits.

*X* and *y* are assumed to be sorted in ascending time order. This method can handle passing in empty or `None` *X* and *y* data but note that *X* and *y* cannot be `None` or empty at the same time.

### Parameters

- **X** (*pd.DataFrame*, *None*) – Features to split.
- **y** (*pd.DataFrame*, *None*) – Target variable to split.
- **groups** – Ignored but kept for compatibility with sklearn api.

**Returns** Iterator of (train, test) indices tuples.

## training\_validation\_split

### Module Contents

### Classes Summary

<i>TrainingValidationSplit</i>	Split the training data into training and validation sets.
--------------------------------	--

### Contents

**class** evalml.preprocessing.data\_splitters.training\_validation\_split.**TrainingValidationSplit**

Split the training data into training and validation sets.

#### Parameters

- **test\_size** (*float*) – What percentage of data points should be included in the validation set. Defaults to the complement of *train\_size* if *train\_size* is set, and 0.25 otherwise.
- **train\_size** (*float*) – What percentage of data points should be included in the training set. Defaults to the complement of *test\_size*
- **shuffle** (*boolean*) – Whether to shuffle the data before splitting. Defaults to False.
- **stratify** (*list*) – Splits the data in a stratified fashion, using this argument as class labels. Defaults to None.
- **random\_seed** (*int*) – The seed to use for random sampling. Defaults to 0.

#### Methods

<i>get_n_splits</i>	Returns the number of splits of this object
<i>split</i>	Divides the data into training and testing sets

**static** **get\_n\_splits** ()

Returns the number of splits of this object

**split** (*self*, *X*, *y=None*)

Divides the data into training and testing sets

#### Parameters

- **X** (*pd.DataFrame*) – Dataframe of points to split
- **y** (*pd.Series*) – Series of points to split

**Returns** Indices to split data into training and test set

**Return type** list

## Package Contents

### Classes Summary

<code>BalancedClassificationSampler</code>	Class for balanced classification downsampler.
<code>SamplerBase</code>	Base class for all custom samplers.
<code>TimeSeriesSplit</code>	Rolling Origin Cross Validation for time series problems.
<code>TrainingValidationSplit</code>	Split the training data into training and validation sets.

### Contents

**class** evalml.preprocessing.data\_splitters.**BalancedClassificationSampler** (*sampling\_ratio=0.25, sampling\_ratio\_dict=None, min\_samples=100, min\_percentage=0.1, random\_seed=0*)

Class for balanced classification downsampler.

#### Parameters

- **sampling\_ratio** (*float*) – The smallest minority:majority ratio that is accepted as ‘balanced’. For instance, a 1:4 ratio would be represented as 0.25, while a 1:1 ratio is 1.0. Must be between 0 and 1, inclusive. Defaults to 0.25.
- **sampling\_ratio\_dict** (*dict*) – A dictionary specifying the desired balanced ratio for each target value. Overrides `sampling_ratio` if provided. Defaults to None.
- **min\_samples** (*int*) – The minimum number of samples that we must have for any class, pre or post sampling. If a class must be downsampled, it will not be downsampled past this value. To determine severe imbalance, the minority class must occur less often than this and must have a class ratio below `min_percentage`. Must be greater than 0. Defaults to 100.
- **min\_percentage** (*float*) – The minimum percentage of the minimum class to total dataset that we tolerate, as long as it is above `min_samples`. To determine severe imbalance, the minority class must have a class ratio below this and must occur less often than `min_samples`. Must be between 0 and 0.5, inclusive. Defaults to 0.1.
- **random\_seed** (*int*) – The seed to use for random sampling. Defaults to 0.

#### Methods

<code>fit_resample</code>	Resampling technique for this sampler.
---------------------------	--

**fit\_resample** (*self, X, y*)  
Resampling technique for this sampler.

#### Parameters

- **X** (*pd.DataFrame*) – Training data to fit and resample
- **y** (*pd.Series*) – Training data targets to fit and resample

**Returns** Indices to keep for training data



**Return type** list

**class** evalml.preprocessing.data\_splitters.**SamplerBase** (*random\_seed=0*)  
Base class for all custom samplers.

**Parameters** **random\_seed** (*int*) – The seed to use for random sampling. Defaults to 0.

**Methods**

---

<i>fit_resample</i>	Resample the input data with this sampling strategy.
---------------------	--

---

**abstract fit\_resample** (*self, X, y*)  
Resample the input data with this sampling strategy.

**Parameters**

- **X** (*pd.DataFrame*) – Training data to fit and resample.
- **y** (*pd.Series*) – Training data targets to fit and resample.

**Returns** resampled X and y data for oversampling or indices to keep for undersampling.

**Return type** Tuple(pd.DataFrame, pd.Series) or list

**class** evalml.preprocessing.data\_splitters.**TimeSeriesSplit** (*max\_delay=0, gap=0, date\_index=None, n\_splits=3*)

Rolling Origin Cross Validation for time series problems.

This class uses max\_delay and gap values to take into account that evalml time series pipelines perform some feature and target engineering, e.g delaying input features and shifting the target variable by the desired amount. If the data that will be split already has all the features and appropriate target values, and then set max\_delay and gap to 0.

**Parameters**

- **max\_delay** (*int*) – Max delay value for feature engineering. Time series pipelines create delayed features from existing features. This process will introduce NaNs into the first max\_delay number of rows. The splitter uses the last max\_delay number of rows from the previous split as the first max\_delay number of rows of the current split to avoid “throwing out” more data than is necessary. Defaults to 0.
- **gap** (*int*) – Gap used in time series problem. Time series pipelines shift the target variable by gap rows. Defaults to 0.
- **date\_index** (*str*) – Name of the column containing the datetime information used to order the data. Defaults to None.
- **n\_splits** (*int*) – number of data splits to make. Defaults to 3.

**Methods**

---

<i>get_n_splits</i>	Get the number of data splits.
<i>split</i>	Get the time series splits.

---

**get\_n\_splits** (*self, X=None, y=None, groups=None*)  
Get the number of data splits.

**split** (*self, X, y=None, groups=None*)  
Get the time series splits.

X and y are assumed to be sorted in ascending time order. This method can handle passing in empty or

None X and y data but note that X and y cannot be None or empty at the same time.

#### Parameters

- **x** (*pd.DataFrame*, *None*) – Features to split.
- **y** (*pd.DataFrame*, *None*) – Target variable to split.
- **groups** – Ignored but kept for compatibility with sklearn api.

**Returns** Iterator of (train, test) indices tuples.

```
class evalml.preprocessing.data_splitters.TrainingValidationSplit (test_size=None,  
                                                                train_size=None,  
                                                                shuf-  
                                                                fle=False,  
                                                                strat-  
                                                                ify=None,  
                                                                ran-  
                                                                dom_seed=0)
```

Split the training data into training and validation sets.

#### Parameters

- **test\_size** (*float*) – What percentage of data points should be included in the validation set. Defaults to the complement of *train\_size* if *train\_size* is set, and 0.25 otherwise.
- **train\_size** (*float*) – What percentage of data points should be included in the training set. Defaults to the complement of *test\_size*
- **shuffle** (*boolean*) – Whether to shuffle the data before splitting. Defaults to False.
- **stratify** (*list*) – Splits the data in a stratified fashion, using this argument as class labels. Defaults to None.
- **random\_seed** (*int*) – The seed to use for random sampling. Defaults to 0.

#### Methods

<code>get_n_splits</code>	Returns the number of splits of this object
<code>split</code>	Divides the data into training and testing sets

**static** `get_n_splits()`

Returns the number of splits of this object

**split** (*self*, *X*, *y=None*)

Divides the data into training and testing sets

#### Parameters

- **x** (*pd.DataFrame*) – Dataframe of points to split
- **y** (*pd.Series*) – Series of points to split

**Returns** Indices to split data into training and test set

**Return type** list

## Submodules

### utils

## Module Contents

## Functions

<code>drop_nan_target_rows</code>	Drops rows in X and y when row in the target y has a value of NaN.
<code>load_data</code>	Load features and target from file.
<code>number_of_features</code>	Get the number of features of each specific dtype in a DataFrame.
<code>split_data</code>	Splits data into train and test sets.
<code>target_distribution</code>	Get the target distributions.

## Contents

`evalml.preprocessing.utils.drop_nan_target_rows(X, y)`

Drops rows in X and y when row in the target y has a value of NaN.

### Parameters

- **x** (`pd.DataFrame`, `np.ndarray`) – Data to transform
- **y** (`pd.Series`, `np.ndarray`) – Target data

**Returns** Transformed X (and y, if passed in) with rows that had a NaN value removed.

**Return type** `pd.DataFrame`, `pd.DataFrame`

`evalml.preprocessing.utils.load_data(path, index, target, n_rows=None, drop=None, verbose=True, **kwargs)`

Load features and target from file.

### Parameters

- **path** (`str`) – Path to file or a http/ftp/s3 URL
- **index** (`str`) – Column for index
- **target** (`str`) – Column for target
- **n\_rows** (`int`) – Number of rows to return
- **drop** (`list`) – List of columns to drop
- **verbose** (`bool`) – If True, prints information about features and target

**Returns** Features matrix and target

**Return type** `pd.DataFrame`, `pd.Series`

`evalml.preprocessing.utils.number_of_features(dtypes)`

Get the number of features of each specific dtype in a DataFrame.

**Parameters** **dtypes** (`pd.Series`) – DataFrame.dtypes to get the number of features for

**Returns** dtypes and the number of features for each input type

**Return type** `pd.Series`

`evalml.preprocessing.utils.split_data(X, y, problem_type, problem_configuration=None, test_size=0.2, random_seed=0)`

Splits data into train and test sets.

**Parameters**

- **X** (`pd.DataFrame` or `np.ndarray`) – data of shape `[n_samples, n_features]`
- **y** (`pd.Series`, or `np.ndarray`) – target data of length `[n_samples]`
- **problem\_type** (`str` or `ProblemTypes`) – type of supervised learning problem. see `evalml.problem_types.problemtypes.all_problem_types` for a full list.
- **problem\_configuration** (`dict`) – Additional parameters needed to configure the search. For example, in time series problems, values should be passed in for the `date_index`, `gap`, and `max_delay` variables.
- **test\_size** (`float`) – What percentage of data points should be included in the test set. Defaults to 0.2 (20%).
- **random\_seed** (`int`) – Seed for the random number generator. Defaults to 0.

**Returns** Feature and target data each split into train and test sets

**Return type** `pd.DataFrame`, `pd.DataFrame`, `pd.Series`, `pd.Series`

`evalml.preprocessing.utils.target_distribution(targets)`

Get the target distributions.

**Parameters** **targets** (`pd.Series`) – Target data

**Returns** Target data and their frequency distribution as percentages.

**Return type** `pd.Series`

## Package Contents

## Classes Summary

<code>TimeSeriesSplit</code>	Rolling Origin Cross Validation for time series problems.
<code>TrainingValidationSplit</code>	Split the training data into training and validation sets.

## Functions

<code>drop_nan_target_rows</code>	Drops rows in X and y when row in the target y has a value of NaN.
<code>load_data</code>	Load features and target from file.
<code>number_of_features</code>	Get the number of features of each specific dtype in a DataFrame.
<code>split_data</code>	Splits data into train and test sets.
<code>target_distribution</code>	Get the target distributions.

## Contents

`evalml.preprocessing.drop_nan_target_rows(X, y)`

Drops rows in X and y when row in the target y has a value of NaN.

### Parameters

- **X** (`pd.DataFrame`, `np.ndarray`) – Data to transform
- **y** (`pd.Series`, `np.ndarray`) – Target data

**Returns** Transformed X (and y, if passed in) with rows that had a NaN value removed.

**Return type** `pd.DataFrame`, `pd.DataFrame`

`evalml.preprocessing.load_data(path, index, target, n_rows=None, drop=None, verbose=True, **kwargs)`

Load features and target from file.

### Parameters

- **path** (`str`) – Path to file or a http/ftp/s3 URL
- **index** (`str`) – Column for index
- **target** (`str`) – Column for target
- **n\_rows** (`int`) – Number of rows to return
- **drop** (`list`) – List of columns to drop
- **verbose** (`bool`) – If True, prints information about features and target

**Returns** Features matrix and target

**Return type** `pd.DataFrame`, `pd.Series`

`evalml.preprocessing.number_of_features(dtypes)`

Get the number of features of each specific dtype in a DataFrame.

**Parameters** **dtypes** (`pd.Series`) – DataFrame.dtypes to get the number of features for

**Returns** dtypes and the number of features for each input type

**Return type** `pd.Series`

`evalml.preprocessing.split_data(X, y, problem_type, problem_configuration=None, test_size=0.2, random_seed=0)`

Splits data into train and test sets.

### Parameters

- **X** (`pd.DataFrame` or `np.ndarray`) – data of shape [n\_samples, n\_features]
- **y** (`pd.Series`, or `np.ndarray`) – target data of length [n\_samples]
- **problem\_type** (`str` or `ProblemTypes`) – type of supervised learning problem. see `evalml.problem_types.problemtype.all_problem_types` for a full list.
- **problem\_configuration** (`dict`) – Additional parameters needed to configure the search. For example, in time series problems, values should be passed in for the `date_index`, `gap`, and `max_delay` variables.
- **test\_size** (`float`) – What percentage of data points should be included in the test set. Defaults to 0.2 (20%).
- **random\_seed** (`int`) – Seed for the random number generator. Defaults to 0.

**Returns** Feature and target data each split into train and test sets

**Return type** `pd.DataFrame`, `pd.DataFrame`, `pd.Series`, `pd.Series`

`evalml.preprocessing.target_distribution(targets)`

Get the target distributions.

**Parameters** `targets` (`pd.Series`) – Target data

**Returns** Target data and their frequency distribution as percentages.

**Return type** `pd.Series`

**class** `evalml.preprocessing.TimeSeriesSplit` (`max_delay=0`, `gap=0`, `date_index=None`,  
`n_splits=3`)

Rolling Origin Cross Validation for time series problems.

This class uses `max_delay` and `gap` values to take into account that evalml time series pipelines perform some feature and target engineering, e.g delaying input features and shifting the target variable by the desired amount. If the data that will be split already has all the features and appropriate target values, and then set `max_delay` and `gap` to 0.

#### Parameters

- **max\_delay** (`int`) – Max delay value for feature engineering. Time series pipelines create delayed features from existing features. This process will introduce NaNs into the first `max_delay` number of rows. The splitter uses the last `max_delay` number of rows from the previous split as the first `max_delay` number of rows of the current split to avoid “throwing out” more data than is necessary. Defaults to 0.
- **gap** (`int`) – Gap used in time series problem. Time series pipelines shift the target variable by `gap` rows. Defaults to 0.
- **date\_index** (`str`) – Name of the column containing the datetime information used to order the data. Defaults to `None`.
- **n\_splits** (`int`) – number of data splits to make. Defaults to 3.

#### Methods

---

<code>get_n_splits</code>	Get the number of data splits.
<code>split</code>	Get the time series splits.

---

**get\_n\_splits** (`self`, `X=None`, `y=None`, `groups=None`)

Get the number of data splits.

**split** (`self`, `X`, `y=None`, `groups=None`)

Get the time series splits.

`X` and `y` are assumed to be sorted in ascending time order. This method can handle passing in empty or `None` `X` and `y` data but note that `X` and `y` cannot be `None` or empty at the same time.

#### Parameters

- **X** (`pd.DataFrame`, `None`) – Features to split.
- **y** (`pd.DataFrame`, `None`) – Target variable to split.
- **groups** – Ignored but kept for compatibility with sklearn api.

**Returns** Iterator of (train, test) indices tuples.

**class** evalml.preprocessing.**TrainingValidationSplit** (*test\_size=None, train\_size=None, shuffle=False, stratify=None, random\_seed=0*)

Split the training data into training and validation sets.

#### Parameters

- **test\_size** (*float*) – What percentage of data points should be included in the validation set. Defaults to the complement of *train\_size* if *train\_size* is set, and 0.25 otherwise.
- **train\_size** (*float*) – What percentage of data points should be included in the training set. Defaults to the complement of *test\_size*
- **shuffle** (*boolean*) – Whether to shuffle the data before splitting. Defaults to False.
- **stratify** (*list*) – Splits the data in a stratified fashion, using this argument as class labels. Defaults to None.
- **random\_seed** (*int*) – The seed to use for random sampling. Defaults to 0.

#### Methods

<code>get_n_splits</code>	Returns the number of splits of this object
<code>split</code>	Divides the data into training and testing sets

**static** `get_n_splits()`

Returns the number of splits of this object

**split** (*self, X, y=None*)

Divides the data into training and testing sets

#### Parameters

- **X** (*pd.DataFrame*) – Dataframe of points to split
- **y** (*pd.Series*) – Series of points to split

**Returns** Indices to split data into training and test set

**Return type** list

## Problem Types

## Submodules

## problem\_types

## Module Contents

## Classes Summary

<code>ProblemTypes</code>	Enum defining the supported types of machine learning problems.
---------------------------	---

## Contents

**class** evalml.problem\_types.problem\_types.**ProblemTypes**

Enum defining the supported types of machine learning problems.

### Attributes

<b>BINARY</b>	Binary classification problem.
<b>MULTI-CLASS</b>	Multiclass classification problem.
<b>REGRESSION</b>	Regression problem.
<b>TIME_SERIES_BINARY</b>	Time series binary classification problem.
<b>TIME_SERIES_MULTICLASS</b>	Time series multiclass classification problem.
<b>TIME_SERIES_REGRESSION</b>	Time series regression problem.

### Methods

<i>all_problem_types</i>	Get a list of all defined problem types.
<i>name</i>	The name of the Enum member.
<i>value</i>	The value of the Enum member.

**all\_problem\_types** (*cls*)

Get a list of all defined problem types.

**Returns** list

**Return type** list(*ProblemTypes*)

**name** (*self*)

The name of the Enum member.

**value** (*self*)

The value of the Enum member.

## utils

## Module Contents

### Functions

<i>detect_problem_type</i>	Determine the type of problem is being solved based on the targets (binary vs multiclass classification, regression)
<i>handle_problem_types</i>	Handles problem_type by either returning the ProblemTypes or converting from a str.
<i>is_binary</i>	Determines if the provided problem_type is a binary classification problem type
<i>is_classification</i>	Determines if the provided problem_type is a classification problem type

continues on next page



Table 693 – continued from previous page

<code>is_multiclass</code>	Determines if the provided <code>problem_type</code> is a multiclass classification problem type
<code>is_regression</code>	Determines if the provided <code>problem_type</code> is a regression problem type
<code>is_time_series</code>	Determines if the provided <code>problem_type</code> is a time series problem type

## Contents

`evalml.problem_types.utils.detect_problem_type(y)`

**Determine the type of problem is being solved based on the targets (binary vs multiclass classification, regression)**  
Ignores missing and null data

**Parameters** `y` (`pd.Series`) – the target labels to predict

**Returns** `ProblemType` Enum

**Return type** `ProblemType`

## Example

```
>>> y = pd.Series([0, 1, 0, 0, 0, 1, 1, 0, 0, 1, 1])
>>> problem_type = detect_problem_type(y)
>>> assert problem_type == ProblemTypes.BINARY
```

`evalml.problem_types.utils.handle_problem_types(problem_type)`

Handles `problem_type` by either returning the `ProblemTypes` or converting from a str.

**Parameters** `problem_type` (`str` or `ProblemTypes`) – Problem type that needs to be handled

**Returns** `ProblemTypes`

`evalml.problem_types.utils.is_binary(problem_type)`

Determines if the provided `problem_type` is a binary classification problem type

**Parameters** `problem_type` (`str` or `ProblemTypes`) – type of supervised learning problem. See `evalml.problem_types.ProblemType.all_problem_types` for a full list.

**Returns** Whether or not the provided `problem_type` is a binary classification problem type.

**Return type** `bool`

`evalml.problem_types.utils.is_classification(problem_type)`

Determines if the provided `problem_type` is a classification problem type

**Parameters** `problem_type` (`str` or `ProblemTypes`) – type of supervised learning problem. See `evalml.problem_types.ProblemType.all_problem_types` for a full list.

**Returns** Whether or not the provided `problem_type` is a classification problem type.

**Return type** `bool`

`evalml.problem_types.utils.is_multiclass(problem_type)`

Determines if the provided `problem_type` is a multiclass classification problem type

**Parameters** `problem_type` (*str or ProblemTypes*) – type of supervised learning problem. See `evalml.problem_types.ProblemType.all_problem_types` for a full list.

**Returns** Whether or not the provided `problem_type` is a multiclass classification problem type.

**Return type** bool

`evalml.problem_types.utils.is_regression(problem_type)`

Determines if the provided `problem_type` is a regression problem type

**Parameters** `problem_type` (*str or ProblemTypes*) – type of supervised learning problem. See `evalml.problem_types.ProblemType.all_problem_types` for a full list.

**Returns** Whether or not the provided `problem_type` is a regression problem type.

**Return type** bool

`evalml.problem_types.utils.is_time_series(problem_type)`

Determines if the provided `problem_type` is a time series problem type

**Parameters** `problem_type` (*str or ProblemTypes*) – type of supervised learning problem. See `evalml.problem_types.ProblemType.all_problem_types` for a full list.

**Returns** Whether or not the provided `problem_type` is a time series problem type.

**Return type** bool

## Package Contents

### Classes Summary

<code>ProblemTypes</code>	Enum defining the supported types of machine learning problems.
---------------------------	---

### Functions

<code>detect_problem_type</code>	Determine the type of problem is being solved based on the targets (binary vs multiclass classification, regression)
<code>handle_problem_types</code>	Handles <code>problem_type</code> by either returning the <code>ProblemTypes</code> or converting from a <code>str</code> .
<code>is_binary</code>	Determines if the provided <code>problem_type</code> is a binary classification problem type
<code>is_classification</code>	Determines if the provided <code>problem_type</code> is a classification problem type
<code>is_multiclass</code>	Determines if the provided <code>problem_type</code> is a multiclass classification problem type
<code>is_regression</code>	Determines if the provided <code>problem_type</code> is a regression problem type
<code>is_time_series</code>	Determines if the provided <code>problem_type</code> is a time series problem type

## Contents

`evalml.problem_types.detect_problem_type(y)`

**Determine the type of problem is being solved based on the targets (binary vs multiclass classification, regression)**

Ignores missing and null data

**Parameters** `y` (`pd.Series`) – the target labels to predict

**Returns** `ProblemType` Enum

**Return type** `ProblemType`

### Example

```
>>> y = pd.Series([0, 1, 0, 0, 0, 1, 1, 0, 0, 1, 1])
>>> problem_type = detect_problem_type(y)
>>> assert problem_type == ProblemTypes.BINARY
```

`evalml.problem_types.handle_problem_types(problem_type)`

Handles `problem_type` by either returning the `ProblemTypes` or converting from a str.

**Parameters** `problem_type` (`str` or `ProblemTypes`) – Problem type that needs to be handled

**Returns** `ProblemTypes`

`evalml.problem_types.is_binary(problem_type)`

Determines if the provided `problem_type` is a binary classification problem type

**Parameters** `problem_type` (`str` or `ProblemTypes`) – type of supervised learning problem. See `evalml.problem_types.ProblemType.all_problem_types` for a full list.

**Returns** Whether or not the provided `problem_type` is a binary classification problem type.

**Return type** `bool`

`evalml.problem_types.is_classification(problem_type)`

Determines if the provided `problem_type` is a classification problem type

**Parameters** `problem_type` (`str` or `ProblemTypes`) – type of supervised learning problem. See `evalml.problem_types.ProblemType.all_problem_types` for a full list.

**Returns** Whether or not the provided `problem_type` is a classification problem type.

**Return type** `bool`

`evalml.problem_types.is_multiclass(problem_type)`

Determines if the provided `problem_type` is a multiclass classification problem type

**Parameters** `problem_type` (`str` or `ProblemTypes`) – type of supervised learning problem. See `evalml.problem_types.ProblemType.all_problem_types` for a full list.

**Returns** Whether or not the provided `problem_type` is a multiclass classification problem type.

**Return type** `bool`

`evalml.problem_types.is_regression(problem_type)`

Determines if the provided `problem_type` is a regression problem type

**Parameters** `problem_type` (`str` or `ProblemTypes`) – type of supervised learning problem. See `evalml.problem_types.ProblemType.all_problem_types` for a full list.

**Returns** Whether or not the provided `problem_type` is a regression problem type.

**Return type** bool

`evalml.problem_types.is_time_series(problem_type)`

Determines if the provided `problem_type` is a time series problem type

**Parameters** `problem_type` (*str* or `ProblemTypes`) – type of supervised learning problem. See `evalml.problem_types.ProblemType.all_problem_types` for a full list.

**Returns** Whether or not the provided `problem_type` is a time series problem type.

**Return type** bool

**class** `evalml.problem_types.ProblemTypes`

Enum defining the supported types of machine learning problems.

**Attributes**

<b>BINARY</b>	Binary classification problem.
<b>MULTI-CLASS</b>	Multiclass classification problem.
<b>REGRESSION</b>	Regression problem.
<b>TIME_SERIES_BINARY</b>	Time series binary classification problem.
<b>TIME_SERIES_MULTICLASS</b>	Time series multiclass classification problem.
<b>TIME_SERIES_REGRESSION</b>	Time series regression problem.

**Methods**

<code>all_problem_types</code>	Get a list of all defined problem types.
<code>name</code>	The name of the Enum member.
<code>value</code>	The value of the Enum member.

**all\_problem\_types** (*cls*)

Get a list of all defined problem types.

**Returns** list

**Return type** list(`ProblemTypes`)

**name** (*self*)

The name of the Enum member.

**value** (*self*)

The value of the Enum member.

**Tuners**

**Submodules**

**grid\_search\_tuner**

**Module Contents**

## Classes Summary

<i>GridSearchTuner</i>	Grid Search Optimizer, which generates all of the possible points to search for using a grid.
------------------------	---

## Contents

**class** evalml.tuners.grid\_search\_tuner.**GridSearchTuner** (*pipeline\_hyperparameter\_ranges*, *n\_points=10*, *random\_seed=0*)

Grid Search Optimizer, which generates all of the possible points to search for using a grid.

### Parameters

- **pipeline\_hyperparameter\_ranges** (*dict*) – a set of hyperparameter ranges corresponding to a pipeline’s parameters
- **n\_points** (*int*) – The number of points to sample from along each dimension defined in the *space* argument. Defaults to 10.
- **random\_seed** (*int*) – Seed for random number generator. Unused in this class, defaults to 0.

### Example

```
>>> tuner = GridSearchTuner({'My Component': {'param a': [0.0, 10.0], 'param b': [
↪ 'a', 'b', 'c']}}, n_points=5)
>>> proposal = tuner.propose()
>>> assert proposal.keys() == {'My Component'}
>>> assert proposal['My Component'] == {'param a': 0.0, 'param b': 'a'}
```

### Methods

<i>add</i>	Not applicable to grid search tuner as generated parameters are
<i>is_search_space_exhausted</i>	Checks if it is possible to generate a set of valid parameters. Stores generated parameters in
<i>propose</i>	Returns parameters from <i>_grid_points</i> iterations

**add** (*self*, *pipeline\_parameters*, *score*)

Not applicable to grid search tuner as generated parameters are not dependent on scores of previous parameters.

### Parameters

- **pipeline\_parameters** (*dict*) – a dict of the parameters used to evaluate a pipeline
- **score** (*float*) – the score obtained by evaluating the pipeline with the provided parameters

**is\_search\_space\_exhausted** (*self*)

Checks if it is possible to generate a set of valid parameters. Stores generated parameters in *self.curr\_params* to be returned by *propose()*.

**Raises** **NoParamsException** – If a search space is exhausted, then this exception is thrown.

**Returns** If no more valid parameters exists in the search space, return false.

**Return type** bool

**propose** (*self*)

Returns parameters from `_grid_points` iterations

If all possible combinations of parameters have been scored, then `NoParamsException` is raised.

**Returns** proposed pipeline parameters

**Return type** dict

## random\_search\_tuner

### Module Contents

### Classes Summary

---

<i>RandomSearchTuner</i>
--------------------------

---

Random Search Optimizer.
--------------------------

---

### Contents

**class** evalml.tuners.random\_search\_tuner.**RandomSearchTuner** (*pipeline\_hyperparameter\_ranges*,  
with\_replacement=False,  
replacement\_max\_attempts=10,  
random\_seed=0)

Random Search Optimizer.

#### Parameters

- **pipeline\_hyperparameter\_ranges** (*dict*) – a set of hyperparameter ranges corresponding to a pipeline’s parameters
- **with\_replacement** (*bool*) – If false, only unique hyperparameters will be shown
- **replacement\_max\_attempts** (*int*) – The maximum number of tries to get a unique set of random parameters. Only used if tuner is initialized with `with_replacement=True`
- **random\_seed** (*int*) – Seed for random number generator. Defaults to 0.

### Example

```
>>> tuner = RandomSearchTuner({'My Component': {'param a': [0.0, 10.0], 'param b':  
↪ ['a', 'b', 'c']}}, random_seed=42)  
>>> proposal = tuner.propose()  
>>> assert proposal.keys() == {'My Component'}  
>>> assert proposal['My Component'] == {'param a': 3.7454011884736254, 'param b':  
↪ 'c'}
```

### Methods

<code>add</code>	Not applicable to random search tuner as generated parameters are
<code>is_search_space_exhausted</code>	Checks if it is possible to generate a set of valid parameters. Stores generated parameters in
<code>propose</code>	Generate a unique set of parameters.

**add** (*self*, *pipeline\_parameters*, *score*)

Not applicable to random search tuner as generated parameters are not dependent on scores of previous parameters.

#### Parameters

- **pipeline\_parameters** (*dict*) – A dict of the parameters used to evaluate a pipeline
- **score** (*float*) – The score obtained by evaluating the pipeline with the provided parameters

**is\_search\_space\_exhausted** (*self*)

Checks if it is possible to generate a set of valid parameters. Stores generated parameters in *self.curr\_params* to be returned by `propose()`.

**Raises** `NoParamsException` – If a search space is exhausted, then this exception is thrown.

**Returns** If no more valid parameters exists in the search space, return false.

**Return type** bool

**propose** (*self*)

Generate a unique set of parameters.

If tuner was initialized with `with_replacement=True` and the tuner is unable to generate a unique set of parameters after `replacement_max_attempts` tries, then `NoParamsException` is raised.

**Returns** Proposed pipeline parameters

**Return type** dict

## skopt\_tuner

### Module Contents

### Classes Summary

<code>SKOptTuner</code>	Bayesian Optimizer.
-------------------------	---------------------

### Attributes Summary

<code>logger</code>
---------------------

## Contents

`evalml.tuners.skopt_tuner.logger`

**class** `evalml.tuners.skopt_tuner.SKOptTuner` (*pipeline\_hyperparameter\_ranges*, *random\_seed=0*)  
Bayesian Optimizer.

### Parameters

- **pipeline\_hyperparameter\_ranges** (*dict*) – A set of hyperparameter ranges corresponding to a pipeline’s parameters.
- **random\_seed** (*int*) – The seed for the random number generator. Defaults to 0.

### Methods

<i>add</i>	Add score to sample.
<i>is_search_space_exhausted</i>	Optional. If possible search space for tuner is finite, this method indicates whether or not all possible parameters have been scored.
<i>propose</i>	Returns a suggested set of parameters to train and score a pipeline with, based off the search space dimensions and prior samples.

**add** (*self*, *pipeline\_parameters*, *score*)  
Add score to sample.

### Parameters

- **pipeline\_parameters** (*dict*) – A dict of the parameters used to evaluate a pipeline
- **score** (*float*) – The score obtained by evaluating the pipeline with the provided parameters

**Returns** None

**is\_search\_space\_exhausted** (*self*)

Optional. If possible search space for tuner is finite, this method indicates whether or not all possible parameters have been scored.

**Returns** Returns true if all possible parameters in a search space has been scored.

**Return type** bool

**propose** (*self*)

Returns a suggested set of parameters to train and score a pipeline with, based off the search space dimensions and prior samples.

**Returns** Proposed pipeline parameters

**Return type** dict



## tuner

### Module Contents

### Classes Summary

<i>Tuner</i>	Defines API for base Tuner classes.
--------------	-------------------------------------

### Contents

**class** evalml.tuners.tuner.**Tuner** (*pipeline\_hyperparameter\_ranges*, *random\_seed=0*)

Defines API for base Tuner classes.

Tuners implement different strategies for sampling from a search space. They're used in EvalML to search the space of pipeline hyperparameters.

#### Parameters

- **pipeline\_hyperparameter\_ranges** (*dict*) – a set of hyperparameter ranges corresponding to a pipeline's parameters.
- **random\_seed** (*int*) – The random state. Defaults to 0.

#### Methods

<i>add</i>	Register a set of hyperparameters with the score obtained from training a pipeline with those hyperparameters.
<i>is_search_space_exhausted</i>	Optional. If possible search space for tuner is finite, this method indicates whether or not all possible parameters have been scored.
<i>propose</i>	Returns a suggested set of parameters to train and score a pipeline with, based off the search space dimensions and prior samples.

**abstract add** (*self*, *pipeline\_parameters*, *score*)

Register a set of hyperparameters with the score obtained from training a pipeline with those hyperparameters.

#### Parameters

- **pipeline\_parameters** (*dict*) – a dict of the parameters used to evaluate a pipeline
- **score** (*float*) – the score obtained by evaluating the pipeline with the provided parameters

**Returns** None

**is\_search\_space\_exhausted** (*self*)

Optional. If possible search space for tuner is finite, this method indicates whether or not all possible parameters have been scored.

**Returns** Returns true if all possible parameters in a search space has been scored.

**Return type** bool

**abstract propose** (*self*)

Returns a suggested set of parameters to train and score a pipeline with, based off the search space dimensions and prior samples.

**Returns** Proposed pipeline parameters

**Return type** dict

## tuner\_exceptions

## Module Contents

### Contents

**exception** evalml.tuners.tuner\_exceptions.NoParamsException

Raised when a tuner exhausts its search space and runs out of parameters to propose.

**exception** evalml.tuners.tuner\_exceptions.ParameterError

Raised when a tuner encounters an error with the parameters being used with it.

## Package Contents

## Classes Summary

<i>GridSearchTuner</i>	Grid Search Optimizer, which generates all of the possible points to search for using a grid.
<i>RandomSearchTuner</i>	Random Search Optimizer.
<i>SKOptTuner</i>	Bayesian Optimizer.
<i>Tuner</i>	Defines API for base Tuner classes.

## Exceptions Summary

---

### Contents

**class** evalml.tuners.GridSearchTuner (*pipeline\_hyperparameter\_ranges*, *n\_points=10*, *random\_seed=0*)

Grid Search Optimizer, which generates all of the possible points to search for using a grid.

#### Parameters

- **pipeline\_hyperparameter\_ranges** (*dict*) – a set of hyperparameter ranges corresponding to a pipeline’s parameters
- **n\_points** (*int*) – The number of points to sample from along each dimension defined in the *space* argument. Defaults to 10.
- **random\_seed** (*int*) – Seed for random number generator. Unused in this class, defaults to 0.

## Example

```
>>> tuner = GridSearchTuner({'My Component': {'param a': [0.0, 10.0], 'param b': [
↪ 'a', 'b', 'c']}}, n_points=5)
>>> proposal = tuner.propose()
>>> assert proposal.keys() == {'My Component'}
>>> assert proposal['My Component'] == {'param a': 0.0, 'param b': 'a'}
```

## Methods

<code>add</code>	Not applicable to grid search tuner as generated parameters are
<code>is_search_space_exhausted</code>	Checks if it is possible to generate a set of valid parameters. Stores generated parameters in
<code>propose</code>	Returns parameters from <code>_grid_points</code> iterations

**add** (*self*, *pipeline\_parameters*, *score*)

Not applicable to grid search tuner as generated parameters are not dependent on scores of previous parameters.

### Parameters

- **pipeline\_parameters** (*dict*) – a dict of the parameters used to evaluate a pipeline
- **score** (*float*) – the score obtained by evaluating the pipeline with the provided parameters

**is\_search\_space\_exhausted** (*self*)

Checks if it is possible to generate a set of valid parameters. Stores generated parameters in `self.curr_params` to be returned by `propose()`.

**Raises** `NoParamsException` – If a search space is exhausted, then this exception is thrown.

**Returns** If no more valid parameters exists in the search space, return false.

**Return type** bool

**propose** (*self*)

Returns parameters from `_grid_points` iterations

If all possible combinations of parameters have been scored, then `NoParamsException` is raised.

**Returns** proposed pipeline parameters

**Return type** dict

**exception** `evalml.tuners.NoParamsException`

Raised when a tuner exhausts its search space and runs out of parameters to propose.

**exception** `evalml.tuners.ParameterError`

Raised when a tuner encounters an error with the parameters being used with it.

**class** `evalml.tuners.RandomSearchTuner` (*pipeline\_hyperparameter\_ranges*,  
*with\_replacement=False*, *replacement\_max\_attempts=10*, *random\_seed=0*)

Random Search Optimizer.

### Parameters

- **pipeline\_hyperparameter\_ranges** (*dict*) – a set of hyperparameter ranges corresponding to a pipeline’s parameters

- **with\_replacement** (*bool*) – If false, only unique hyperparameters will be shown
- **replacement\_max\_attempts** (*int*) – The maximum number of tries to get a unique set of random parameters. Only used if tuner is initialized with `with_replacement=True`
- **random\_seed** (*int*) – Seed for random number generator. Defaults to 0.

### Example

```
>>> tuner = RandomSearchTuner({'My Component': {'param a': [0.0, 10.0], 'param b':
↪ ['a', 'b', 'c']}}, random_seed=42)
>>> proposal = tuner.propose()
>>> assert proposal.keys() == {'My Component'}
>>> assert proposal['My Component'] == {'param a': 3.7454011884736254, 'param b':
↪ 'c'}
```

### Methods

<code>add</code>	Not applicable to random search tuner as generated parameters are
<code>is_search_space_exhausted</code>	Checks if it is possible to generate a set of valid parameters. Stores generated parameters in
<code>propose</code>	Generate a unique set of parameters.

**add** (*self*, *pipeline\_parameters*, *score*)

Not applicable to random search tuner as generated parameters are not dependent on scores of previous parameters.

#### Parameters

- **pipeline\_parameters** (*dict*) – A dict of the parameters used to evaluate a pipeline
- **score** (*float*) – The score obtained by evaluating the pipeline with the provided parameters

**is\_search\_space\_exhausted** (*self*)

Checks if it is possible to generate a set of valid parameters. Stores generated parameters in `self.curr_params` to be returned by `propose()`.

**Raises** `NoParamsException` – If a search space is exhausted, then this exception is thrown.

**Returns** If no more valid parameters exists in the search space, return false.

**Return type** `bool`

**propose** (*self*)

Generate a unique set of parameters.

If tuner was initialized with `with_replacement=True` and the tuner is unable to generate a unique set of parameters after `replacement_max_attempts` tries, then `NoParamsException` is raised.

**Returns** Proposed pipeline parameters

**Return type** `dict`

**class** `evalml.tuners.SKOptTuner` (*pipeline\_hyperparameter\_ranges*, *random\_seed=0*)  
Bayesian Optimizer.

#### Parameters

- **pipeline\_hyperparameter\_ranges** (*dict*) – A set of hyperparameter ranges corresponding to a pipeline’s parameters.
- **random\_seed** (*int*) – The seed for the random number generator. Defaults to 0.

### Methods

<i>add</i>	Add score to sample.
<i>is_search_space_exhausted</i>	Optional. If possible search space for tuner is finite, this method indicates whether or not all possible parameters have been scored.
<i>propose</i>	Returns a suggested set of parameters to train and score a pipeline with, based off the search space dimensions and prior samples.

**add** (*self*, *pipeline\_parameters*, *score*)  
Add score to sample.

### Parameters

- **pipeline\_parameters** (*dict*) – A dict of the parameters used to evaluate a pipeline
- **score** (*float*) – The score obtained by evaluating the pipeline with the provided parameters

**Returns** None

**is\_search\_space\_exhausted** (*self*)

Optional. If possible search space for tuner is finite, this method indicates whether or not all possible parameters have been scored.

**Returns** Returns true if all possible parameters in a search space has been scored.

**Return type** bool

**propose** (*self*)

Returns a suggested set of parameters to train and score a pipeline with, based off the search space dimensions and prior samples.

**Returns** Proposed pipeline parameters

**Return type** dict

**class** evalml.tuners.**Tuner** (*pipeline\_hyperparameter\_ranges*, *random\_seed=0*)

Defines API for base Tuner classes.

Tuners implement different strategies for sampling from a search space. They’re used in EvalML to search the space of pipeline hyperparameters.

### Parameters

- **pipeline\_hyperparameter\_ranges** (*dict*) – a set of hyperparameter ranges corresponding to a pipeline’s parameters.
- **random\_seed** (*int*) – The random state. Defaults to 0.

### Methods

<i>add</i>	Register a set of hyperparameters with the score obtained from training a pipeline with those hyperparameters.
------------	--

continues on next page

Table 711 – continued from previous page

<code>is_search_space_exhausted</code>	Optional. If possible search space for tuner is finite, this method indicates whether or not all possible parameters have been scored.
<code>propose</code>	Returns a suggested set of parameters to train and score a pipeline with, based off the search space dimensions and prior samples.

**abstract add** (*self*, *pipeline\_parameters*, *score*)

Register a set of hyperparameters with the score obtained from training a pipeline with those hyperparameters.

**Parameters**

- **pipeline\_parameters** (*dict*) – a dict of the parameters used to evaluate a pipeline
- **score** (*float*) – the score obtained by evaluating the pipeline with the provided parameters

**Returns** None

**is\_search\_space\_exhausted** (*self*)

Optional. If possible search space for tuner is finite, this method indicates whether or not all possible parameters have been scored.

**Returns** Returns true if all possible parameters in a search space has been scored.

**Return type** bool

**abstract propose** (*self*)

Returns a suggested set of parameters to train and score a pipeline with, based off the search space dimensions and prior samples.

**Returns** Proposed pipeline parameters

**Return type** dict

## Utils

## Submodules

## base\_meta

## Module Contents

## Classes Summary

<code>BaseMeta</code>	Metaclass that overrides creating a new component or pipeline by wrapping methods with validators and setters
-----------------------	---

## Contents

### **class** evalml.utils.base\_meta.BaseMeta

Metaclass that overrides creating a new component or pipeline by wrapping methods with validators and setters

#### Attributes

<b>FIT_METHODS</b>	['fit', 'fit_transform']
<b>METHODS_TO_CHECK</b>	['predict', 'predict_proba', 'transform', 'inverse_transform']
<b>PROPERTIES_TO_CHECK</b>	['feature_importance']

#### Methods

<i>register</i>	Register a virtual subclass of an ABC.
<i>set_fit</i>	

**register** (*cls*, *subclass*)

Register a virtual subclass of an ABC.

Returns the subclass, to allow usage as a class decorator.

**classmethod** **set\_fit** (*cls*, *method*)

## cli\_utils

### Module Contents

#### Functions

<i>get_evalml_root</i>	Gets location where evalml is installed.
<i>get_installed_packages</i>	Get dictionary mapping installed package names to their versions.
<i>get_sys_info</i>	Returns system information.
<i>print_deps</i>	Prints the version number of each dependency.
<i>print_info</i>	Prints information about the system, evalml, and dependencies of evalml.
<i>print_sys_info</i>	Prints system information.

## Attributes Summary

---

*logger*

---

## Contents

`evalml.utils.cli_utils.get_evalml_root()`

Gets location where evalml is installed.

**Returns** Location where evalml is installed.

`evalml.utils.cli_utils.get_installed_packages()`

Get dictionary mapping installed package names to their versions.

**Returns** Dictionary mapping installed package names to their versions.

`evalml.utils.cli_utils.get_sys_info()`

Returns system information.

**Returns** List of tuples about system stats.

`evalml.utils.cli_utils.logger`

`evalml.utils.cli_utils.print_deps()`

Prints the version number of each dependency.

**Returns** None

`evalml.utils.cli_utils.print_info()`

Prints information about the system, evalml, and dependencies of evalml.

**Returns** None

`evalml.utils.cli_utils.print_sys_info()`

Prints system information.

**Returns** None

## gen\_utils

## Module Contents

## Classes Summary

---

*classproperty*

Allows function to be accessed as a class level property.

---



## Functions

<code>convert_to_seconds</code>	Converts a string describing a length of time to its length in seconds.
<code>deprecate_arg</code>	Helper to raise warnings when a deprecated arg is used.
<code>drop_rows_with_nans</code>	Drop rows that have any NaNs in all dataframes or series.
<code>get_importable_subclasses</code>	Get importable subclasses of a base class. Used to list all of our
<code>get_random_seed</code>	Given a <code>numpy.random.RandomState</code> object, generate an int representing a seed value for another random number generator. Or, if given an int, return that int.
<code>get_random_state</code>	Generates a <code>numpy.random.RandomState</code> instance using seed.
<code>import_or_raise</code>	Attempts to import the requested library by name.
<code>is_all_numeric</code>	Checks if the given DataFrame contains only numeric values
<code>jupyter_check</code>	Get whether or not the code is being run in a Jupyter environment (such as Jupyter Notebook or Jupyter Lab)
<code>pad_with_nans</code>	Pad the beginning num_to_pad rows with nans.
<code>safe_repr</code>	Convert the given value into a string that can safely be used for repr
<code>save_plot</code>	Saves fig to filepath if specified, or to a default location if not.

## Attributes Summary

<code>logger</code>
<code>SEED_BOUNDS</code>

## Contents

**class** `evalml.utils.gen_utils.classproperty` (*func*)

Allows function to be accessed as a class level property.

Example:

```
class LogisticRegressionBinaryPipeline(PipelineBase):
    component_graph = ['Simple Imputer', 'Logistic Regression Classifier']

    @classproperty
    def summary(cls):
        summary = ""
        for component in cls.component_graph:
            component = handle_component_class(component)
            summary += component.name + " + "
        return summary
```

(continues on next page)

(continued from previous page)

```

assert LogisticRegressionBinaryPipeline.summary == "Simple Imputer + Logistic_
↪Regression Classifier + "
assert LogisticRegressionBinaryPipeline().summary == "Simple Imputer + Logistic_
↪Regression Classifier + "

```

`evalml.utils.gen_utils.convert_to_seconds(input_str)`  
 Converts a string describing a length of time to its length in seconds.

`evalml.utils.gen_utils.deprecate_arg(old_arg, new_arg, old_value, new_value)`  
 Helper to raise warnings when a deprecated arg is used.

**Parameters**

- **old\_arg** (*str*) – Name of old/deprecated argument.
- **new\_arg** (*str*) – Name of new argument.
- **old\_value** (*Any*) – Value the user passed in for the old argument.
- **new\_value** (*Any*) – Value the user passed in for the new argument.

**Returns** old\_value if not None, else new\_value

`evalml.utils.gen_utils.drop_rows_with_nans(*pd_data)`  
 Drop rows that have any NaNs in all dataframes or series.

**Parameters** **\*pd\_data** (*sequence of pd.Series or pd.DataFrame or None*) –

**Returns** list of pd.DataFrame or pd.Series or None

`evalml.utils.gen_utils.get_importable_subclasses(base_class, used_in_automl=True)`  
 Get importable subclasses of a base class. Used to list all of our estimators, transformers, components and pipelines dynamically.

**Parameters**

- **base\_class** (*abc.ABCMeta*) – Base class to find all of the subclasses for.
- **args** (*list*) – Args used to instantiate the subclass. `[{}]` for a pipeline, and `[]` for all other classes.
- **used\_in\_automl** – Not all components/pipelines/estimators are used in automl search. If True, only include those subclasses that are used in the search. This would mean excluding classes related to ExtraTrees, ElasticNet, and Baseline estimators.

**Returns** List of subclasses.

`evalml.utils.gen_utils.get_random_seed(random_state, min_bound=SEED_BOUNDS.min_bound, max_bound=SEED_BOUNDS.max_bound)`  
 Given a `numpy.random.RandomState` object, generate an int representing a seed value for another random number generator. Or, if given an int, return that int.

To protect against invalid input to a particular library’s random number generator, if an int value is provided, and it is outside the bounds “[min\_bound, max\_bound)”, the value will be projected into the range between the min\_bound (inclusive) and max\_bound (exclusive) using modular arithmetic.

**Parameters**

- **random\_state** (*int, numpy.random.RandomState*) – random state
- **min\_bound** (*None, int*) – if not default of None, will be min bound when generating seed (inclusive). Must be less than max\_bound.

- **max\_bound** (*None, int*) – if not default of *None*, will be max bound when generating seed (exclusive). Must be greater than *min\_bound*.

**Returns** seed for random number generator

**Return type** *int*

`evalml.utils.gen_utils.get_random_state(seed)`

Generates a `numpy.random.RandomState` instance using *seed*.

**Parameters** *seed* (*None, int, np.random.RandomState object*) – *seed* to use to generate `numpy.random.RandomState`. Must be between `SEED_BOUNDS.min_bound` and `SEED_BOUNDS.max_bound`, inclusive. Otherwise, an exception will be thrown.

`evalml.utils.gen_utils.import_or_raise(library, error_msg=None, warning=False)`

Attempts to import the requested library by name. If the import fails, raises an `ImportError` or warning.

**Parameters**

- **library** (*str*) – the name of the library
- **error\_msg** (*str*) – error message to return if the import fails
- **warning** (*bool*) – if *True*, `import_or_raise` gives a warning instead of `ImportError`. Defaults to *False*.

`evalml.utils.gen_utils.is_all_numeric(df)`

Checks if the given `DataFrame` contains only numeric values

**Parameters** *df* (*pd.DataFrame*) – The `DataFrame` to check data types of.

**Returns** *True* if all the columns are numeric and are not missing any values, *False* otherwise.

`evalml.utils.gen_utils.jupyter_check()`

Get whether or not the code is being run in a `IPython` environment (such as `Jupyter Notebook` or `Jupyter Lab`)

**Parameters** *None* –

**Returns** *True* if `IPython`, *False* otherwise

**Return type** *Boolean*

`evalml.utils.gen_utils.logger`

`evalml.utils.gen_utils.pad_with_nans(pd_data, num_to_pad)`

Pad the beginning *num\_to\_pad* rows with nans.

**Parameters** *pd\_data* (*pd.DataFrame or pd.Series*) – Data to pad.

**Returns** *pd.DataFrame or pd.Series*

`evalml.utils.gen_utils.safe_repr(value)`

Convert the given value into a string that can safely be used for repr

**Parameters** *value* – the item to convert

**Returns** String representation of the value

`evalml.utils.gen_utils.save_plot(fig, filepath=None, format='png', interactive=False, return_filepath=False)`

Saves *fig* to *filepath* if specified, or to a default location if not.

**Parameters**

- **fig** (*Figure*) – Figure to be saved.
- **filepath** (*str or Path, optional*) – Location to save file. Default is with filename “test\_plot”.

- **format** (*str*) – Extension for figure to be saved as. Ignored if interactive is True and fig
- **of type** `plotly.Figure`. Defaults to 'png'. (*is*) –
- **interactive** (*bool*, *optional*) – If True and fig is of type `plotly.Figure`, saves the fig as interactive
- **of static** (*instead*) –
- **format will be set to** 'html'. Defaults to False. (*and*) –
- **return\_filepath** (*bool*, *optional*) – Whether to return the final filepath the image is saved to. Defaults to False.

**Returns** String representing the final filepath the image was saved to if `return_filepath` is set to True. Defaults to None.

`evalml.utils.gen_utils.SEED_BOUNDS`

## logger

### Module Contents

### Functions

---

<code>get_logger</code>	
<code>log_subtitle</code>	
<code>log_title</code>	
<code>time_elapsed</code>	How much time has elapsed since the search started.

---

### Contents

`evalml.utils.logger.get_logger` (*name*)

`evalml.utils.logger.log_subtitle` (*logger*, *title*, *underline*='=')

`evalml.utils.logger.log_title` (*logger*, *title*)

`evalml.utils.logger.time_elapsed` (*start\_time*)

How much time has elapsed since the search started.

**Parameters** `start_time` (*int*) – Time when search started.

**Returns** elapsed time formatted as a string [H:]MM:SS

**Return type** `str`

## update\_checker

### Module Contents

#### Contents

`evalml.utils.update_checker.method`

## woodwork\_utils

### Module Contents

#### Functions

---

<code>infer_feature_types</code>	Create a Woodwork structure from the given list, pandas, or numpy input, with specified types for columns.
----------------------------------	--

---

### Attributes Summary

---

`numeric_and_boolean_ww`

---

#### Contents

`evalml.utils.woodwork_utils.infer_feature_types` (*data*, *feature\_types=None*)

**Create a Woodwork structure from the given list, pandas, or numpy input, with specified types for columns.**

If a column's type is not specified, it will be inferred by Woodwork.

#### Parameters

- **data** (*pd.DataFrame*, *pd.Series*) – Input data to convert to a Woodwork data structure.
- **feature\_types** (*string*, *ww.logical\_type obj*, *dict*, *optional*) – If data is a 2D structure, feature\_types must be a dictionary mapping column names to the type of data represented in the column. If data is a 1D structure, then feature\_types must be a Woodwork logical type or a string representing a Woodwork logical type (“Double”, “Integer”, “Boolean”, “Categorical”, “Datetime”, “NaturalLanguage”)

**Returns** A Woodwork data structure where the data type of each column was either specified or inferred.

`evalml.utils.woodwork_utils.numeric_and_boolean_ww`

## Package Contents

### Classes Summary

---

<code>classproperty</code>	Allows function to be accessed as a class level property.
----------------------------	---

---

### Functions

---

<code>convert_to_seconds</code>	Converts a string describing a length of time to its length in seconds.
<code>deprecate_arg</code>	Helper to raise warnings when a deprecated arg is used.
<code>drop_rows_with_nans</code>	Drop rows that have any NaNs in all dataframes or series.
<code>get_importable_subclasses</code>	Get importable subclasses of a base class. Used to list all of our
<code>get_logger</code>	
<code>get_random_seed</code>	Given a <code>numpy.random.RandomState</code> object, generate an int representing a seed value for another random number generator. Or, if given an int, return that int.
<code>get_random_state</code>	Generates a <code>numpy.random.RandomState</code> instance using seed.
<code>import_or_raise</code>	Attempts to import the requested library by name.
<code>infer_feature_types</code>	Create a Woodwork structure from the given list, pandas, or numpy input, with specified types for columns.
<code>is_all_numeric</code>	Checks if the given DataFrame contains only numeric values
<code>jupyter_check</code>	Get whether or not the code is being run in a Jupyter environment (such as Jupyter Notebook or Jupyter Lab)
<code>log_subtitle</code>	
<code>log_title</code>	
<code>pad_with_nans</code>	Pad the beginning <code>num_to_pad</code> rows with nans.
<code>safe_repr</code>	Convert the given value into a string that can safely be used for repr
<code>save_plot</code>	Saves fig to filepath if specified, or to a default location if not.

---

### Attributes Summary

---

<code>SEED_BOUNDS</code>
--------------------------

---

## Contents

**class** evalml.utils.**classproperty** (*func*)

Allows function to be accessed as a class level property.

Example:

```
class LogisticRegressionBinaryPipeline(PipelineBase):
    component_graph = ['Simple Imputer', 'Logistic Regression Classifier']

    @classproperty
    def summary(cls):
        summary = ""
        for component in cls.component_graph:
            component = handle_component_class(component)
            summary += component.name + " + "
        return summary

assert LogisticRegressionBinaryPipeline.summary == "Simple Imputer + Logistic_
↪Regression Classifier + "
assert LogisticRegressionBinaryPipeline().summary == "Simple Imputer + Logistic_
↪Regression Classifier + "
```

evalml.utils.**convert\_to\_seconds** (*input\_str*)

Converts a string describing a length of time to its length in seconds.

evalml.utils.**deprecate\_arg** (*old\_arg, new\_arg, old\_value, new\_value*)

Helper to raise warnings when a deprecated arg is used.

### Parameters

- **old\_arg** (*str*) – Name of old/deprecated argument.
- **new\_arg** (*str*) – Name of new argument.
- **old\_value** (*Any*) – Value the user passed in for the old argument.
- **new\_value** (*Any*) – Value the user passed in for the new argument.

**Returns** old\_value if not None, else new\_value

evalml.utils.**drop\_rows\_with\_nans** (*\*pd\_data*)

Drop rows that have any NaNs in all dataframes or series.

**Parameters** *\*pd\_data* (*sequence of pd.Series or pd.DataFrame or None*) –

**Returns** list of pd.DataFrame or pd.Series or None

evalml.utils.**get\_importable\_subclasses** (*base\_class, used\_in\_automl=True*)

Get importable subclasses of a base class. Used to list all of our estimators, transformers, components and pipelines dynamically.

### Parameters

- **base\_class** (*abc.ABCMeta*) – Base class to find all of the subclasses for.
- **args** (*list*) – Args used to instantiate the subclass. [{}] for a pipeline, and [] for all other classes.
- **used\_in\_automl** – Not all components/pipelines/estimators are used in automl search. If True, only include those subclasses that are used in the search. This would mean excluding classes related to ExtraTrees, ElasticNet, and Baseline estimators.

**Returns** List of subclasses.

```
evalml.utils.get_logger(name)
```

```
evalml.utils.get_random_seed(random_state, min_bound=SEED_BOUNDS.min_bound,  
                             max_bound=SEED_BOUNDS.max_bound)
```

Given a `numpy.random.RandomState` object, generate an `int` representing a seed value for another random number generator. Or, if given an `int`, return that `int`.

To protect against invalid input to a particular library's random number generator, if an `int` value is provided, and it is outside the bounds "[`min_bound`, `max_bound`)", the value will be projected into the range between the `min_bound` (inclusive) and `max_bound` (exclusive) using modular arithmetic.

#### Parameters

- **random\_state** (*int*, *numpy.random.RandomState*) – random state
- **min\_bound** (*None*, *int*) – if not default of *None*, will be min bound when generating seed (inclusive). Must be less than `max_bound`.
- **max\_bound** (*None*, *int*) – if not default of *None*, will be max bound when generating seed (exclusive). Must be greater than `min_bound`.

**Returns** seed for random number generator

**Return type** `int`

```
evalml.utils.get_random_state(seed)
```

Generates a `numpy.random.RandomState` instance using seed.

**Parameters** **seed** (*None*, *int*, *np.random.RandomState object*) – seed to use to generate `numpy.random.RandomState`. Must be between `SEED_BOUNDS.min_bound` and `SEED_BOUNDS.max_bound`, inclusive. Otherwise, an exception will be thrown.

```
evalml.utils.import_or_raise(library, error_msg=None, warning=False)
```

Attempts to import the requested library by name. If the import fails, raises an `ImportError` or warning.

#### Parameters

- **library** (*str*) – the name of the library
- **error\_msg** (*str*) – error message to return if the import fails
- **warning** (*bool*) – if *True*, `import_or_raise` gives a warning instead of `ImportError`. Defaults to *False*.

```
evalml.utils.infer_feature_types(data, feature_types=None)
```

**Create a Woodwork structure from the given list, pandas, or numpy input, with specified types for columns.**

If a column's type is not specified, it will be inferred by Woodwork.

#### Parameters

- **data** (*pd.DataFrame*, *pd.Series*) – Input data to convert to a Woodwork data structure.
- **feature\_types** (*string*, *ww.logical\_type obj*, *dict*, *optional*) – If data is a 2D structure, `feature_types` must be a dictionary mapping column names to the type of data represented in the column. If data is a 1D structure, then `feature_types` must be a Woodwork logical type or a string representing a Woodwork logical type ("Double", "Integer", "Boolean", "Categorical", "Datetime", "NaturalLanguage")

**Returns** A Woodwork data structure where the data type of each column was either specified or inferred.



`evalml.utils.is_all_numeric(df)`

Checks if the given DataFrame contains only numeric values

**Parameters** `df` (*pd.DataFrame*) – The DataFrame to check data types of.

**Returns** True if all the columns are numeric and are not missing any values, False otherwise.

`evalml.utils.jupyter_check()`

Get whether or not the code is being run in a Ipython environment (such as Jupyter Notebook or Jupyter Lab)

**Parameters** None –

**Returns** True if Ipython, False otherwise

**Return type** Boolean

`evalml.utils.log_subtitle(logger, title, underline='')`

`evalml.utils.log_title(logger, title)`

`evalml.utils.pad_with_nans(pd_data, num_to_pad)`

Pad the beginning num\_to\_pad rows with nans.

**Parameters** `pd_data` (*pd.DataFrame* or *pd.Series*) – Data to pad.

**Returns** *pd.DataFrame* or *pd.Series*

`evalml.utils.safe_repr(value)`

Convert the given value into a string that can safely be used for repr

**Parameters** `value` – the item to convert

**Returns** String representation of the value

`evalml.utils.save_plot(fig, filepath=None, format='png', interactive=False, return_filepath=False)`

Saves fig to filepath if specified, or to a default location if not.

**Parameters**

- **fig** (*Figure*) – Figure to be saved.
- **filepath** (*str* or *Path*, *optional*) – Location to save file. Default is with filename “test\_plot”.
- **format** (*str*) – Extension for figure to be saved as. Ignored if interactive is True and fig
- **of type** `plotly.Figure`. Defaults to 'png'. (*is*) –
- **interactive** (*bool*, *optional*) – If True and fig is of type `plotly.Figure`, saves the fig as interactive
- **of static** (*instead*) –
- **format will be set to 'html'. Defaults to False.** (*and*) –
- **return\_filepath** (*bool*, *optional*) – Whether to return the final filepath the image is saved to. Defaults to False.

**Returns** String representing the final filepath the image was saved to if `return_filepath` is set to True. Defaults to None.

`evalml.utils.SEED_BOUNDS`

## Package Contents

### Classes Summary

---

<i>AutoMLSearch</i>	Automated Pipeline search.
---------------------	----------------------------

---

### Functions

---

<i>search</i>	Given data and configuration, run an automl search.
---------------	---

---

### Contents

```
class evalml.AutoMLSearch(X_train=None, y_train=None, problem_type=None, objective='auto',  
                           max_iterations=None, max_time=None, patience=None, toler-  
                           ance=None, data_splitter=None, allowed_component_graphs=None,  
                           allowed_model_families=None, start_iteration_callback=None,  
                           add_result_callback=None, error_callback=None, addi-  
                           tional_objectives=None, alternate_thresholding_objective='F1',  
                           random_seed=0, n_jobs=-1, tuner_class=None, opti-  
                           mize_thresholds=True, ensembling=False, max_batches=None,  
                           problem_configuration=None, train_best_pipeline=True,  
                           pipeline_parameters=None, custom_hyperparameters=None,  
                           sampler_method='auto', sampler_balanced_ratio=0.25, _ensem-  
                           bling_split_size=0.2, _pipelines_per_batch=5, engine=None)
```

Automated Pipeline search.

#### Parameters

- **X\_train** (*pd.DataFrame*) – The input training data of shape [n\_samples, n\_features]. Required.
- **y\_train** (*pd.Series*) – The target training data of length [n\_samples]. Required for supervised learning tasks.
- **problem\_type** (*str* or *ProblemTypes*) – type of supervised learning problem. See `evalml.problem_types.ProblemType.all_problem_types` for a full list.
- **objective** (*str*, *ObjectiveBase*) – The objective to optimize for. Used to propose and rank pipelines, but not for optimizing each pipeline during fit-time. When set to ‘auto’, chooses:
  - LogLossBinary for binary classification problems,
  - LogLossMulticlass for multiclass classification problems, and
  - R2 for regression problems.
- **max\_iterations** (*int*) – Maximum number of iterations to search. If `max_iterations` and `max_time` is not set, then `max_iterations` will default to `max_iterations` of 5.
- **max\_time** (*int*, *str*) – Maximum time to search for pipelines. This will not start a new pipeline search after the duration has elapsed. If it is an integer, then the time will be in seconds. For strings, time can be specified as seconds, minutes, or hours.

- **patience** (*int*) – Number of iterations without improvement to stop search early. Must be positive. If None, early stopping is disabled. Defaults to None.
- **tolerance** (*float*) – Minimum percentage difference to qualify as score improvement for early stopping. Only applicable if patience is not None. Defaults to None.
- **allowed\_component\_graphs** (*dict*) – A dictionary of lists or ComponentGraphs indicating the component graphs allowed in the search. The format should follow { “Name\_0”: [list\_of\_components], “Name\_1”: [ComponentGraph(...)] }

The default of None indicates all pipeline component graphs for this problem type are allowed. Setting this field will cause allowed\_model\_families to be ignored.

e.g. allowed\_component\_graphs = { “My\_Graph”: [“Imputer”, “One Hot Encoder”, “Random Forest Classifier”] }

- **allowed\_model\_families** (*list(str, ModelFamily)*) – The model families to search. The default of None searches over all model families. Run evalml.pipelines.components.utils.allowed\_model\_families(“binary”) to see options. Change *binary* to *multiclass* or *regression* depending on the problem type. Note that if allowed\_pipelines is provided, this parameter will be ignored.
- **data\_splitter** (*sklearn.model\_selection.BaseCrossValidator*) – Data splitting method to use. Defaults to StratifiedKFold.
- **tuner\_class** – The tuner class to use. Defaults to SKOptTuner.
- **optimize\_thresholds** (*bool*) – Whether or not to optimize the binary pipeline threshold. Defaults to True.
- **start\_iteration\_callback** (*callable*) – Function called before each pipeline training iteration. Callback function takes three positional parameters: The pipeline instance and the AutoMLSearch object.
- **add\_result\_callback** (*callable*) – Function called after each pipeline training iteration. Callback function takes three positional parameters: A dictionary containing the training results for the new pipeline, an untrained\_pipeline containing the parameters used during training, and the AutoMLSearch object.
- **error\_callback** (*callable*) – Function called when *search()* errors and raises an Exception. Callback function takes three positional parameters: the Exception raised, the traceback, and the AutoMLSearch object. Must also accepts kwargs, so AutoMLSearch is able to pass along other appropriate parameters by default. Defaults to None, which will call *log\_error\_callback*.
- **additional\_objectives** (*list*) – Custom set of objectives to score on. Will override default objectives for problem type if not empty.
- **alternate\_thresholding\_objective** (*str*) – The objective to use for thresholding binary classification pipelines if the main objective provided isn’t tuneable. Defaults to F1.
- **random\_seed** (*int*) – Seed for the random number generator. Defaults to 0.
- **n\_jobs** (*int or None*) – Non-negative integer describing level of parallelism used for pipelines. None and 1 are equivalent. If set to -1, all CPUs are used. For n\_jobs below -1, (n\_cpus + 1 + n\_jobs) are used.
- **ensembling** (*boolean*) – If True, runs ensembling in a separate batch after every allowed pipeline class has been iterated over. If the number of unique pipelines to search over per batch is one, ensembling will not run. Defaults to False.

- **max\_batches** (*int*) – The maximum number of batches of pipelines to search. Parameters `max_time`, and `max_iterations` have precedence over stopping the search.
- **problem\_configuration** (*dict*, *None*) – Additional parameters needed to configure the search. For example, in time series problems, values should be passed in for the `date_index`, `gap`, and `max_delay` variables.
- **train\_best\_pipeline** (*boolean*) – Whether or not to train the best pipeline before returning it. Defaults to `True`.
- **pipeline\_parameters** (*dict*) – A dict of the parameters used to initialize a pipeline with. Keys should consist of the component names and values should specify parameter values  
e.g. `pipeline_parameters = { 'Imputer' : { 'numeric_impute_strategy': 'most_frequent' } }`
- **custom\_hyperparameters** (*dict*) – A dict of the hyperparameter ranges used to iterate over during search. Keys should consist of the component names and values should specify a singular value or `skopt.Space`.  
e.g. `custom_hyperparameters = { 'Imputer' : { 'numeric_impute_strategy': Categorical(['most_frequent', 'median']) } }`
- **sampler\_method** (*str*) – The data sampling component to use in the pipelines if the problem type is classification and the target balance is smaller than the `sampler_balanced_ratio`. Either `'auto'`, which will use our preferred sampler for the data, `'Undersampler'`, `'Oversampler'`, or `None`. Defaults to `'auto'`.
- **sampler\_balanced\_ratio** (*float*) – The minority:majority class ratio that we consider balanced, so a 1:4 ratio would be equal to 0.25. If the class balance is larger than this provided value, then we will not add a sampler since the data is then considered balanced. Overrides the `sampler_ratio` of the samplers. Defaults to 0.25.
- **\_ensembling\_split\_size** (*float*) – The amount of the training data we'll set aside for training ensemble metalearners. Only used when `ensembling` is `True`. Must be between 0 and 1, exclusive. Defaults to 0.2
- **pipelines\_per\_batch** (*int*) – The number of pipelines to train for every batch after the first one. The first batch will train a baseline pipeline + one of each pipeline family allowed in the search.
- **engine** (*EngineBase* or *None*) – The engine instance used to evaluate pipelines. If `None`, a `SequentialEngine` will be used.

## Methods

<code>add_to_rankings</code>	Fits and evaluates a given pipeline then adds the results to the automl rankings with the requirement that automl search has been run.
<code>best_pipeline</code>	Returns a trained instance of the best pipeline and parameters found during automl search. If <code>train_best_pipeline</code> is set to <code>False</code> , returns an untrained pipeline instance.
<code>describe_pipeline</code>	Describe a pipeline
<code>full_rankings</code>	Returns a <code>pandas.DataFrame</code> with scoring results from all pipelines searched
<code>get_pipeline</code>	Given the ID of a pipeline training result, returns an untrained instance of the specified pipeline

continues on next page

Table 727 – continued from previous page

<code>load</code>	Loads AutoML object at file path
<code>plot</code>	
<code>rankings</code>	Returns a pandas.DataFrame with scoring results from the highest-scoring set of parameters used with each pipeline.
<code>results</code>	Class that allows access to a copy of the results from <code>automl_search</code> .
<code>save</code>	Saves AutoML object at file path
<code>score_pipelines</code>	Score a list of pipelines on the given holdout data.
<code>search</code>	Find the best pipeline for the data set.
<code>train_pipelines</code>	Train a list of pipelines on the training data.

**add\_to\_rankings** (*self*, *pipeline*)

Fits and evaluates a given pipeline then adds the results to the automl rankings with the requirement that automl search has been run.

**Parameters** **pipeline** (*PipelineBase*) – pipeline to train and evaluate.

**property best\_pipeline** (*self*)

Returns a trained instance of the best pipeline and parameters found during automl search. If `train_best_pipeline` is set to False, returns an untrained pipeline instance.

**Returns** A trained instance of the best pipeline and parameters found during automl search. If `train_best_pipeline` is set to False, returns an untrained pipeline instance.

**Return type** PipelineBase

**describe\_pipeline** (*self*, *pipeline\_id*, *return\_dict=False*)

Describe a pipeline

**Parameters**

- **pipeline\_id** (*int*) – pipeline to describe
- **return\_dict** (*bool*) – If True, return dictionary of information about pipeline. Defaults to False.

**Returns** Description of specified pipeline. Includes information such as type of pipeline components, problem, training time, cross validation, etc.

**property full\_rankings** (*self*)

Returns a pandas.DataFrame with scoring results from all pipelines searched

**get\_pipeline** (*self*, *pipeline\_id*)

Given the ID of a pipeline training result, returns an untrained instance of the specified pipeline initialized with the parameters used to train that pipeline during automl search.

**Parameters** **pipeline\_id** (*int*) – pipeline to retrieve

**Returns** untrained pipeline instance associated with the provided ID

**Return type** PipelineBase

**static load** (*file\_path*, *pickle\_type='cloudpickle'*)

Loads AutoML object at file path

**Parameters**

- **file\_path** (*str*) – location to find file to load

- **"pickle"** (*pickle\_type*) – the pickling library to use. Currently not used since the standard pickle library can handle cloudpickles.
- **"cloudpickle"** – the pickling library to use. Currently not used since the standard pickle library can handle cloudpickles.

**Returns** AutoSearchBase object

**property plot** (*self*)

**property rankings** (*self*)

Returns a pandas.DataFrame with scoring results from the highest-scoring set of parameters used with each pipeline.

**property results** (*self*)

Class that allows access to a copy of the results from *automl\_search*.

**Returns:** dict containing *pipeline\_results*: a dict with results from each pipeline, and *search\_order*: a list describing the order the pipelines were searched.

**save** (*self*, *file\_path*, *pickle\_type*='cloudpickle', *pickle\_protocol*=cloudpickle.DEFAULT\_PROTOCOL)

Saves AutoML object at file path

#### Parameters

- **file\_path** (*str*) – location to save file
- **"pickle"** (*pickle\_type*) – the pickling library to use.
- **"cloudpickle"** – the pickling library to use.
- **pickle\_protocol** (*int*) – the pickle data stream format.

**Returns** None

**score\_pipelines** (*self*, *pipelines*, *X\_holdout*, *y\_holdout*, *objectives*)

Score a list of pipelines on the given holdout data.

#### Parameters

- **pipelines** (*list (PipelineBase)*) – List of pipelines to train.
- **X\_holdout** (*pd.DataFrame*) – Holdout features.
- **y\_holdout** (*pd.Series*) – Holdout targets for scoring.
- **objectives** (*list (str)*, *list (ObjectiveBase)*) – Objectives used for scoring.

**Returns** Dictionary keyed by pipeline name that maps to a dictionary of scores. Note that the any pipelines that error out during scoring will not be included in the dictionary but the exception and stacktrace will be displayed in the log.

**Return type** Dict[str, Dict[str, float]]

**search** (*self*, *show\_iteration\_plot*=True)

Find the best pipeline for the data set.

#### Parameters

- **feature\_types** (*list*, *optional*) – list of feature types, either numerical or categorical. Categorical features will automatically be encoded
- **show\_iteration\_plot** (*boolean*, *True*) – Shows an iteration vs. score plot in Jupyter notebook. Disabled by default in non-Jupyter environments.

**train\_pipelines** (*self*, *pipelines*)

Train a list of pipelines on the training data.

This can be helpful for training pipelines once the search is complete.

**Parameters** *pipelines* (*list* (*PipelineBase*)) – List of pipelines to train.

**Returns** Dictionary keyed by pipeline name that maps to the fitted pipeline. Note that the any pipelines that error out during training will not be included in the dictionary but the exception and stacktrace will be displayed in the log.

**Return type** Dict[str, PipelineBase]

`evalml.search` (*X\_train=None*, *y\_train=None*, *problem\_type=None*, *objective='auto'*, *problem\_configuration=None*, *\*\*kwargs*)

Given data and configuration, run an automl search.

This method will run EvalML's default suite of data checks. If the data checks produce errors, the data check results will be returned before running the automl search. In that case we recommend you alter your data to address these errors and try again.

This method is provided for convenience. If you'd like more control over when each of these steps is run, consider making calls directly to the various pieces like the data checks and AutoMLSearch, instead of using this method.

#### Parameters

- **X\_train** (*pd.DataFrame*) – The input training data of shape [n\_samples, n\_features]. Required.
- **y\_train** (*pd.Series*) – The target training data of length [n\_samples]. Required for supervised learning tasks.
- **problem\_type** (*str* or *ProblemTypes*) – type of supervised learning problem. See `evalml.problem_types.ProblemType.all_problem_types` for a full list.
- **objective** (*str*, *ObjectiveBase*) – The objective to optimize for. Used to propose and rank pipelines, but not for optimizing each pipeline during fit-time. When set to 'auto', chooses:
  - LogLossBinary for binary classification problems,
  - LogLossMulticlass for multiclass classification problems, and
  - R2 for regression problems.
- **problem\_configuration** (*dict*) – Additional parameters needed to configure the search. For example,
- **time series problems** (*in*) –
- **should be passed in for the date\_index** (*values*) –
- **gap** –
- **max\_delay variables.** (*and*) –

Other keyword arguments which are provided will be passed to AutoMLSearch.

**Returns** the automl search object containing pipelines and rankings, and the results from running the data checks. If the data check results contain errors, automl search will not be run and an automl search object will not be returned.

**Return type** (*AutoMLSearch*, dict)





## RELEASE NOTES

### Future Releases

- Enhancements
- Fixes
- Changes
- Documentation Changes
- Testing Changes

<b>Warning: Breaking Changes</b>
----------------------------------

### v0.30.2 Aug. 16, 2021

- Fixes
  - Updated changelog and version numbers to match the release. Release 0.30.1 was release erroneously without a change to the version numbers. 0.30.2 replaces it.

### v0.30.1 Aug. 12, 2021

- Enhancements
  - Added `DatetimeFormatDataCheck` for time series problems [#2603](#)
  - Added `ProphetRegressor` to estimators [#2242](#)
  - Updated `ComponentGraph` to handle not calling samplers' transform during predict, and updated samplers' transform methods s.t. `fit_transform` is equivalent to `fit(X, y).transform(X, y)` [#2583](#)
  - Updated `ComponentGraph_validate_component_dict` logic to be stricter about input values [#2599](#)
  - Patched bug in `xgboost` estimators where predicting on a feature matrix of only booleans would throw an exception. [#2602](#)
  - Updated `ARIMARegressor` to use relative forecasting to predict values [#2613](#)
  - Updated to support `Woodwork 0.5.1` [#2610](#)
- Fixes
  - Updated `get_best_sampler_for_data` to consider all non-numeric datatypes as categorical for `SMOTE` [#2590](#)
  - Fixed inconsistent test results from `TargetDistributionDataCheck` [#2608](#)

- Adopted vectorized pd.NA checking for Woodwork 0.5.1 support [#2626](#)
- Pinned upper version of astroid to 2.6.6 to keep ReadTheDocs working. [#2638](#)
- **Changes**
  - Renamed SMOTE samplers to SMOTE oversampler [#2595](#)
  - Changed `partial_dependence` and `graph_partial_dependence` to raise a `PartialDependenceError` instead of `ValueError`. This is not a breaking change because `PartialDependenceError` is a subclass of `ValueError` [#2604](#)
  - Cleaned up code duplication in `ComponentGraph` [#2612](#)
- **Documentation Changes**
  - To avoid local docs build error, only add warning disable and download headers on ReadTheDocs builds, not locally [#2617](#)
- **Testing Changes**
  - Updated `partial_dependence` tests to change the element-wise comparison per the Plotly 5.2.1 upgrade [#2638](#)
  - Changed the lint CI job to only check against python 3.9 via the `-t` flag [#2586](#)
  - Installed Prophet in linux nightlies test and fixed `test_all_components` [#2598](#)
  - Refactored and fixed all `make_pipeline` tests to assert correct order and address new Woodwork Unknown type inference [#2572](#)
  - Removed `component_graphs` as a global variable in `test_component_graphs.py` [#2609](#)

**Warning:**

**Breaking Changes**

- Renamed SMOTE samplers to SMOTE oversampler. Please use `SMOTEOversampler`, `SMOTENCOverSampler`, `SMOTENOverSampler` instead of `SMOTESampler`, `SMOTENCSampler`, and `SMOTENSampler` [#2595](#)

**v0.30.0 Aug. 3, 2021**

- **Enhancements**
  - Added `LogTransformer` and `TargetDistributionDataCheck` [#2487](#)
  - Issue a warning to users when a pipeline parameter passed in isn't used in the pipeline [#2564](#)
  - Added Gini coefficient as an objective [#2544](#)
  - Added `repr` to `ComponentGraph` [#2565](#)
  - Added components to extract features from URL and EmailAddress Logical Types [#2550](#)
  - Added support for `NaN` values in `TextFeaturizer` [#2532](#)
  - Added `SelectByType` transformer [#2531](#)
  - Added separate thresholds for percent null rows and columns in `HighlyNullDataCheck` [#2562](#)
  - Added support for `NaN` natural language values [#2577](#)
- **Fixes**

- Raised error message for types `URL`, `NaturalLanguage`, and `EmailAddress` in `partial_dependence` [#2573](#)

- **Changes**

- Updated `PipelineBase` implementation for creating pipelines from a list of components [#2549](#)
- Moved `get_hyperparameter_ranges` to `PipelineBase` class from `automl/utils` module [#2546](#)
- Renamed `ComponentGraph`'s `get_parents` to `get_inputs` [#2540](#)
- Removed `ComponentGraph.linearized_component_graph` and `ComponentGraph.from_list` [#2556](#)
- Updated `ComponentGraph` to enforce requiring `.x` and `.y` inputs for each component in the graph [#2563](#)
- Renamed existing ensembler implementation from `StackedEnsemblers` to `SklearnStackedEnsemblers` [#2578](#)

- **Documentation Changes**

- Added documentation for `DaskEngine` and `CFEngine` parallel engines [#2560](#)
- Improved detail of `TextFeaturizer` docstring and tutorial [#2568](#)

- **Testing Changes**

- Added test that makes sure `split_data` does not shuffle for time series problems [#2552](#)

**Warning:**

**Breaking Changes**

- Moved `get_hyperparameter_ranges` to `PipelineBase` class from `automl/utils` module [#2546](#)
- Renamed `ComponentGraph`'s `get_parents` to `get_inputs` [#2540](#)
- Removed `ComponentGraph.linearized_component_graph` and `ComponentGraph.from_list` [#2556](#)
- Updated `ComponentGraph` to enforce requiring `.x` and `.y` inputs for each component in the graph [#2563](#)

**v0.29.0 Jul. 21, 2021**

- **Enhancements**

- Updated 1-way partial dependence support for datetime features [#2454](#)
- Added details on how to fix error caused by broken `ww` schema [#2466](#)
- Added ability to use built-in pickle for saving `AutoMLSearch` [#2463](#)
- Updated our components and component graphs to use latest features of `ww` 0.4.1, e.g. `concat_columns` and `drop in-place`. [#2465](#)
- Added new, concurrent.futures based engine for parallel AutoML [#2506](#)
- Added support for new Woodwork `Unknown` type in `AutoMLSearch` [#2477](#)
- Updated our components with an attribute that describes if they modify features or targets and can be used in list API for pipeline initialization [#2504](#)

- Updated `ComponentGraph` to accept `X` and `y` as inputs #2507
- Removed unused `TARGET_BINARY_INVALID_VALUES` from `DataCheckMessageCode` enum and fixed formatting of objective documentation #2520
- Added `EvalMLAlgorithm` #2525
- Added support for `NaN` values in `TextFeaturizer` #2532
- **Fixes**
  - Fixed `FraudCost` objective and reverted threshold optimization method for binary classification to `Golden` #2450
  - Added custom exception message for partial dependence on features with scales that are too small #2455
  - Ensures the typing for `Ordinal` and `Datetime` ltypes are passed through `_retain_custom_types_and_initialize_woodwork` #2461
  - Updated to work with Pandas 1.3.0 #2442
  - Updated to work with sktime 0.7.0 #2499
- **Changes**
  - Updated XGBoost dependency to `>=1.4.2` #2484, #2498
  - Added a `DeprecationWarning` about deprecating the list API for `ComponentGraph` #2488
  - Updated `make_pipeline` for AutoML to create dictionaries, not lists, to initialize pipelines #2504
  - No longer installing `graphviz` on windows in our CI pipelines because release 0.17 breaks windows 3.7 #2516
- **Documentation Changes**
  - Moved docstrings from `__init__` to class pages, added missing docstrings for missing classes, and updated missing default values #2452
  - Build documentation with `sphinx-autoapi` #2458
  - Change `autoapi_ignore` to only ignore files in `evalml/tests/*` #2530
- **Testing Changes**
  - Fixed flaky dask tests #2471
  - Removed `shellcheck` action from `build_conda_pkg` action #2514
  - Added a `tmp_dir` fixture that deletes its contents after tests run #2505
  - Added a test that makes sure all pipelines in `AutoMLSearch` get the same data splits #2513
  - Condensed warning output in test logs #2521

**Warning:**

**Breaking Changes**

- `NaN` values in the *Natural Language* type are no longer supported by the `Imputer` with the pandas upgrade. #2477

v0.28.0 Jul. 2, 2021

- **Enhancements**
  - Added support for showing a Individual Conditional Expectations plot when graphing Partial Dependence [#2386](#)
  - Exposed `thread_count` for Catboost estimators as `n_jobs` parameter [#2410](#)
  - Updated Objectives API to allow for sample weighting [#2433](#)
- **Fixes**
  - Deleted unreachable line from `IterativeAlgorithm` [#2464](#)
- **Changes**
  - Pinned Woodwork version between 0.4.1 and 0.4.2 [#2460](#)
  - Updated psutils minimum version in requirements [#2438](#)
  - Updated `log_error_callback` to not include filepath in logged message [#2429](#)
- **Documentation Changes**
  - Sped up docs [#2430](#)
  - Removed mentions of `DataTable` and `DataColumn` from the docs [#2445](#)
- **Testing Changes**
  - Added slack integration for nightlies tests [#2436](#)
  - Changed `build_conda_pkg` CI job to run only when dependencies are updates [#2446](#)
  - Updated workflows to store pytest runtimes as test artifacts [#2448](#)
  - Added `AutoMLTestEnv` test fixture for making it easy to mock automl tests [#2406](#)

## v0.27.0 Jun. 22, 2021

- **Enhancements**
  - Adds force plots for prediction explanations [#2157](#)
  - Removed self-reference from `AutoMLSearch` [#2304](#)
  - Added support for nonlinear pipelines for `generate_pipeline_code` [#2332](#)
  - Added `inverse_transform` method to pipelines [#2256](#)
  - Add optional automatic update checker [#2350](#)
  - Added `search_order` to `AutoMLSearch`'s rankings and `full_rankings` tables [#2345](#)
  - Updated threshold optimization method for binary classification [#2315](#)
  - Updated demos to pull data from S3 instead of including demo data in package [#2387](#)
  - Upgrade woodwork version to v0.4.1 [#2379](#)
- **Fixes**
  - Preserve user-specified woodwork types throughout pipeline fit/predict [#2297](#)
  - Fixed `ComponentGraph` appending target to `final_component_features` if there is a component that returns both X and y [#2358](#)
  - Fixed partial dependence graph method failing on multiclass problems when the class labels are numeric [#2372](#)

- Added `thresholding_objective` argument to `AutoMLSearch` for binary classification problems #2320
- Added change for `k_neighbors` parameter in SMOTE Oversamplers to automatically handle small samples #2375
- Changed naming for Logistic Regression Classifier file #2399
- Pinned `pytest-timeout` to fix minimum dependence checker #2425
- Replaced Elastic Net Classifier base class with Logistic Regression to avoid NaN outputs #2420

- **Changes**

- Cleaned up `PipelineBase`'s `component_graph` and `_component_graph` attributes. Updated `PipelineBase` `__repr__` and added `__eq__` for `ComponentGraph` #2332
- Added and applied `black` linting package to the EvalML repo in place of `autopep8` #2306
- Separated `custom_hyperparameters` from pipelines and added them as an argument to `AutoMLSearch` #2317
- Replaced `allowed_pipelines` with `allowed_component_graphs` #2364
- Removed private method `_compute_features_during_fit` from `PipelineBase` #2359
- Updated `compute_order` in `ComponentGraph` to be a read-only property #2408
- Unpinned `PyZMQ` version in `requirements.txt` #2389
- Uncapping `LightGBM` version in `requirements.txt` #2405
- Updated minimum version of `plotly` #2415
- Removed `SensitivityLowAlert` objective from core objectives #2418

- **Documentation Changes**

- Fixed lead scoring weights in the demos documentation #2315
- Fixed start page code and description dataset naming discrepancy #2370

- **Testing Changes**

- Update minimum unit tests to run on all pull requests #2314
- Pass token to authorize uploading of codecov reports #2344
- Add `pytest-timeout`. All tests that run longer than 6 minutes will fail. #2374
- Separated the dask tests out into separate github action jobs to isolate dask failures. #2376
- Refactored dask tests #2377
- Added the combined dask/non-dask unit tests back and renamed the dask only unit tests. #2382
- Sped up unit tests and split into separate jobs #2365
- Change CI job names, run lint for python 3.9, run nightlies on python 3.8 at 3am EST #2395 #2398
- Set fail-fast to false for CI jobs that run for PRs #2402

**Warning:****Breaking Changes**

- *AutoMLSearch* will accept *allowed\_component\_graphs* instead of *allowed\_pipelines* #2364
- Removed *PipelineBase*'s *\_component\_graph* attribute. Updated *PipelineBase* *\_\_repr\_\_* and added *\_\_eq\_\_* for *ComponentGraph* #2332
- *pipeline\_parameters* will no longer accept *skopt.space* variables since hyperparameter ranges will now be specified through *custom\_hyperparameters* #2317

**v0.25.0 Jun. 01, 2021**• **Enhancements**

- Upgraded minimum *woodwork* to version 0.3.1. Previous versions will not be supported #2181
- Added a new callback parameter for *explain\_predictions\_best\_worst* #2308

• **Fixes**• **Changes**

- Deleted the *return\_pandas* flag from our demo data loaders #2181
- Moved *default\_parameters* to *ComponentGraph* from *PipelineBase* #2307

• **Documentation Changes**

- Updated the release procedure documentation #2230

• **Testing Changes**

- Ignoring *test\_saving\_png\_file* while building conda package #2323

**Warning:****Breaking Changes**

- Deleted the *return\_pandas* flag from our demo data loaders #2181
- Upgraded minimum *woodwork* to version 0.3.1. Previous versions will not be supported #2181
- Due to the weak-ref in *woodwork*, set the result of *infer\_feature\_types* to a variable before accessing *woodwork* #2181

**v0.24.2 May. 24, 2021**• **Enhancements**

- Added oversamplers to *AutoMLSearch* #2213 #2286
- Added dictionary input functionality for *Undersampler* component #2271
- Changed the default parameter values for *Elastic Net Classifier* and *Elastic Net Regressor* #2269
- Added dictionary input functionality for the *Oversampler* components #2288

• **Fixes**

- Set default *n\_jobs* to 1 for *StackedEnsembleClassifier* and *StackedEnsembleRegressor* until fix for text-based parallelism in *sklearn* stacking can be found #2295

- **Changes**
  - Updated `start_iteration_callback` to accept a pipeline instance instead of a pipeline class and no longer accept pipeline parameters as a parameter #2290
  - Refactored `calculate_permutation_importance` method and add per-column permutation importance method #2302
  - Updated logging information in `AutoMLSearch.__init__` to clarify pipeline generation #2263
- **Documentation Changes**
  - Minor changes to the release procedure #2230
- **Testing Changes**
  - Use codecov action to update coverage reports #2238
  - Removed MarkupSafe dependency version pin from requirements.txt and moved instead into RTD docs build CI #2261

**Warning:**

**Breaking Changes**

- Updated `start_iteration_callback` to accept a pipeline instance instead of a pipeline class and no longer accept pipeline parameters as a parameter #2290
- Moved `default_parameters` to `ComponentGraph` from `PipelineBase`. A pipeline's `default_parameters` is now accessible via `pipeline.component_graph.default_parameters` #2307

**v0.24.1 May. 16, 2021**

- **Enhancements**
  - Integrated `ARIMAREgressor` into `AutoML` #2009
  - Updated `HighlyNullDataCheck` to also perform a null row check #2222
  - Set `max_depth` to 1 in calls to `featuretools dfs` #2231
- **Fixes**
  - Removed data splitter sampler calls during training #2253
  - Set minimum required version for `pyzmq`, `colorama`, and `docutils` #2254
  - Changed `BaseSampler` to return `None` instead of `y` #2272
- **Changes**
  - Removed ensemble split and indices in `AutoMLSearch` #2260
  - Updated `pipeline repr()` and `generate_pipeline_code` to return pipeline instances without generating custom pipeline class #2227
- **Documentation Changes**
  - Capped Sphinx version under 4.0.0 #2244
- **Testing Changes**
  - Change number of cores for `pytest` from 4 to 2 #2266



- Add minimum dependency checker to generate minimum requirement files #2267
- Add unit tests with minimum dependencies #2277

**v0.24.0 May. 04, 2021****• Enhancements**

- Added `date_index` as a required parameter for TimeSeries problems #2217
- Have the `OneHotEncoder` return the transformed columns as booleans rather than floats #2170
- Added `Oversampler` transformer component to EvalML #2079
- Added `Undersampler` to `AutoMLSearch`, as well as arguments `_sampler_method` and `sampler_balanced_ratio` #2128
- Updated prediction explanations functions to allow pipelines with XGBoost estimators #2162
- Added partial dependence for datetime columns #2180
- Update precision-recall curve with positive label index argument, and fix for 2d predicted probabilities #2090
- Add `pct_null_rows` to `HighlyNullDataCheck` #2211
- Added a standalone `AutoML search` method for convenience, which runs data checks and then runs `automl` #2152
- Make the first batch of AutoML have a predefined order, with linear models first and complex models last #2223 #2225
- Added sampling dictionary support to `BalancedClassificationSampler` #2235

**• Fixes**

- Fixed partial dependence not respecting grid resolution parameter for numerical features #2180
- Enable prediction explanations for catboost for multiclass problems #2224

**• Changes**

- Deleted baseline pipeline classes #2202
- Reverting user specified date feature PR #2155 until `pmdarima` installation fix is found #2214
- Updated pipeline API to accept component graph and other class attributes as instance parameters. Old pipeline API still works but will not be supported long-term. #2091
- Removed all old `datasplitters` from EvalML #2193
- Deleted `make_pipeline_from_components` #2218

**• Documentation Changes**

- Renamed `dataset` to clarify that its gzipped but not a tarball #2183
- Updated documentation to use pipeline instances instead of pipeline subclasses #2195
- Updated contributing guide with a note about GitHub Actions permissions #2090
- Updated `automl` and model understanding user guides #2090

**• Testing Changes**

- Use machineFL user token for dependency update bot, and add more reviewers #2189

**Warning:****Breaking Changes**

- All `baseline` `pipeline` `classes` (`BaselineBinaryPipeline`, `BaselineMulticlassPipeline`, `BaselineRegressionPipeline`, etc.) have been deleted [#2202](#)
- Updated pipeline API to accept component graph and other class attributes as instance parameters. Old pipeline API still works but will not be supported long-term. Pipelines can now be initialized by specifying the component graph as the first parameter, and then passing in optional arguments such as `custom_name`, `parameters`, etc. For example, `BinaryClassificationPipeline(["Random Forest Classifier"], parameters={})`. [#2091](#)
- Removed all old datasplitters from EvalML [#2193](#)
- Deleted utility method `make_pipeline_from_components` [#2218](#)

**v0.23.0 Apr. 20, 2021****• Enhancements**

- Refactored `EngineBase` and `SequentialEngine` api. Adding `DaskEngine` [#1975](#).
- Added optional engine argument to `AutoMLSearch` [#1975](#)
- Added a warning about how time series support is still in beta when a user passes in a time series problem to `AutoMLSearch` [#2118](#)
- Added `NaturalLanguageNaNDataCheck` data check [#2122](#)
- Added `ValueError` to `partial_dependence` to prevent users from computing partial dependence on columns with all NaNs [#2120](#)
- Added standard deviation of cv scores to rankings table [#2154](#)

**• Fixes**

- Fixed `BalancedClassificationDataCVSplit`, `BalancedClassificationDataTVSplit`, and `BalancedClassificationSampler` to use `minority:majority ratio` instead of `majority:minority` [#2077](#)
- Fixed bug where two-way partial dependence plots with categorical variables were not working correctly [#2117](#)
- Fixed bug where hyperparameters were not displaying properly for pipelines with a list `component_graph` and duplicate components [#2133](#)
- Fixed bug where `pipeline_parameters` argument in `AutoMLSearch` was not applied to pipelines passed in as `allowed_pipelines` [#2133](#)
- Fixed bug where `AutoMLSearch` was not applying custom hyperparameters to pipelines with a list `component_graph` and duplicate components [#2133](#)

**• Changes**

- Removed `hyperparameter_ranges` from `Undersampler` and renamed `balanced_ratio` to `sampling_ratio` for samplers [#2113](#)
- Renamed `TARGET_BINARY_NOT_TWO_EXAMPLES_PER_CLASS` data check message code to `TARGET_MULTICLASS_NOT_TWO_EXAMPLES_PER_CLASS` [#2126](#)
- Modified one-way partial dependence plots of categorical features to display data with a bar plot [#2117](#)

- Renamed `score` column for `automl.rankings` as `mean_cv_score` #2135
- Remove ‘warning’ from docs tool output #2031

- **Documentation Changes**

- Fixed `conf.py` file #2112
- Added a sentence to the automl user guide stating that our support for time series problems is still in beta. #2118
- Fixed documentation demos #2139
- Update test badge in README to use GitHub Actions #2150

- **Testing Changes**

- Fixed `test_describe_pipeline` for pandas v1.2.4 #2129
- Added a GitHub Action for building the conda package #1870 #2148

**Warning:**

**Breaking Changes**

- Renamed `balanced_ratio` to `sampling_ratio` for the `BalancedClassificationDataCVSplit`, `BalancedClassificationDataTVSplit`, `BalancedClassificationSampler`, and `Undersampler` #2113
- Deleted the “errors” key from automl results #1975
- Deleted the `raise_and_save_error_callback` and the `log_and_save_error_callback` #1975
- Fixed `BalancedClassificationDataCVSplit`, `BalancedClassificationDataTVSplit`, and `BalancedClassificationSampler` to use `minority:majority` ratio instead of `majority:minority` #2077

**v0.22.0 Apr. 06, 2021**

- **Enhancements**

- Added a GitHub Action for `linux_unit_tests` #2013
- Added recommended actions for `InvalidTargetDataCheck`, updated `_make_component_list_from_actions` to address new action, and added `TargetImputer` component #1989
- Updated `AutoMLSearch._check_for_high_variance` to not emit `RuntimeWarning` #2024
- Added exception when pipeline passed to `explain_predictions` is a `Stacked Ensemble` pipeline #2033
- Added sensitivity at low alert rates as an objective #2001
- Added `Undersampler` transformer component #2030

- **Fixes**

- Updated Engine’s `train_batch` to apply undersampling #2038
- Fixed bug in where Time Series Classification pipelines were not encoding targets in `predict` and `predict_proba` #2040

- Fixed data splitting errors if target is float for classification problems [#2050](#)
- Pinned docutils to <0.17 to fix ReadtheDocs warning issues [#2088](#)
- **Changes**
  - Removed lists as acceptable hyperparameter ranges in `AutoMLSearch` [#2028](#)
  - Renamed “details” to “metadata” for data check actions [#2008](#)
- **Documentation Changes**
  - Catch and suppress warnings in documentation [#1991](#) [#2097](#)
  - Change spacing in `start.ipynb` to provide clarity for `AutoMLSearch` [#2078](#)
  - Fixed start code on README [#2108](#)
- **Testing Changes**

**v0.21.0 Mar. 24, 2021**

- **Enhancements**
  - Changed `AutoMLSearch` to default `optimize_thresholds` to `True` [#1943](#)
  - Added multiple oversampling and undersampling sampling methods as data splitters for imbalanced classification [#1775](#)
  - Added params to balanced classification data splitters for visibility [#1966](#)
  - Updated `make_pipeline` to not add `Imputer` if input data does not have numeric or categorical columns [#1967](#)
  - Updated `ClassImbalanceDataCheck` to better handle multiclass imbalances [#1986](#)
  - Added recommended actions for the output of data check’s `validate` method [#1968](#)
  - Added error message for `partial_dependence` when features are mostly the same value [#1994](#)
  - Updated `OneHotEncoder` to drop one redundant feature by default for features with two categories [#1997](#)
  - Added a `PolynomialDetrender` component [#1992](#)
  - Added `DateTimeNaNDataCheck` data check [#2039](#)
- **Fixes**
  - Changed best pipeline to train on the entire dataset rather than just ensemble indices for ensemble problems [#2037](#)
  - Updated binary classification pipelines to use objective decision function during scoring of custom objectives [#1934](#)
- **Changes**
  - Removed `data_checks` parameter, `data_check_results` and data checks logic from `AutoMLSearch` [#1935](#)
  - Deleted `random_state` argument [#1985](#)
  - Updated Woodwork version requirement to `v0.0.11` [#1996](#)
- **Documentation Changes**
- **Testing Changes**

- Removed `build_docs` CI job in favor of RTD GH builder [#1974](#)
- Added tests to confirm support for Python 3.9 [#1724](#)
- Added tests to support Dask AutoML/Engine [#1990](#)
- Changed `build_conda_pkg` job to use `latest_release_changes` branch in the feed-stock. [#1979](#)

**Warning:****Breaking Changes**

- Changed `AutoMLSearch` to default `optimize_thresholds` to `True` [#1943](#)
- Removed `data_checks` parameter, `data_check_results` and `data checks` logic from `AutoMLSearch`. To run the data checks which were previously run by default in `AutoMLSearch`, please call `DefaultDataChecks().validate(X_train, y_train)` or take a look at our documentation for more examples. [#1935](#)
- Deleted `random_state` argument [#1985](#)

**v0.20.0 Mar. 10, 2021**• **Enhancements**

- Added a GitHub Action for Detecting dependency changes [#1933](#)
- Create a separate CV split to train stacked ensembler on for `AutoMLSearch` [#1814](#)
- Added a GitHub Action for Linux unit tests [#1846](#)
- Added `ARIMAREgressor` estimator [#1894](#)
- Added `DataCheckAction` class and `DataCheckActionCode` enum [#1896](#)
- Updated Woodwork requirement to `v0.0.10` [#1900](#)
- Added `BalancedClassificationDataCVSplit` and `BalancedClassificationDataTVSplit` to `AutoMLSearch` [#1875](#)
- Update default classification data splitter to use downsampling for highly imbalanced data [#1875](#)
- Updated `describe_pipeline` to return more information, including `id` of pipelines used for ensemble models [#1909](#)
- Added utility method to create list of components from a list of `DataCheckAction` [#1907](#)
- Updated `validate` method to include a `action` key in returned dictionary for all `DataCheck` and `DataChecks` [#1916](#)
- Aggregating the shap values for predictions that we know the provenance of, e.g. OHE, text, and date-time. [#1901](#)
- Improved error message when custom objective is passed as a string in `pipeline.score` [#1941](#)
- Added `score_pipelines` and `train_pipelines` methods to `AutoMLSearch` [#1913](#)
- Added support for pandas version 1.2.0 [#1708](#)
- Added `score_batch` and `train_batch` abstract methods to `EngineBase` and implementations in `SequentialEngine` [#1913](#)
- Added ability to handle index columns in `AutoMLSearch` and `DataChecks` [#2138](#)

- **Fixes**

- Removed CI check for `check_dependencies_updated_linux` #1950
- Added metaclass for time series pipelines and fix binary classification pipeline `predict` not using objective if it is passed as a named argument #1874
- Fixed stack trace in prediction explanation functions caused by mixed string/numeric pandas column names #1871
- Fixed stack trace caused by passing pipelines with duplicate names to `AutoMLSearch` #1932
- Fixed `AutoMLSearch.get_pipelines` returning pipelines with the same attributes #1958

- **Changes**

- Reversed GitHub Action for Linux unit tests until a fix for report generation is found #1920
- Updated `add_results` in `AutoMLAlgorithm` to take in entire pipeline results dictionary from `AutoMLSearch` #1891
- Updated `ClassImbalanceDataCheck` to look for severe class imbalance scenarios #1905
- Deleted the `explain_prediction` function #1915
- Removed `HighVarianceCVDataCheck` and converted it to an `AutoMLSearch` method instead #1928
- Removed warning in `InvalidTargetDataCheck` returned when numeric binary classification targets are not (0, 1) #1959

- **Documentation Changes**

- Updated `model_understanding.ipynb` to demo the two-way partial dependence capability #1919

- **Testing Changes**

**Warning:****Breaking Changes**

- Deleted the `explain_prediction` function #1915
- Removed `HighVarianceCVDataCheck` and converted it to an `AutoMLSearch` method instead #1928
- Added `score_batch` and `train_batch` abstract methods to `EngineBase`. These need to be implemented in `Engine` subclasses #1913

**v0.19.0 Feb. 23, 2021**

- **Enhancements**

- Added a GitHub Action for Python windows unit tests #1844
- Added a GitHub Action for checking updated release notes #1849
- Added a GitHub Action for Python lint checks #1837
- Adjusted `explain_prediction`, `explain_predictions` and `explain_predictions_best_worst` to handle timeseries problems. #1818
- Updated `InvalidTargetDataCheck` to check for mismatched indices in target and features #1816

- Updated Woodwork structures returned from components to support Woodwork logical type overrides set by the user [#1784](#)
- Updated estimators to keep track of input feature names during `fit()` [#1794](#)
- Updated `visualize_decision_tree` to include feature names in output [#1813](#)
- Added `is_bounded_like_percentage` property for objectives. If true, the `calculate_percent_difference` method will return the absolute difference rather than relative difference [#1809](#)
- Added full error traceback to AutoMLSearch logger file [#1840](#)
- Changed TargetEncoder to preserve custom indices in the data [#1836](#)
- Refactored `explain_predictions` and `explain_predictions_best_worst` to only compute features once for all rows that need to be explained [#1843](#)
- Added custom random undersampler data splitter for classification [#1857](#)
- Updated OutliersDataCheck implementation to calculate the probability of having no outliers [#1855](#)
- Added Engines pipeline processing API [#1838](#)
- **Fixes**
  - Changed EngineBase `random_state` arg to `random_seed` and same for user guide docs [#1889](#)
- **Changes**
  - Modified `calculate_percent_difference` so that division by 0 is now `inf` rather than `nan` [#1809](#)
  - Removed `text_columns` parameter from LSA and TextFeaturizer components [#1652](#)
  - Added `random_seed` as an argument to our `automl/pipeline/component` API. Using `random_state` will raise a warning [#1798](#)
  - Added `DataCheckError` message in `InvalidTargetDataCheck` if input target is `None` and removed exception raised [#1866](#)
- **Documentation Changes**
- **Testing Changes**
  - Added back coverage for `_get_feature_provenance` in TextFeaturizer after `text_columns` was removed [#1842](#)
  - Pin graphviz version for windows builds [#1847](#)
  - Unpin graphviz version for windows builds [#1851](#)

**Warning:****Breaking Changes**

- Added a deprecation warning to `explain_prediction`. It will be deleted in the next release. [#1860](#)

v0.18.2 Feb. 10, 2021

- **Enhancements**
  - Added uniqueness score data check [#1785](#)

- Added “dataframe” output format for prediction explanations #1781
- Updated LightGBM estimators to handle `pandas.MultiIndex` #1770
- Sped up permutation importance for some pipelines #1762
- Added sparsity data check #1797
- Confirmed support for threshold tuning for binary time series classification problems #1803
- Fixes
- Changes
- **Documentation Changes**
  - Added section on conda to the contributing guide #1771
  - Updated release process to reflect freezing *main* before perf tests #1787
  - Moving some prs to the right section of the release notes #1789
  - Tweak README.md. #1800
  - Fixed back arrow on install page docs #1795
  - Fixed docstring for *ClassImbalanceDataCheck.validate()* #1817
- Testing Changes

**v0.18.1 Feb. 1, 2021**

- **Enhancements**
  - Added `graph_t_sne` as a visualization tool for high dimensional data #1731
  - Added the ability to see the linear coefficients of features in linear models terms #1738
  - Added support for `scikit-learn v0.24.0` #1733
  - Added support for `scipy v1.6.0` #1752
  - Added SVM Classifier and Regressor to estimators #1714 #1761
- **Fixes**
  - Addressed bug with `partial_dependence` and categorical data with more categories than grid resolution #1748
  - Removed `random_state` arg from `get_pipelines` in `AutoMLSearch` #1719
  - Pinned `pymzq` at less than 22.0.0 till we add support #1756
- **Changes**
  - Updated components and pipelines to return `Woodwork` data structures #1668
  - Updated `clone()` for pipelines and components to copy over random state automatically #1753
  - Dropped support for Python version 3.6 #1751
  - Removed deprecated `verbose` flag from `AutoMLSearch` parameters #1772
- **Documentation Changes**
  - Add Twitter and Github link to documentation toolbar #1754
  - Added Open Graph info to documentation #1758
- Testing Changes



**Warning:****Breaking Changes**

- Components and pipelines return `Woodwork` data structures instead of `pandas` data structures [#1668](#)
- Python 3.6 will not be actively supported due to discontinued support from EvalML dependencies.
- Deprecated `verbose` flag is removed for `AutoMLSearch` [#1772](#)

**v0.18.0 Jan. 26, 2021**

- **Enhancements**

- Added RMSLE, MSLE, and MAPE to core objectives while checking for negative target values in `invalid_targets_data_check` [#1574](#)
- Added validation checks for binary problems with regression-like datasets and multiclass problems without true multiclass targets in `invalid_targets_data_check` [#1665](#)
- Added time series support for `make_pipeline` [#1566](#)
- Added target name for output of pipeline `predict` method [#1578](#)
- Added multiclass check to `InvalidTargetDataCheck` for two examples per class [#1596](#)
- Added support for `graphviz` v0.16 [#1657](#)
- Enhanced time series pipelines to accept empty features [#1651](#)
- Added KNN Classifier to estimators. [#1650](#)
- Added support for list inputs for objectives [#1663](#)
- Added support for `AutoMLSearch` to handle time series classification pipelines [#1666](#)
- Enhanced `DelayedFeaturesTransformer` to encode categorical features and targets before delaying them [#1691](#)
- Added 2-way dependence plots. [#1690](#)
- Added ability to directly iterate through components within Pipelines [#1583](#)

- **Fixes**

- Fixed inconsistent attributes and added Exceptions to docs [#1673](#)
- Fixed `TargetLeakageDataCheck` to use `Woodwork` `mutual_information` rather than using `Pandas`’ `Pearson Correlation` [#1616](#)
- Fixed thresholding for pipelines in `AutoMLSearch` to only threshold binary classification pipelines [#1622](#) [#1626](#)
- Updated `load_data` to return `Woodwork` structures and update default parameter value for `index` to `None` [#1610](#)
- Pinned `scipy` at < 1.6.0 while we work on adding support [#1629](#)
- Fixed data check message formatting in `AutoMLSearch` [#1633](#)
- Addressed stacked ensemble component for `scikit-learn` v0.24 support by setting `shuffle=True` for default CV [#1613](#)
- Fixed bug where `Imputer` reset the index on `X` [#1590](#)
- Fixed `AutoMLSearch` `stacktrace` when a custom objective was passed in as a primary objective or additional objective [#1575](#)

- Fixed custom index bug for MAPE objective #1641
- Fixed index bug for TextFeaturizer and LSA components #1644
- Limited load\_fraud dataset loaded into automl.ipynb #1646
- add\_to\_rankings updates AutoMLSearch.best\_pipeline when necessary #1647
- Fixed bug where time series baseline estimators were not receiving gap and max\_delay in AutoMLSearch #1645
- Fixed jupyter notebooks to help the RTD buildtime #1654
- Added positive\_only objectives to non\_core\_objectives #1661
- Fixed stacking argument n\_jobs for IterativeAlgorithm #1706
- Updated CatBoost estimators to return self in .fit() rather than the underlying model for consistency #1701
- Added ability to initialize pipeline parameters in AutoMLSearch constructor #1676
- **Changes**
  - Added labeling to graph\_confusion\_matrix #1632
  - Rerunning search for AutoMLSearch results in a message thrown rather than failing the search, and removed has\_searched property #1647
  - Changed tuner class to allow and ignore single parameter values as input #1686
  - Capped LightGBM version limit to remove bug in docs #1711
  - Removed support for np.random.RandomState in EvalML #1727
- **Documentation Changes**
  - Update Model Understanding in the user guide to include visualize\_decision\_tree #1678
  - Updated docs to include information about AutoMLSearch callback parameters and methods #1577
  - Updated docs to prompt users to install graphviz on Mac #1656
  - Added infer\_feature\_types to the start.ipynb guide #1700
  - Added multicollinearity data check to API reference and docs #1707
- **Testing Changes**

**Warning:**

**Breaking Changes**

- Removed has\_searched property from AutoMLSearch #1647
- Components and pipelines return Woodwork data structures instead of pandas data structures #1668
- Removed support for np.random.RandomState in EvalML. Rather than passing np.random.RandomState as component and pipeline random\_state values, we use int random\_seed #1727

v0.17.0 Dec. 29, 2020

- **Enhancements**
  - Added save\_plot that allows for saving figures from different backends #1588

- Added LightGBM Regressor to regression components [#1459](#)
- Added `visualize_decision_tree` for tree visualization with `decision_tree_data_from_estimator` and `decision_tree_data_from_pipeline` to reformat tree structure output [#1511](#)
- Added *DFS Transformer* component into transformer components [#1454](#)
- Added MAPE to the standard metrics for time series problems and update objectives [#1510](#)
- Added `graph_prediction_vs_actual_over_time` and `get_prediction_vs_actual_over_time_data` to the model understanding module for time series problems [#1483](#)
- Added a `ComponentGraph` class that will support future pipelines as directed acyclic graphs [#1415](#)
- Updated data checks to accept Woodwork data structures [#1481](#)
- Added parameter to `InvalidTargetDataCheck` to show only top unique values rather than all unique values [#1485](#)
- Added multicollinearity data check [#1515](#)
- Added baseline pipeline and components for time series regression problems [#1496](#)
- Added more information to users about ensembling behavior in `AutoMLSearch` [#1527](#)
- Add woodwork support for more utility and graph methods [#1544](#)
- Changed `DateTimeFeaturizer` to encode features as int [#1479](#)
- Return trained pipelines from `AutoMLSearch.best_pipeline` [#1547](#)
- Added utility method so that users can set feature types without having to learn about Woodwork directly [#1555](#)
- Added Linear Discriminant Analysis transformer for dimensionality reduction [#1331](#)
- Added multiclass support for `partial_dependence` and `graph_partial_dependence` [#1554](#)
- Added `TimeSeriesBinaryClassificationPipeline` and `TimeSeriesMulticlassClassificationPipeline` classes [#1528](#)
- Added `make_data_splitter` method for easier automl data split customization [#1568](#)
- Integrated `ComponentGraph` class into Pipelines for full non-linear pipeline support [#1543](#)
- Update `AutoMLSearch` constructor to take training data instead of search and `add_to_leaderboard` [#1597](#)
- Update `split_data` helper args [#1597](#)
- Add problem type utils `is_regression`, `is_classification`, `is_timeseries` [#1597](#)
- Rename `AutoMLSearch` `data_split` arg to `data_splitter` [#1569](#)

- **Fixes**

- Fix AutoML not passing CV folds to `DefaultDataChecks` for usage by `ClassImbalanceDataCheck` [#1619](#)
- Fix Windows CI jobs: install numba via conda, required for shap [#1490](#)
- Added custom-index support for `reset-index-get_prediction_vs_actual_over_time_data` [#1494](#)

- Fix `generate_pipeline_code` to account for boolean and None differences between Python and JSON [#1524](#) [#1531](#)
- Set max value for plotly and xgboost versions while we debug CI failures with newer versions [#1532](#)
- Undo version pinning for plotly [#1533](#)
- Fix ReadTheDocs build by updating the version of setuptools [#1561](#)
- Set `random_state` of data splitter in `AutoMLSearch` to take int to keep consistency in the resulting splits [#1579](#)
- Pin sklearn version while we work on adding support [#1594](#)
- Pin pandas at <1.2.0 while we work on adding support [#1609](#)
- Pin graphviz at < 0.16 while we work on adding support [#1609](#)
- **Changes**
  - Reverting `save_graph` [#1550](#) to resolve kaleido build issues [#1585](#)
  - Update circleci badge to apply to main [#1489](#)
  - Added script to generate github markdown for releases [#1487](#)
  - Updated selection using pandas dtypes to selecting using Woodwork logical types [#1551](#)
  - Updated dependencies to fix `ImportError: cannot import name 'MaskedArray' from 'sklearn.utils.fixes'` error and to address Woodwork and Featuretools dependencies [#1540](#)
  - Made `get_prediction_vs_actual_data()` a public method [#1553](#)
  - Updated Woodwork version requirement to v0.0.7 [#1560](#)
  - Move data splitters from `evalml.automl.data_splitters` to `evalml.preprocessing.data_splitters` [#1597](#)
  - Rename “# Testing” in automl log output to “# Validation” [#1597](#)
- **Documentation Changes**
  - Added partial dependence methods to API reference [#1537](#)
  - Updated documentation for confusion matrix methods [#1611](#)
- **Testing Changes**
  - Set `n_jobs=1` in most unit tests to reduce memory [#1505](#)

**Warning:****Breaking Changes**

- Updated minimal dependencies: `numpy>=1.19.1`, `pandas>=1.1.0`, `scikit-learn>=0.23.1`, `scikit-optimize>=0.8.1`
- Updated `AutoMLSearch.best_pipeline` to return a trained pipeline. Pass in `train_best_pipeline=False` to `AutoMLSearch` in order to return an untrained pipeline.
- Pipeline component instances can no longer be iterated through using `Pipeline.component_graph` [#1543](#)

- Update `AutoMLSearch` constructor to take training data instead of search and `add_to_leaderboard` #1597
- Update `split_data` helper args #1597
- Move data splitters from `evalml.automl.data_splitters` to `evalml.preprocessing.data_splitters` #1597
- Rename `AutoMLSearch` `data_split` arg to `data_splitter` #1569

**v0.16.1 Dec. 1, 2020**• **Enhancements**

- Pin `woodwork` version to `v0.0.6` to avoid breaking changes #1484
- Updated `Woodwork` to `>=0.0.5` in `core-requirements.txt` #1473
- Removed `copy_dataframe` parameter for `Woodwork`, updated `Woodwork` to `>=0.0.6` in `core-requirements.txt` #1478
- Updated `detect_problem_type` to use `pandas.api.is_numeric_dtype` #1476

• **Changes**

- Changed `make_clean` to delete coverage reports as a convenience for developers #1464
- Set `n_jobs=-1` by default for stacked ensemble components #1472

• **Documentation Changes**

- Updated pipeline and component documentation and demos to use `Woodwork` #1466

• **Testing Changes**

- Update dependency update checker to use everything from core and optional dependencies #1480

**v0.16.0 Nov. 24, 2020**• **Enhancements**

- Updated pipelines and `make_pipeline` to accept `Woodwork` inputs #1393
- Updated components to accept `Woodwork` inputs #1423
- Added ability to freeze hyperparameters for `AutoMLSearch` #1284
- Added `Target Encoder` into transformer components #1401
- Added callback for error handling in `AutoMLSearch` #1403
- Added the index id to the `explain_predictions_best_worst` output to help users identify which rows in their data are included #1365
- The `top_k` features displayed in `explain_predictions_*` functions are now determined by the magnitude of shap values as opposed to the `top_k` largest and smallest shap values. #1374
- Added a problem type for time series regression #1386
- Added a `is_defined_for_problem_type` method to `ObjectiveBase` #1386
- Added a `random_state` parameter to `make_pipeline_from_components` function #1411
- Added `DelayedFeaturesTransformer` #1396
- Added a `TimeSeriesRegressionPipeline` class #1418

- Removed `core-requirements.txt` from the package distribution #1429
- Updated data check messages to include a “*code*” and “*details*” fields #1451, #1462
- Added a `TimeSeriesSplit` data splitter for time series problems #1441
- Added a `problem_configuration` parameter to `AutoMLSearch` #1457
- **Fixes**
  - Fixed `IndexError` raised in `AutoMLSearch` when `ensembling = True` but only one pipeline to iterate over #1397
  - Fixed stacked ensemble input bug and `LightGBM` warning and bug in `AutoMLSearch` #1388
  - Updated enum classes to show possible enum values as attributes #1391
  - Updated calls to `Woodwork`’s `to_pandas()` to `to_series()` and `to_dataframe()` #1428
  - Fixed bug in OHE where column names were not guaranteed to be unique #1349
  - Fixed bug with percent improvement of `ExpVariance` objective on data with highly skewed target #1467
  - Fix `SimpleImputer` error which occurs when all features are bool type #1215
- **Changes**
  - Changed `OutliersDataCheck` to return the list of columns, rather than rows, that contain outliers #1377
  - Simplified and cleaned output for Code Generation #1371
  - Reverted changes from #1337 #1409
  - Updated data checks to return dictionary of warnings and errors instead of a list #1448
  - Updated `AutoMLSearch` to pass `Woodwork` data structures to every pipeline (instead of pandas `DataFrames`) #1450
  - Update `AutoMLSearch` to default to `max_batches=1` instead of `max_iterations=5` #1452
  - Updated `_evaluate_pipelines` to consolidate side effects #1410
- **Documentation Changes**
  - Added description of CLA to contributing guide, updated description of draft PRs #1402
  - Updated documentation to include all data checks, `DataChecks`, and usage of data checks in `AutoML` #1412
  - Updated docstrings from `np.array` to `np.ndarray` #1417
  - Added section on stacking ensembles in `AutoMLSearch` documentation #1425
- **Testing Changes**
  - Removed `category_encoders` from `test-requirements.txt` #1373
  - Tweak `codecov.io` settings again to avoid flakes #1413
  - Modified `make lint` to check notebook versions in the docs #1431
  - Modified `make lint-fix` to standardize notebook versions in the docs #1431
  - Use new version of pull request Github Action for dependency check (#1443)

- Reduced number of workers for tests to 4 [#1447](#)

**Warning:****Breaking Changes**

- The `top_k` and `top_k_features` parameters in `explain_predictions_*` functions now return `k` features as opposed to `2 * k` features [#1374](#)
- Renamed `problem_type` to `problem_types` in `RegressionObjective`, `BinaryClassificationObjective`, and `MulticlassClassificationObjective` [#1319](#)
- Data checks now return a dictionary of warnings and errors instead of a list [#1448](#)

**v0.15.0 Oct. 29, 2020**• **Enhancements**

- Added stacked ensemble component classes (`StackedEnsembleClassifier`, `StackedEnsembleRegressor`) [#1134](#)
- Added stacked ensemble components to `AutoMLSearch` [#1253](#)
- Added `DecisionTreeClassifier` and `DecisionTreeRegressor` to `AutoML` [#1255](#)
- Added `graph_prediction_vs_actual` in `model_understanding` for regression problems [#1252](#)
- Added parameter to `OneHotEncoder` to enable filtering for features to encode for [#1249](#)
- Added `percent-better-than-baseline` for all objectives to `automl.results` [#1244](#)
- Added `HighVarianceCVDataCheck` and replaced synonymous warning in `AutoMLSearch` [#1254](#)
- Added *PCA Transformer* component for dimensionality reduction [#1270](#)
- Added `generate_pipeline_code` and `generate_component_code` to allow for code generation given a pipeline or component instance [#1306](#)
- Added *PCA Transformer* component for dimensionality reduction [#1270](#)
- Updated `AutoMLSearch` to support Woodwork data structures [#1299](#)
- Added `cv_folds` to `ClassImbalanceDataCheck` and added this check to `DefaultDataChecks` [#1333](#)
- Make `max_batches` argument to `AutoMLSearch.search` public [#1320](#)
- Added text support to `automl search` [#1062](#)
- Added `_pipelines_per_batch` as a private argument to `AutoMLSearch` [#1355](#)

• **Fixes**

- Fixed ML performance issue with ordered datasets: always shuffle data in `automl`'s default CV splits [#1265](#)
- Fixed broken `evalml info` CLI command [#1293](#)
- Fixed boosting `type='rf'` for `LightGBM Classifier`, as well as `num_leaves` error [#1302](#)
- Fixed bug in `explain_predictions_best_worst` where a custom index in the target variable would cause a `ValueError` [#1318](#)

- Added stacked ensemble estimators to `evalml.pipelines.__init__` file #1326
- Fixed bug in OHE where calls to transform were not deterministic if `top_n` was less than the number of categories in a column #1324
- Fixed LightGBM warning messages during AutoMLSearch #1342
- Fix warnings thrown during AutoMLSearch in `HighVarianceCVDDataCheck` #1346
- Fixed bug where `TrainingValidationSplit` would return invalid location indices for dataframes with a custom index #1348
- Fixed bug where the AutoMLSearch `random_state` was not being passed to the created pipelines #1321

- **Changes**

- Allow `add_to_rankings` to be called before AutoMLSearch is called #1250
- Removed Graphviz from test-requirements to add to requirements.txt #1327
- Removed `max_pipelines` parameter from AutoMLSearch #1264
- Include editable installs in all install make targets #1335
- Made pip dependencies *featuretools* and *nlp\_primitives* core dependencies #1062
- Removed *PartOfSpeechCount* from *TextFeaturizer* transform primitives #1062
- Added warning for `partial_dependency` when the feature includes null values #1352

- **Documentation Changes**

- Fixed and updated code blocks in Release Notes #1243
- Added DecisionTree estimators to API Reference #1246
- Changed class inheritance display to flow vertically #1248
- Updated cost-benefit tutorial to use a holdout/test set #1159
- Added `evalml info` command to documentation #1293
- Miscellaneous doc updates #1269
- Removed conda pre-release testing from the release process document #1282
- Updates to contributing guide #1310
- Added Alteryx footer to docs with Twitter and Github link #1312
- Added documentation for evalml installation for Python 3.6 #1322
- Added documentation changes to make the API Docs easier to understand #1323
- Fixed documentation for `feature_importance` #1353
- Added tutorial for running *AutoML* with text data #1357
- Added documentation for woodwork integration with automl search #1361

- **Testing Changes**

- Added tests for `jupyter_check` to handle IPython #1256
- Cleaned up `make_pipeline` tests to test for all estimators #1257
- Added a test to check conda build after merge to main #1247
- Removed code that was lacking codecov for `__main__.py` and unnecessary #1293



- Codecov: round coverage up instead of down [#1334](#)
- Add DockerHub credentials to CI testing environment [#1356](#)
- Add DockerHub credentials to conda testing environment [#1363](#)

**Warning:****Breaking Changes**

- Renamed `LabelLeakageDataCheck` to `TargetLeakageDataCheck` [#1319](#)
- `max_pipelines` parameter has been removed from `AutoMLSearch`. Please use `max_iterations` instead. [#1264](#)
- `AutoMLSearch.search()` will now log a warning if the input is not a Woodwork data structure (pandas, numpy) [#1299](#)
- Make `max_batches` argument to `AutoMLSearch.search` public [#1320](#)
- Removed unused argument `feature_types` from `AutoMLSearch.search` [#1062](#)

**v0.14.1 Sep. 29, 2020**• **Enhancements**

- Updated partial dependence methods to support calculating numeric columns in a dataset with non-numeric columns [#1150](#)
- Added `get_feature_names` on `OneHotEncoder` [#1193](#)
- Added `detect_problem_type` to `problem_type/utils.py` to automatically detect the problem type given targets [#1194](#)
- Added `LightGBM` to `AutoMLSearch` [#1199](#)
- Updated `scikit-learn` and `scikit-optimize` to use latest versions - 0.23.2 and 0.8.1 respectively [#1141](#)
- Added `__str__` and `__repr__` for pipelines and components [#1218](#)
- Included internal target check for both training and validation data in `AutoMLSearch` [#1226](#)
- Added `ProblemTypes.all_problem_types` helper to get list of supported problem types [#1219](#)
- Added `DecisionTreeClassifier` and `DecisionTreeRegressor` classes [#1223](#)
- Added `ProblemTypes.all_problem_types` helper to get list of supported problem types [#1219](#)
- `DataChecks` can now be parametrized by passing a list of `DataCheck` classes and a parameter dictionary [#1167](#)
- Added first CV fold score as validation score in `AutoMLSearch.rankings` [#1221](#)
- Updated `flake8` configuration to enable linting on `__init__.py` files [#1234](#)
- Refined `make_pipeline_from_components` implementation [#1204](#)

• **Fixes**

- Updated GitHub URL after migration to Alteryx GitHub org [#1207](#)
- Changed Problem Type enum to be more similar to the string name [#1208](#)

- Wrapped call to scikit-learn’s partial dependence method in a `try/finally` block [#1232](#)
- **Changes**
  - Added `allow_writing_files` as a named argument to CatBoost estimators. [#1202](#)
  - Added `solver` and `multi_class` as named arguments to `LogisticRegressionClassifier` [#1202](#)
  - Replaced pipeline’s `._transform` method to evaluate all the preprocessing steps of a pipeline with `.compute_estimator_features` [#1231](#)
  - Changed default large dataset train/test splitting behavior [#1205](#)
- **Documentation Changes**
  - Included description of how to access the component instances and features for pipeline user guide [#1163](#)
  - Updated API docs to refer to target as “target” instead of “labels” for non-classification tasks and minor docs cleanup [#1160](#)
  - Added Class Imbalance Data Check to `api_reference.rst` [#1190](#) [#1200](#)
  - Added pipeline properties to API reference [#1209](#)
  - Clarified what the objective parameter in AutoML is used for in AutoML API reference and AutoML user guide [#1222](#)
  - Updated API docs to include `skopt.space.Categorical` option for component hyperparameter range definition [#1228](#)
  - Added install documentation for `libomp` in order to use LightGBM on Mac [#1233](#)
  - Improved description of `max_iterations` in documentation [#1212](#)
  - Removed unused code from sphinx conf [#1235](#)
- **Testing Changes**

**Warning:****Breaking Changes**

- `DefaultDataChecks` now accepts a `problem_type` parameter that must be specified [#1167](#)
- Pipeline’s `._transform` method to evaluate all the preprocessing steps of a pipeline has been replaced with `.compute_estimator_features` [#1231](#)
- `get_objectives` has been renamed to `get_core_objectives`. This function will now return a list of valid objective instances [#1230](#)

**v0.13.2 Sep. 17, 2020**

- **Enhancements**
  - Added `output_format` field to explain predictions functions [#1107](#)
  - Modified `get_objective` and `get_objectives` to be able to return any objective in `evalml.objectives` [#1132](#)
  - Added a `return_instance` boolean parameter to `get_objective` [#1132](#)
  - Added `ClassImbalanceDataCheck` to determine whether target imbalance falls below a given threshold [#1135](#)

- Added label encoder to LightGBM for binary classification #1152
- Added labels for the row index of confusion matrix #1154
- Added `AutoMLSearch` object as another parameter in search callbacks #1156
- Added the corresponding probability threshold for each point displayed in `graph_roc_curve` #1161
- Added `__eq__` for `ComponentBase` and `PipelineBase` #1178
- Added support for multiclass classification for `roc_curve` #1164
- Added `categories` accessor to `OneHotEncoder` for listing the categories associated with a feature #1182
- Added utility function to create pipeline instances from a list of component instances #1176

- **Fixes**

- Fixed XGBoost column names for partial dependence methods #1104
- Removed dead code validating column type from `TextFeaturizer` #1122
- Fixed issue where `Imputer` cannot fit when there is `None` in a categorical or boolean column #1144
- `OneHotEncoder` preserves the custom index in the input data #1146
- Fixed representation for `ModelFamily` #1165
- Removed duplicate `nbsphinx` dependency in `dev-requirements.txt` #1168
- Users can now pass in any valid kwargs to all estimators #1157
- Remove broken accessor `OneHotEncoder.get_feature_names` and unneeded base class #1179
- Removed LightGBM Estimator from AutoML models #1186

- **Changes**

- Pinned `scikit-optimize` version to 0.7.4 #1136
- Removed `tqdm` as a dependency #1177
- Added `lightgbm` version 3.0.0 to `latest_dependency_versions.txt` #1185
- Rename `max_pipelines` to `max_iterations` #1169

- **Documentation Changes**

- Fixed API docs for `AutoMLSearch.add_result_callback` #1113
- Added a step to our release process for pushing our latest version to conda-forge #1118
- Added warning for missing `ipywidgets` dependency for using `PipelineSearchPlots` on Jupyterlab #1145
- Updated `README.md` example to load demo dataset #1151
- Swapped mapping of breast cancer targets in `model_understanding.ipynb` #1170

- **Testing Changes**

- Added test confirming `TextFeaturizer` never outputs null values #1122
- Changed Python version of `Update Dependencies` action to 3.8.x #1137
- Fixed release notes check-in test for `Update Dependencies` actions #1172

**Warning:****Breaking Changes**

- `get_objective` will now return a class definition rather than an instance by default [#1132](#)
- Deleted `OPTIONS` dictionary in `evalml.objectives.utils.py` [#1132](#)
- If specifying an objective by string, the string must now match the objective's name field, case-insensitive [#1132](#)
- Passing “Cost Benefit Matrix”, “Fraud Cost”, “Lead Scoring”, “Mean Squared Log Error”, “Recall”, “Recall Macro”, “Recall Micro”, “Recall Weighted”, or “Root Mean Squared Log Error” to `AutoMLSearch` will now result in a `ValueError` rather than an `ObjectiveNotFoundError` [#1132](#)
- Search callbacks `start_iteration_callback` and `add_results_callback` have changed to include a copy of the `AutoMLSearch` object as a third parameter [#1156](#)
- Deleted `OneHotEncoder.get_feature_names` method which had been broken for a while, in favor of pipelines' `input_feature_names` [#1179](#)
- Deleted empty base class `CategoricalEncoder` which `OneHotEncoder` component was inheriting from [#1176](#)
- Results from `roc_curve` will now return as a list of dictionaries with each dictionary representing a class [#1164](#)
- `max_pipelines` now raises a `DeprecationWarning` and will be removed in the next release. `max_iterations` should be used instead. [#1169](#)

**v0.13.1 Aug. 25, 2020****• Enhancements**

- Added Cost-Benefit Matrix objective for binary classification [#1038](#)
- Split `fill_value` into `categorical_fill_value` and `numeric_fill_value` for `Imputer` [#1019](#)
- Added `explain_predictions` and `explain_predictions_best_worst` for explaining multiple predictions with SHAP [#1016](#)
- Added new LSA component for text featurization [#1022](#)
- Added guide on installing with conda [#1041](#)
- Added a “cost-benefit curve” util method to graph cost-benefit matrix scores vs. binary classification thresholds [#1081](#)
- Standardized error when calling `transform/predict` before `fit` for pipelines [#1048](#)
- Added `percent_better_than_baseline` to `AutoML` search rankings and full rankings table [#1050](#)
- Added one-way partial dependence and partial dependence plots [#1079](#)
- Added “Feature Value” column to prediction explanation reports. [#1064](#)
- Added LightGBM classification estimator [#1082](#), [#1114](#)
- Added `max_batches` parameter to `AutoMLSearch` [#1087](#)

**• Fixes**

- Updated `TextFeaturizer` component to no longer require an internet connection to run #1022
- Fixed non-deterministic element of `TextFeaturizer` transformations #1022
- Added a `StandardScaler` to all `ElasticNet` pipelines #1065
- Updated cost-benefit matrix to normalize score #1099
- Fixed logic in `calculate_percent_difference` so that it can handle negative values #1100

- **Changes**

- Added `needs_fitting` property to `ComponentBase` #1044
- Updated references to data types to use datatype lists defined in `evalml.utils.gen_utils` #1039
- Remove maximum version limit for SciPy dependency #1051
- Moved `all_components` and other component importers into runtime methods #1045
- Consolidated graphing utility methods under `evalml.utils.graph_utils` #1060
- Made slight tweaks to how `TextFeaturizer` uses `featuretools`, and did some refactoring of that and of LSA #1090
- Changed `show_all_features` parameter into `importance_threshold`, which allows for thresholding feature importance #1097, #1103

- **Documentation Changes**

- Update `setup.py` URL to point to the github repo #1037
- Added tutorial for using the cost-benefit matrix objective #1088
- Updated `model_understanding.ipynb` to include documentation for using plotly on Jupyter Lab #1108

- **Testing Changes**

- Refactor CircleCI tests to use matrix jobs (#1043)
- Added a test to check that all test directories are included in evalml package #1054

**Warning:****Breaking Changes**

- `confusion_matrix` and `normalize_confusion_matrix` have been moved to `evalml.utils` #1038
- All graph utility methods previously under `evalml.pipelines.graph_utils` have been moved to `evalml.utils.graph_utils` #1060

**v0.12.2 Aug. 6, 2020**

- **Enhancements**

- Add save/load method to components #1023
- Expose `pickle_protocol` as optional arg to save/load #1023
- Updated estimators used in AutoML to include `ExtraTrees` and `ElasticNet` estimators #1030

- **Fixes**

- **Changes**
  - Removed `DeprecationWarning` for `SimpleImputer` [#1018](#)
- **Documentation Changes**
  - Add note about version numbers to release process docs [#1034](#)
- **Testing Changes**
  - Test files are now included in the evalml package [#1029](#)

## v0.12.0 Aug. 3, 2020

- **Enhancements**
  - Added string and categorical targets support for binary and multiclass pipelines and check for numeric targets for `DetectLabelLeakage` data check [#932](#)
  - Added clear exception for regression pipelines if target datatype is string or categorical [#960](#)
  - Added target column names and class labels in `predict` and `predict_proba` output for pipelines [#951](#)
  - Added `_compute_shap_values` and `normalize_values` to `pipelines/explanations` module [#958](#)
  - Added `explain_prediction` feature which explains single predictions with SHAP [#974](#)
  - Added `Imputer` to allow different imputation strategies for numerical and categorical dtypes [#991](#)
  - Added support for configuring logfile path using env var, and don't create logger if there are filesystem errors [#975](#)
  - Updated catboost estimators' default parameters and automl hyperparameter ranges to speed up fit time [#998](#)
- **Fixes**
  - Fixed `ReadtheDocs` warning failure regarding embedded gif [#943](#)
  - Removed incorrect parameter passed to pipeline classes in `_add_baseline_pipelines` [#941](#)
  - Added universal error for calling `predict`, `predict_proba`, `transform`, and `feature_importances` before fitting [#969](#), [#994](#)
  - Made `TextFeaturizer` component and pip dependencies `featuretools` and `nlp_primitives` optional [#976](#)
  - Updated imputation strategy in automl to no longer limit impute strategy to `most_frequent` for all features if there are any categorical columns [#991](#)
  - Fixed `UnboundLocalError` for `cv_pipeline` when automl search errors [#996](#)
  - Fixed `Imputer` to reset dataframe index to preserve behavior expected from `SimpleImputer` [#1009](#)
- **Changes**
  - Moved `get_estimators` to `evalml.pipelines.components.utils` [#934](#)
  - Modified Pipelines to raise `PipelineScoreError` when they encounter an error during scoring [#936](#)
  - Moved `evalml.model_families.list_model_families` to `evalml.pipelines.components.allowed_model_families` [#959](#)

- Renamed `DateTimeFeaturization` to `DateTimeFeaturizer` [#977](#)
- Added check to stop search and raise an error if all pipelines in a batch return NaN scores [#1015](#)
- **Documentation Changes**
  - Updated `README.md` [#963](#)
  - Reworded message when errors are returned from data checks in search [#982](#)
  - Added section on understanding model predictions with `explain_prediction` to User Guide [#981](#)
  - Added a section to the user guide and api reference about how XGBoost and CatBoost are not fully supported. [#992](#)
  - Added custom components section in user guide [#993](#)
  - Updated FAQ section formatting [#997](#)
  - Updated release process documentation [#1003](#)
- **Testing Changes**
  - Moved `predict_proba` and `predict` tests regarding string / categorical targets to `test_pipelines.py` [#972](#)
  - Fixed dependency update bot by updating python version to 3.7 to avoid frequent github version updates [#1002](#)

**Warning:****Breaking Changes**

- `get_estimators` has been moved to `evalml.pipelines.components.utils` (previously was under `evalml.pipelines.utils`) [#934](#)
- Removed the `raise_errors` flag in AutoML search. All errors during pipeline evaluation will be caught and logged. [#936](#)
- `evalml.model_families.list_model_families` has been moved to `evalml.pipelines.components.allowed_model_families` [#959](#)
- `TextFeaturizer`: the `featuretools` and `nlp_primitives` packages must be installed after installing evalml in order to use this component [#976](#)
- Renamed `DateTimeFeaturization` to `DateTimeFeaturizer` [#977](#)

**v0.11.2 July 16, 2020**

- **Enhancements**
  - Added `NoVarianceDataCheck` to `DefaultDataChecks` [#893](#)
  - Added text processing and featurization component `TextFeaturizer` [#913](#), [#924](#)
  - Added additional checks to `InvalidTargetDataCheck` to handle invalid target data types [#929](#)
  - `AutoMLSearch` will now handle `KeyboardInterrupt` and prompt user for confirmation [#915](#)
- **Fixes**
  - Makes `automl` results a read-only property [#919](#)

- **Changes**

- Deleted static pipelines and refactored tests involving static pipelines, removed `all_pipelines()` and `get_pipelines()` #904
- Moved `list_model_families` to `evalml.model_family.utils` #903
- Updated `all_pipelines`, `all_estimators`, `all_components` to use the same mechanism for dynamically generating their elements #898
- Rename master branch to main #918
- Add pypi release github action #923
- Updated `AutoMLSearch.search` stdout output and logging and removed tqdm progress bar #921
- Moved automl config checks previously in `search()` to `init` #933

- **Documentation Changes**

- Reorganized and rewrote documentation #937
- Updated to use pydata sphinx theme #937
- Updated docs to use `release_notes` instead of `changelog` #942

- **Testing Changes**

- Cleaned up fixture names and usages in tests #895

**Warning:****Breaking Changes**

- `list_model_families` has been moved to `evalml.model_family.utils` (previously was under `evalml.pipelines.utils`) #903
- `get_estimators` has been moved to `evalml.pipelines.components.utils` (previously was under `evalml.pipelines.utils`) #934
- Static pipeline definitions have been removed, but similar pipelines can still be constructed via creating an instance of `PipelineBase` #904
- `all_pipelines()` and `get_pipelines()` utility methods have been removed #904

**v0.11.0 June 30, 2020**

- **Enhancements**

- Added multiclass support for ROC curve graphing #832
- Added preprocessing component to drop features whose percentage of NaN values exceeds a specified threshold #834
- Added data check to check for problematic target labels #814
- Added `PerColumnImputer` that allows imputation strategies per column #824
- Added transformer to drop specific columns #827
- Added support for categories, `handle_error`, and `drop` parameters in `OneHotEncoder` #830 #897
- Added preprocessing component to handle `DateTime` columns featurization #838



- Added ability to clone pipelines and components #842
- Define getter method for component `parameters` #847
- Added utility methods to calculate and graph permutation importances #860, #880
- Added new utility functions necessary for generating dynamic preprocessing pipelines #852
- Added kwargs to all components #863
- Updated `AutoSearchBase` to use dynamically generated preprocessing pipelines #870
- Added `SelectColumns` transformer #873
- Added ability to evaluate additional pipelines for automl search #874
- Added `default_parameters` class property to components and pipelines #879
- Added better support for disabling data checks in automl search #892
- Added ability to save and load AutoML objects to file #888
- Updated `AutoSearchBase.get_pipelines` to return an untrained pipeline instance #876
- Saved learned binary classification thresholds in automl results cv data dict #876
- **Fixes**
  - Fixed bug where `SimpleImputer` cannot handle dropped columns #846
  - Fixed bug where `PerColumnImputer` cannot handle dropped columns #855
  - Enforce requirement that builtin components save all inputted values in their parameters dict #847
  - Don't list base classes in `all_components` output #847
  - Standardize all components to output pandas data structures, and accept either pandas or numpy #853
  - Fixed rankings and `full_rankings` error when search has not been run #894
- **Changes**
  - Update `all_pipelines` and `all_components` to try initializing pipelines/components, and on failure exclude them #849
  - Refactor `handle_components` to `handle_components_class`, standardize to `ComponentBase` subclass instead of instance #850
  - Refactor “blacklist”/“whitelist” to “allow”/“exclude” lists #854
  - Replaced `AutoClassificationSearch` and `AutoRegressionSearch` with `AutoMLSearch` #871
  - Renamed `feature_importances` and `permutation_importances` methods to use singular names (`feature_importance` and `permutation_importance`) #883
  - Updated automl default data splitter to train/validation split for large datasets #877
  - Added open source license, update some repo metadata #887
  - Removed dead code in `_get_preprocessing_components` #896
- **Documentation Changes**
  - Fix some typos and update the EvalML logo #872
- **Testing Changes**
  - Update the changelog check job to expect the new branching pattern for the deps update bot #836

- Check that all components output pandas datastructures, and can accept either pandas or numpy #853
- Replaced `AutoClassificationSearch` and `AutoRegressionSearch` with `AutoMLSearch` #871

**Warning:**

**Breaking Changes**

- Pipelines' static `component_graph` field must contain either `ComponentBase` subclasses or `str`, instead of `ComponentBase` subclass instances #850
- Rename `handle_component` to `handle_component_class`. Now standardizes to `ComponentBase` subclasses instead of `ComponentBase` subclass instances #850
- Renamed `automl`'s `cv` argument to `data_split` #877
- Pipelines' and classifiers' `feature_importances` is renamed `feature_importance`, `graph_feature_importances` is renamed `graph_feature_importance` #883
- Passing `data_checks=None` to `automl` search will not perform any data checks as opposed to default checks. #892
- Pipelines to search for in AutoML are now determined automatically, rather than using the statically-defined pipeline classes. #870
- Updated `AutoSearchBase.get_pipelines` to return an untrained pipeline instance, instead of one which happened to be trained on the final cross-validation fold #876

**v0.10.0 May 29, 2020**

• **Enhancements**

- Added baseline models for classification and regression, add functionality to calculate baseline models before searching in AutoML #746
- Port over highly-null guardrail as a data check and define `DefaultDataChecks` and `DisableDataChecks` classes #745
- Update `Tuner` classes to work directly with pipeline parameters dicts instead of flat parameter lists #779
- Add Elastic Net as a pipeline option #812
- Added new Pipeline option `ExtraTrees` #790
- Added precision-recall curve metrics and plot for binary classification problems in `evalml.pipeline.graph_utils` #794
- Update the default `automl` algorithm to search in batches, starting with default parameters for each pipeline and iterating from there #793
- Added `AutoMLAlgorithm` class and `IterativeAlgorithm` impl, separated from `AutoSearchBase` #793

• **Fixes**

- Update pipeline score to return nan score for any objective which throws an exception during scoring #787
- Fixed bug introduced in #787 where binary classification metrics requiring predicted probabilities error in scoring #798

- CatBoost and XGBoost classifiers and regressors can no longer have a learning rate of 0 [#795](#)

- **Changes**

- Cleanup pipeline `score` code, and cleanup codecov [#711](#)
- Remove `pass` for abstract methods for codecov [#730](#)
- Added `__str__` for AutoSearch object [#675](#)
- Add util methods to graph ROC and confusion matrix [#720](#)
- Refactor AutoBase to AutoSearchBase [#758](#)
- Updated AutoBase with `data_checks` parameter, removed previous `detect_label_leakage` parameter, and added functionality to run data checks before search in AutoML [#765](#)
- Updated our logger to use Python’s logging utils [#763](#)
- Refactor most of AutoSearchBase.`_do_iteration impl` into AutoSearchBase.`_evaluate` [#762](#)
- Port over all guardrails to use the new DataCheck API [#789](#)
- Expanded `import_or_raise` to catch all exceptions [#759](#)
- Adds RMSE, MSLE, RMSLE as standard metrics [#788](#)
- Don’t allow Recall to be used as an objective for AutoML [#784](#)
- Removed feature selection from pipelines [#819](#)
- Update default estimator parameters to make automl search faster and more accurate [#793](#)

- **Documentation Changes**

- Add instructions to freeze master on `release.md` [#726](#)
- Update release instructions with more details [#727](#) [#733](#)
- Add objective base classes to API reference [#736](#)
- Fix components API to match other modules [#747](#)

- **Testing Changes**

- Delete codecov yml, use codecov.io’s default [#732](#)
- Added unit tests for fraud cost, lead scoring, and standard metric objectives [#741](#)
- Update codecov client [#782](#)
- Updated AutoBase `__str__` test to include no parameters case [#783](#)
- Added unit tests for ExtraTrees pipeline [#790](#)
- If codecov fails to upload, fail build [#810](#)
- Updated Python version of dependency action [#816](#)
- Update the dependency update bot to use a suffix when creating branches [#817](#)

**Warning:**
**Breaking Changes**

- The `detect_label_leakage` parameter for AutoML classes has been removed and replaced by a `data_checks` parameter [#765](#)
- Moved ROC and confusion matrix methods from `evalml.pipeline.plot_utils` to `evalml.pipeline.graph_utils` [#720](#)
- Tuner classes require a pipeline hyperparameter range dict as an init arg instead of a space definition [#779](#)
- `Tuner.propose` and `Tuner.add` work directly with pipeline parameters dicts instead of flat parameter lists [#779](#)
- `PipelineBase.hyperparameters` and `custom_hyperparameters` use pipeline parameters dict format instead of being represented as a flat list [#779](#)
- All guardrail functions previously under `evalml.guardrails.utils` will be removed and replaced by data checks [#789](#)
- Recall disallowed as an objective for AutoML [#784](#)
- `AutoSearchBase` parameter `tuner` has been renamed to `tuner_class` [#793](#)
- `AutoSearchBase` parameter `possible_pipelines` and `possible_model_families` have been renamed to `allowed_pipelines` and `allowed_model_families` [#793](#)

#### v0.9.0 Apr. 27, 2020

- **Enhancements**

- Added `Accuracy` as a standard objective [#624](#)
- Added `verbose` parameter to `load_fraud` [#560](#)
- Added `Balanced Accuracy` metric for binary, multiclass [#612](#) [#661](#)
- Added `XGBoost` regressor and `XGBoost` regression pipeline [#666](#)
- Added `Accuracy` metric for multiclass [#672](#)
- Added objective name in `AutoBase.describe_pipeline` [#686](#)
- Added `DataCheck` and `DataChecks`, `Message` classes and relevant subclasses [#739](#)

- **Fixes**

- Removed direct access to `cls.component_graph` [#595](#)
- Add testing files to `.gitignore` [#625](#)
- Remove circular dependencies from `Makefile` [#637](#)
- Add error case for `normalize_confusion_matrix()` [#640](#)
- Fixed `XGBoostClassifier` and `XGBoostRegressor` bug with feature names that contain `[, ],` or `<` [#659](#)
- Update `make_pipeline_graph` to not accidentally create empty file when testing if path is valid [#649](#)
- Fix pip installation warning about `docsutils` version, from `boto` dependency [#664](#)
- Removed zero division warning for F1/precision/recall metrics [#671](#)
- Fixed `summary` for pipelines without estimators [#707](#)

- **Changes**

- Updated default objective for binary/multiclass classification to log loss #613
- Created classification and regression pipeline subclasses and removed objective as an attribute of pipeline classes #405
- Changed the output of `score` to return one dictionary #429
- Created binary and multiclass objective subclasses #504
- Updated objectives API #445
- Removed call to `get_plot_data` from AutoML #615
- Set `raise_error` to default to True for AutoML classes #638
- Remove unnecessary “u” prefixes on some unicode strings #641
- Changed one-hot encoder to return uint8 dtypes instead of ints #653
- Pipeline `_name` field changed to `custom_name` #650
- Removed `graphs.py` and moved methods into `PipelineBase` #657, #665
- Remove `s3fs` as a dev dependency #664
- Changed requirements-parser to be a core dependency #673
- Replace `supported_problem_types` field on pipelines with `problem_type` attribute on base classes #678
- Changed AutoML to only show best results for a given pipeline template in rankings, added `full_rankings` property to show all #682
- Update `ModelFamily` values: don’t list `xgboost`/`catboost` as classifiers now that we have regression pipelines for them #677
- Changed AutoML’s `describe_pipeline` to get problem type from pipeline instead #685
- Standardize `import_or_raise` error messages #683
- Updated argument order of objectives to align with `sklearn`’s #698
- Renamed `pipeline.feature_importance_graph` to `pipeline.graph_feature_importances` #700
- Moved ROC and confusion matrix methods to `evalml.pipelines.plot_utils` #704
- Renamed `MultiClassificationObjective` to `MulticlassClassificationObjective`, to align with pipeline naming scheme #715

- **Documentation Changes**

- Fixed some sphinx warnings #593
- Fixed docstring for `AutoClassificationSearch` with correct command #599
- Limit `readthedocs` formats to pdf, not `htmlzip` and `epub` #594 #600
- Clean up objectives API documentation #605
- Fixed function on Exploring search results page #604
- Update release process doc #567
- `AutoClassificationSearch` and `AutoRegressionSearch` show inherited methods in API reference #651
- Fixed improperly formatted code in breaking changes for changelog #655

- Added configuration to treat Sphinx warnings as errors #660
- Removed separate plotting section for pipelines in API reference #657, #665
- Have leads example notebook load S3 files using https, so we can delete s3fs dev dependency #664
- Categorized components in API reference and added descriptions for each category #663
- Fixed Sphinx warnings about `BalancedAccuracy` objective #669
- Updated API reference to include missing components and clean up pipeline docstrings #689
- Reorganize API ref, and clarify pipeline sub-titles #688
- Add and update preprocessing utils in API reference #687
- Added inheritance diagrams to API reference #695
- Documented which default objective AutoML optimizes for #699
- Create separate install page #701
- Include more utils in API ref, like `import_or_raise` #704
- Add more color to pipeline documentation #705

- **Testing Changes**

- Matched install commands of `check_latest_dependencies` test and its GitHub action #578
- Added Github app to auto assign PR author as assignee #477
- Removed unneeded conda installation of xgboost in windows checkin tests #618
- Update graph tests to always use tmpfile dir #649
- Changelog checkin test workaround for release PRs: If ‘future release’ section is empty of PR refs, pass check #658
- Add changelog checkin test exception for `dep-update` branch #723

**Warning: Breaking Changes**

- Pipelines will now no longer take an `objective` parameter during instantiation, and will no longer have an `objective` attribute.
- `fit()` and `predict()` now use an optional `objective` parameter, which is only used in binary classification pipelines to fit for a specific objective.
- `score()` will now use a required `objectives` parameter that is used to determine all the objectives to score on. This differs from the previous behavior, where the pipeline’s objective was scored on regardless.
- `score()` will now return one dictionary of all objective scores.
- ROC and ConfusionMatrix plot methods via `Auto(*).plot` have been removed by #615 and are replaced by `roc_curve` and `confusion_matrix` in `evalml.pipelines.plot_utils` in #704
- `normalize_confusion_matrix` has been moved to `evalml.pipelines.plot_utils` #704
- Pipelines `_name` field changed to `custom_name`
- Pipelines `supported_problem_types` field is removed because it is no longer necessary #678
- Updated argument order of objectives’ `objective_function` to align with sklearn #698

- `pipeline.feature_importance_graph` has been renamed to `pipeline.graph_feature_importances` in #700
- Removed unsupported MSLE objective #704

v0.8.0 Apr. 1, 2020

- **Enhancements**

- Add normalization option and information to confusion matrix #484
- Add util function to drop rows with NaN values #487
- Renamed `PipelineBase.name` as `PipelineBase.summary` and redefined `PipelineBase.name` as class property #491
- Added access to parameters in Pipelines with `PipelineBase.parameters` (used to be return of `PipelineBase.describe`) #501
- Added `fill_value` parameter for `SimpleImputer` #509
- Added functionality to override component hyperparameters and made pipelines take hyperparameters from components #516
- Allow `numpy.random.RandomState` for `random_state` parameters #556

- **Fixes**

- Removed unused dependency `matplotlib`, and move `category_encoders` to test reqs #572

- **Changes**

- Undo version cap in XGBoost placed in #402 and allowed all released of XGBoost #407
- Support pandas 1.0.0 #486
- Made all references to the logger static #503
- Refactored `model_type` parameter for components and pipelines to `model_family` #507
- Refactored `problem_types` for pipelines and components into `supported_problem_types` #515
- Moved `pipelines/utils.save_pipeline` and `pipelines/utils.load_pipeline` to `PipelineBase.save` and `PipelineBase.load` #526
- Limit number of categories encoded by `OneHotEncoder` #517

- **Documentation Changes**

- Updated API reference to remove `PipelinePlot` and added moved `PipelineBase` plotting methods #483
- Add code style and github issue guides #463 #512
- Updated API reference for to surface class variables for pipelines and components #537
- Fixed README documentation link #535
- Unhid PR references in changelog #656

- **Testing Changes**

- Added automated dependency check PR #482, #505
- Updated automated dependency check comment #497

- Have build\_docs job use python executor, so that env vars are set properly [#547](#)
- Added simple test to make sure OneHotEncoder's top\_n works with large number of categories [#552](#)
- Run windows unit tests on PRs [#557](#)

**Warning: Breaking Changes**

- AutoClassificationSearch and AutoRegressionSearch's model\_types parameter has been refactored into allowed\_model\_families
- ModelTypes enum has been changed to ModelFamily
- Components and Pipelines now have a model\_family field instead of model\_type
- get\_pipelines utility function now accepts model\_families as an argument instead of model\_types
- PipelineBase.name no longer returns structure of pipeline and has been replaced by PipelineBase.summary
- PipelineBase.problem\_types and Estimator.problem\_types has been renamed to supported\_problem\_types
- pipelines/utils.save\_pipeline and pipelines/utils.load\_pipeline moved to PipelineBase.save and PipelineBase.load

**v0.7.0 Mar. 9, 2020****• Enhancements**

- Added emacs buffers to .gitignore [#350](#)
- Add CatBoost (gradient-boosted trees) classification and regression components and pipelines [#247](#)
- Added Tuner abstract base class [#351](#)
- Added n\_jobs as parameter for AutoClassificationSearch and AutoRegressionSearch [#403](#)
- Changed colors of confusion matrix to shades of blue and updated axis order to match scikit-learn's [#426](#)
- Added PipelineBase .graph and .feature\_importance\_graph methods, moved from previous location [#423](#)
- Added support for python 3.8 [#462](#)

**• Fixes**

- Fixed ROC and confusion matrix plots not being calculated if user passed own additional\_objectives [#276](#)
- Fixed ReadtheDocs FileNotFoundError exception for fraud dataset [#439](#)

**• Changes**

- Added n\_estimators as a tunable parameter for XGBoost [#307](#)
- Remove unused parameter ObjectiveBase.fit\_needs\_proba [#320](#)
- Remove extraneous parameter component\_type from all components [#361](#)



- Remove unused `rankings.csv` file #397
- Downloaded demo and test datasets so unit tests can run offline #408
- Remove `_needs_fitting` attribute from Components #398
- Changed `plot.feature_importance` to show only non-zero feature importances by default, added optional parameter to show all #413
- Refactored `PipelineBase` to take in parameter dictionary and moved pipeline metadata to class attribute #421
- Dropped support for Python 3.5 #438
- Removed unused `apply.py` file #449
- Clean up `requirements.txt` to remove unused deps #451
- Support installation without all required dependencies #459

- **Documentation Changes**

- Update `release.md` with instructions to release to internal license key #354

- **Testing Changes**

- Added tests for utils (and moved current utils to `gen_utils`) #297
- Moved XGBoost install into it's own separate step on Windows using Conda #313
- Rewind pandas version to before 1.0.0, to diagnose test failures for that version #325
- Added dependency update checkin test #324
- Rewind XGBoost version to before 1.0.0 to diagnose test failures for that version #402
- Update dependency check to use a whitelist #417
- Update unit test jobs to not install dev deps #455

**Warning: Breaking Changes**

- Python 3.5 will not be actively supported.

**v0.6.0 Dec. 16, 2019**

- **Enhancements**

- Added ability to create a plot of feature importances #133
- Add early stopping to AutoML using patience and tolerance parameters #241
- Added ROC and confusion matrix metrics and plot for classification problems and introduce `PipelineSearchPlots` class #242
- Enhanced AutoML results with search order #260
- Added utility function to show system and environment information #300

- **Fixes**

- Lower botocore requirement #235
- Fixed `decision_function` calculation for `FraudCost` objective #254
- Fixed return value of `Recall` metrics #264
- Components return `self` on fit #289

- **Changes**

- Renamed `automl` classes to `AutoRegressionSearch` and `AutoClassificationSearch` #287
- Updating demo datasets to retain column names #223
- Moving pipeline visualization to `PipelinePlot` class #228
- Standarizing inputs as `pd.DataFrame` / `pd.Series` #130
- Enforcing that pipelines must have an estimator as last component #277
- Added `ipywidgets` as a dependency in `requirements.txt` #278
- Added Random and Grid Search Tuners #240

- **Documentation Changes**

- Adding class properties to API reference #244
- Fix and filter `FutureWarnings` from `scikit-learn` #249, #257
- Adding Linear Regression to API reference and cleaning up some Sphinx warnings #227

- **Testing Changes**

- Added support for testing on Windows with CircleCI #226
- Added support for doctests #233

**Warning: Breaking Changes**

- The `fit()` method for `AutoClassifier` and `AutoRegressor` has been renamed to `search()`.
- `AutoClassifier` has been renamed to `AutoClassificationSearch`
- `AutoRegressor` has been renamed to `AutoRegressionSearch`
- `AutoClassificationSearch.results` and `AutoRegressionSearch.results` now is a dictionary with `pipeline_results` and `search_order` keys. `pipeline_results` can be used to access a dictionary that is identical to the old `.results` dictionary. Whereas, `search_order` returns a list of the search order in terms of `pipeline_id`.
- Pipelines now require an estimator as the last component in `component_list`. Slicing pipelines now throws an `NotImplementedError` to avoid returning pipelines without an estimator.

**v0.5.2 Nov. 18, 2019**

- **Enhancements**

- Adding basic pipeline structure visualization #211

- **Documentation Changes**

- Added notebooks to build process #212

**v0.5.1 Nov. 15, 2019**

- **Enhancements**

- Added basic outlier detection guardrail #151
- Added basic ID column guardrail #135
- Added support for unlimited pipelines with a `max_time` limit #70

- Updated `.readthedocs.yaml` to successfully build [#188](#)
- **Fixes**
  - Removed MSLE from default additional objectives [#203](#)
  - Fixed `random_state` passed in pipelines [#204](#)
  - Fixed slow down in `RFRegressor` [#206](#)
- **Changes**
  - Pulled information for `describe_pipeline` from pipeline's new `describe` method [#190](#)
  - Refactored pipelines [#108](#)
  - Removed guardrails from `Auto(*)` [#202](#), [#208](#)
- **Documentation Changes**
  - Updated documentation to show `max_time` enhancements [#189](#)
  - Updated release instructions for RTD [#193](#)
  - Added notebooks to build process [#212](#)
  - Added contributing instructions [#213](#)
  - Added new content [#222](#)

**v0.5.0 Oct. 29, 2019**

- **Enhancements**
  - Added basic one hot encoding [#73](#)
  - Use enums for `model_type` [#110](#)
  - Support for splitting regression datasets [#112](#)
  - Auto-infer multiclass classification [#99](#)
  - Added support for other units in `max_time` [#125](#)
  - Detect highly null columns [#121](#)
  - Added additional regression objectives [#100](#)
  - Show an interactive iteration vs. score plot when using `fit()` [#134](#)
- **Fixes**
  - Reordered `describe_pipeline` [#94](#)
  - Added type check for `model_type` [#109](#)
  - Fixed `s` units when setting string `max_time` [#132](#)
  - Fix objectives not appearing in API documentation [#150](#)
- **Changes**
  - Reorganized tests [#93](#)
  - Moved logging to its own module [#119](#)
  - Show progress bar history [#111](#)
  - Using `cloudpickle` instead of `pickle` to allow unloading of custom objectives [#113](#)
  - Removed `render.py` [#154](#)

- **Documentation Changes**

- Update release instructions [#140](#)
- Include additional\_objectives parameter [#124](#)
- Added Changelog [#136](#)

- **Testing Changes**

- Code coverage [#90](#)
- Added CircleCI tests for other Python versions [#104](#)
- Added doc notebooks as tests [#139](#)
- Test metadata for CircleCI and 2 core parallelism [#137](#)

#### v0.4.1 Sep. 16, 2019

- **Enhancements**

- Added AutoML for classification and regressor using Autobase and Skopt [#7](#) [#9](#)
- Implemented standard classification and regression metrics [#7](#)
- Added logistic regression, random forest, and XGBoost pipelines [#7](#)
- Implemented support for custom objectives [#15](#)
- Feature importance for pipelines [#18](#)
- Serialization for pipelines [#19](#)
- Allow fitting on objectives for optimal threshold [#27](#)
- Added detect label leakage [#31](#)
- Implemented callbacks [#42](#)
- Allow for multiclass classification [#21](#)
- Added support for additional objectives [#79](#)

- **Fixes**

- Fixed feature selection in pipelines [#13](#)
- Made random\_seed usage consistent [#45](#)

- **Documentation Changes**

- Documentation Changes
- Added docstrings [#6](#)
- Created notebooks for docs [#6](#)
- Initialized readthedocs EvalML [#6](#)
- Added favicon [#38](#)

- **Testing Changes**

- Added testing for loading data [#39](#)

#### v0.2.0 Aug. 13, 2019

- **Enhancements**

- Created fraud detection objective [#4](#)

**v0.1.0 July. 31, 2019**

- *First Release*
- **Enhancements**
  - Added lead scoring objective #1
  - Added basic classifier #1
- **Documentation Changes**
  - Initialized Sphinx for docs #1



## PYTHON MODULE INDEX

### e

`evalml`, 156  
`evalml.automl`, 156  
`evalml.automl.automl_algorithm`, 156  
`evalml.automl.automl_algorithm.automl_algorithm`, 156  
`evalml.automl.automl_algorithm.evalml_algorithm`, 157  
`evalml.automl.automl_algorithm.iterative_algorithm`, 159  
`evalml.automl.automl_search`, 180  
`evalml.automl.callbacks`, 187  
`evalml.automl.engine`, 166  
`evalml.automl.engine.cf_engine`, 166  
`evalml.automl.engine.dask_engine`, 168  
`evalml.automl.engine.engine_base`, 171  
`evalml.automl.engine.sequential_engine`, 174  
`evalml.automl.pipeline_search_plots`, 187  
`evalml.automl.utils`, 188  
`evalml.data_checks`, 198  
`evalml.data_checks.class_imbalance_data_check`, 198  
`evalml.data_checks.data_check`, 200  
`evalml.data_checks.data_check_action`, 201  
`evalml.data_checks.data_check_action_code`, 202  
`evalml.data_checks.data_check_message`, 202  
`evalml.data_checks.data_check_message_code`, 204  
`evalml.data_checks.data_check_message_type`, 206  
`evalml.data_checks.data_checks`, 206  
`evalml.data_checks.datetime_format_data_check`, 207  
`evalml.data_checks.datetime_nan_data_check`, 208  
`evalml.data_checks.default_data_checks`, 210  
`evalml.data_checks.highly_null_data_check`, 211  
`evalml.data_checks.id_columns_data_check`, 213  
`evalml.data_checks.invalid_targets_data_check`, 214  
`evalml.data_checks.multicollinearity_data_check`, 215  
`evalml.data_checks.natural_language_nan_data_check`, 216  
`evalml.data_checks.no_variance_data_check`, 217  
`evalml.data_checks.outliers_data_check`, 218  
`evalml.data_checks.sparsity_data_check`, 219  
`evalml.data_checks.target_distribution_data_check`, 221  
`evalml.data_checks.target_leakage_data_check`, 222  
`evalml.data_checks.uniqueness_data_check`, 223  
`evalml.data_checks.utils`, 225  
`evalml.demos`, 246  
`evalml.demos.breast_cancer`, 246  
`evalml.demos.churn`, 246  
`evalml.demos.diabetes`, 247  
`evalml.demos.fraud`, 247  
`evalml.demos.wine`, 248  
`evalml.exceptions`, 249  
`evalml.exceptions.exceptions`, 249  
`evalml.model_family`, 254  
`evalml.model_family.model_family`, 254  
`evalml.model_family.utils`, 255  
`evalml.model_understanding`, 256  
`evalml.model_understanding.force_plots`, 263  
`evalml.model_understanding.graphs`, 264  
`evalml.model_understanding.permutation_importance`, 274  
`evalml.model_understanding.prediction_explanations`, 256  
`evalml.model_understanding.prediction_explanations`





`evalml.pipelines.components.transformers.scalars`, 818  
`evalml.pipelines.components.transformers.sequential_pipeline_base`, 818  
`evalml.pipelines.components.transformers.sequential_pipeline`, 831  
`evalml.pipelines.components.transformers.sequential_pipeline_components.utils`, 910  
`evalml.pipelines.components.transformers.feature_selection.feature_selector`, 1079  
`evalml.pipelines.components.transformers.feature_selection.pipeline_base`, 1074  
`evalml.pipelines.components.transformers.feature_selection.pipeline_feature_selector`, 1079  
`evalml.pipelines.components.transformers.feature_selection.rf_regressor_feature_selector`, 1080  
`evalml.pipelines.components.transformers.imputer`, 1084  
`evalml.pipelines.components.transformers.imputer.imputer`, 1098  
`evalml.pipelines.components.transformers.imputer.preprocessing.imputer`, 1223  
`evalml.pipelines.components.transformers.imputer.preprocessing.data_splitters`, 1223  
`evalml.pipelines.components.transformers.imputer.preprocessing.simple_imputer`, 1223  
`evalml.pipelines.components.transformers.imputer.preprocessing.target_imputer`, 1223  
`evalml.pipelines.components.transformers.preprocessing`, 1225  
`evalml.pipelines.components.transformers.preprocessing.datetime_featurizer`, 1225  
`evalml.pipelines.components.transformers.preprocessing.delayed_feature_transformer`, 1233  
`evalml.pipelines.components.transformers.problem_types`, 1235  
`evalml.pipelines.components.transformers.problem_types.problem_types`, 1235  
`evalml.pipelines.components.transformers.random_search_tuner`, 1240  
`evalml.pipelines.components.transformers.random_search_tuner.tuner`, 1240  
`evalml.pipelines.components.transformers.random_search_tuner.tuner`, 1242  
`evalml.pipelines.components.transformers.random_search_tuner.tuner`, 1243  
`evalml.pipelines.components.transformers.random_search_tuner.tuner`, 1245  
`evalml.pipelines.components.transformers.random_search_tuner.tuner`, 1246  
`evalml.pipelines.components.transformers.random_search_tuner.tuner`, 1250  
`evalml.pipelines.components.transformers.random_search_tuner.tuner`, 1251  
`evalml.pipelines.components.transformers.random_search_tuner.tuner`, 1252  
`evalml.pipelines.components.transformers.random_search_tuner.tuner`, 1257  
`evalml.pipelines.components.transformers.samplers`, 793  
`evalml.pipelines.components.transformers.samplers.base_sampler`, 793  
`evalml.pipelines.components.transformers.samplers.oversamplers`, 798  
`evalml.pipelines.components.transformers.samplers.undersampler`, 805  
`evalml.pipelines.components.transformers.scalars`, 818



## INDEX

### A

- `abs_error()` (in module `evalml.pipelines.components.utils`), 911
- `all_components()` (in module `evalml.pipelines.components.utils`), 911
- `all_problem_types()` (in module `evalml.problem_types.problem_types`), 1236
- `AccuracyBinary` (class in `evalml.objectives`), 374
- `AccuracyBinary` (class in `evalml.objectives.standard_metrics`), 308
- `AccuracyMulticlass` (class in `evalml.objectives`), 376
- `AccuracyMulticlass` (class in `evalml.objectives.standard_metrics`), 310
- `add()` (`evalml.tuners.grid_search_tuner.GridSearchTuner` method), 1241
- `add()` (`evalml.tuners.GridSearchTuner` method), 1247
- `add()` (`evalml.tuners.random_search_tuner.RandomSearchTuner` method), 1243
- `add()` (`evalml.tuners.RandomSearchTuner` method), 1248
- `add()` (`evalml.tuners.skopt_tuner.SKOptTuner` method), 1244
- `add()` (`evalml.tuners.SKOptTuner` method), 1249
- `add()` (`evalml.tuners.Tuner` method), 1250
- `add()` (`evalml.tuners.tuner.Tuner` method), 1245
- `add_result()` (`evalml.automl.automl_algorithm.automl_algorithm.AutoMLAlgorithm` method), 157
- `add_result()` (`evalml.automl.automl_algorithm.AutoMLAlgorithm` method), 162
- `add_result()` (`evalml.automl.automl_algorithm.evalml_algorithm.EvalMLAlgorithm` method), 159
- `add_result()` (`evalml.automl.automl_algorithm.EvalMLAlgorithm` method), 164
- `add_result()` (`evalml.automl.automl_algorithm.iterative_algorithm.IterativeAlgorithm` method), 161
- `add_result()` (`evalml.automl.automl_algorithm.IterativeAlgorithm` method), 165
- `add_to_rankings()` (`evalml.automl.automl_search.AutoMLSearch` method), 184
- `add_to_rankings()` (`evalml.automl.AutoMLSearch` method), 194
- `add_to_rankings()` (`evalml.AutoMLSearch` method), 1265
- `all_problem_types()` (in module `evalml.problem_types.problem_types`), 1236
- `all_problem_types()` (in module `evalml.problem_types.problem_types`), 1240
- `allowed_model_families()` (in module `evalml.pipelines.components.utils`), 911
- `ARIMAREgressor` (class in `evalml.pipelines`), 1105
- `ARIMAREgressor` (class in `evalml.pipelines.components`), 916
- `ARIMAREgressor` (class in `evalml.pipelines.components.estimators`), 612
- `ARIMAREgressor` (class in `evalml.pipelines.components.estimators.regressors`), 574
- `ARIMAREgressor` (class in `evalml.pipelines.components.estimators.regressors.arima_regressors`), 536
- `AUC` (class in `evalml.objectives`), 378
- `AUC` (class in `evalml.objectives.standard_metrics`), 312
- `AUCMacro` (class in `evalml.objectives`), 380
- `AUCMacro` (class in `evalml.objectives.standard_metrics`), 314
- `AUCMicro` (class in `evalml.objectives`), 381
- `AUCMicro` (class in `evalml.objectives.standard_metrics`), 316
- `AUCWeighted` (class in `evalml.objectives`), 383
- `AUCWeighted` (class in `evalml.objectives.standard_metrics`), 317
- `AutoMLAlgorithm` (class in `evalml.automl.automl_algorithm`), 162
- `AutoMLAlgorithm` (class in `evalml.automl.automl_algorithm.automl_algorithm`), 156
- `AutoMLAlgorithmException`, 157, 163
- `AutoMLConfig` (in module `evalml.automl.utils`), 189
- `AutoMLSearch` (class in `evalml`), 1262
- `AutoMLSearch` (class in `evalml.automl`), 191

AutoMLSearch (class *evalml.automl.automl\_search*), 181  
AutoMLSearchException, 250, 252

## B

BalancedAccuracyBinary (class  
evalml.objectives), 385

```
BalancedAccuracyBinary (class
    evalml.objectives.standard_metrics), 319
```

BalancedAccuracyMulticlass (class  
evalml.objectives), 387

```
BalancedAccuracyMulticlass (class
    evalml.objectives.standard_metrics), 321
```

BalancedClassificationSampler (class *evalml.preprocessing.data splitters*), 1228

BalancedClassificationSampler (class in  
evalml.preprocessing.data\_splitters.balanced\_c  
1224

BaselineClassifier (class *evalml.pipelines.components*), 919

```
BaselineClassifier (class
    evalml.pipelines.components.estimators),
614
```

```
BaselineClassifier (class in  
evalml.pipelines.components.estimators.classifiers)
```

```
BaselineClassifier (class in sklearn.linear_model)
evalml.pipelines.components.estimators.classifiers.baseline_classifier
471 (evalml.pipelines.components.estimators.classifiers.baseline_classifier)
```

```
BaselineRegressor (class
evalml.pipelines.components), 921
```

```
BaselineRegressor (class
    evalml.pipelines.components.estimators),
```

```
BaselineRegressor (class in
evalml.pipelines.components.estimators.regressors),
bu
```

```

537         ProphetRegressor (class in evalml.pipelines.components.ProphetRegressor)
538         evalml.pipelines.components.estimators.regressors.baseline_regressor, 999
539         build_prophet_df()

```

BaseMeta (class in evalml.utils.base meta), 1251

```
BaseOversampler          (class)          in          method
evalml.pipelines.components.transformers.samplers.base_sampler),
793
```

ers.  
C

```
BaseSampler (class in calculate_percent_difference()
evalml.pipelines.components.transformers.samplers.base_sampler.py)
evalml.objectives.AccuracyBinary class
method) 375
```

`batch_number()` (*evalml.automl.automl\_algorithm.automl\_algorithm.evaluate\_percent\_difference* property), 157

```
batch_number() (evalml automl automl_algorithm AutoMLAlgorithm (method, 377
property), 162 calculate percent
```

```
batch_number() (evalml.automl.automl_algorithm.evalml_algorithm.BaseAlgorithm class method), 378
property), 159 calculate_percent_difference()
```

```
(evalml.objectives.AUCMacro class method),
```

---

[380](#)  
`calculate_percent_difference()`  
 (`evalml.objectives.AUCMicro` class method), [382](#)  
`calculate_percent_difference()`  
 (`evalml.objectives.AUCWeighted` class method), [384](#)  
`calculate_percent_difference()`  
 (`evalml.objectives.BalancedAccuracyBinary` class method), [385](#)  
`calculate_percent_difference()`  
 (`evalml.objectives.BalancedAccuracyMulticlass` class method), [387](#)  
`calculate_percent_difference()`  
 (`evalml.objectives.binary_classification_objective.BinaryClassificationObjective` class method), [288](#)  
`calculate_percent_difference()`  
 (`evalml.objectives.BinaryClassificationObjective` class method), [389](#)  
`calculate_percent_difference()`  
 (`evalml.objectives.cost_benefit_matrix.CostBenefitMatrix` class method), [291](#)  
`calculate_percent_difference()`  
 (`evalml.objectives.CostBenefitMatrix` class method), [392](#)  
`calculate_percent_difference()`  
 (`evalml.objectives.ExpVariance` class method), [394](#)  
`calculate_percent_difference()`  
 (`evalml.objectives.F1` class method), [396](#)  
`calculate_percent_difference()`  
 (`evalml.objectives.F1Macro` class method), [398](#)  
`calculate_percent_difference()`  
 (`evalml.objectives.F1Micro` class method), [399](#)  
`calculate_percent_difference()`  
 (`evalml.objectives.F1Weighted` class method), [401](#)  
`calculate_percent_difference()`  
 (`evalml.objectives.fraud_cost.FraudCost` class method), [293](#)  
`calculate_percent_difference()`  
 (`evalml.objectives.FraudCost` class method), [403](#)  
`calculate_percent_difference()`  
 (`evalml.objectives.Gini` class method), [406](#)  
`calculate_percent_difference()`  
 (`evalml.objectives.lead_scoring.LeadScoring` class method), [296](#)  
`calculate_percent_difference()`  
 (`evalml.objectives.LeadScoring` class method), [408](#)  
`calculate_percent_difference()`  
 (`evalml.objectives.LogLossBinary` class method), [410](#)  
`calculate_percent_difference()`  
 (`evalml.objectives.LogLossMulticlass` class method), [412](#)  
`calculate_percent_difference()`  
 (`evalml.objectives.MAE` class method), [414](#)  
`calculate_percent_difference()`  
 (`evalml.objectives.MAPE` class method), [416](#)  
`calculate_percent_difference()`  
 (`evalml.objectives.MaxError` class method), [417](#)  
`calculate_percent_difference()`  
 (`evalml.objectives.MCCBinary` class method), [419](#)  
`calculate_percent_difference()`  
 (`evalml.objectives.MCCMulticlass` class method), [421](#)  
`calculate_percent_difference()`  
 (`evalml.objectives.MeanSquaredLogError` class method), [423](#)  
`calculate_percent_difference()`  
 (`evalml.objectives.MedianAE` class method), [424](#)  
`calculate_percent_difference()`  
 (`evalml.objectives.MSE` class method), [426](#)  
`calculate_percent_difference()`  
 (`evalml.objectives.multiclass_classification_objective.MulticlassClassificationObjective` class method), [299](#)  
`calculate_percent_difference()`  
 (`evalml.objectives.MulticlassClassificationObjective` class method), [427](#)  
`calculate_percent_difference()`  
 (`evalml.objectives.objective_base.ObjectiveBase` class method), [301](#)  
`calculate_percent_difference()`  
 (`evalml.objectives.ObjectiveBase` class method), [429](#)  
`calculate_percent_difference()`  
 (`evalml.objectives.Precision` class method), [431](#)  
`calculate_percent_difference()`  
 (`evalml.objectives.PrecisionMacro` class method), [433](#)  
`calculate_percent_difference()`  
 (`evalml.objectives.PrecisionMicro` class method), [435](#)  
`calculate_percent_difference()`  
 (`evalml.objectives.PrecisionWeighted` class method), [437](#)  
`calculate_percent_difference()`  
 (`evalml.objectives.R2` class method), [438](#)  
`calculate_percent_difference()`

<code>(evalml.objectives.Recall class method), 440</code>	<code>(evalml.objectives.standard_metrics.ExpVariance class method), 323</code>
<code>calculate_percent_difference()</code> <code>(evalml.objectives.RecallMacro class method), 442</code>	<code>calculate_percent_difference()</code> <code>(evalml.objectives.standard_metrics.F1 class method), 325</code>
<code>calculate_percent_difference()</code> <code>(evalml.objectives.RecallMicro class method), 444</code>	<code>calculate_percent_difference()</code> <code>(evalml.objectives.standard_metrics.F1Macro class method), 327</code>
<code>calculate_percent_difference()</code> <code>(evalml.objectives.RecallWeighted class method), 445</code>	<code>calculate_percent_difference()</code> <code>(evalml.objectives.standard_metrics.F1Micro class method), 329</code>
<code>calculate_percent_difference()</code> <code>(evalml.objectives.regression_objective.RegressionObjective class method), 303</code>	<code>calculate_percent_difference()</code> <code>(evalml.objectives.standard_metrics.F1Weighted class method), 330</code>
<code>calculate_percent_difference()</code> <code>(evalml.objectives.RegressionObjective class method), 447</code>	<code>calculate_percent_difference()</code> <code>(evalml.objectives.standard_metrics.Gini class method), 332</code>
<code>calculate_percent_difference()</code> <code>(evalml.objectives.RootMeanSquaredError class method), 449</code>	<code>calculate_percent_difference()</code> <code>(evalml.objectives.standard_metrics.LogLossBinary class method), 334</code>
<code>calculate_percent_difference()</code> <code>(evalml.objectives.RootMeanSquaredLogError class method), 450</code>	<code>calculate_percent_difference()</code> <code>(evalml.objectives.standard_metrics.LogLossMulticlass class method), 336</code>
<code>calculate_percent_difference()</code> <code>(evalml.objectives.sensitivity_low_alert.SensitivityLowAlert class method), 306</code>	<code>calculate_percent_difference()</code> <code>(evalml.objectives.standard_metrics.MAE class method), 338</code>
<code>calculate_percent_difference()</code> <code>(evalml.objectives.SensitivityLowAlert class method), 452</code>	<code>calculate_percent_difference()</code> <code>(evalml.objectives.standard_metrics.MAPE class method), 339</code>
<code>calculate_percent_difference()</code> <code>(evalml.objectives.standard_metrics.AccuracyBinary class method), 309</code>	<code>calculate_percent_difference()</code> <code>(evalml.objectives.standard_metrics.MaxError class method), 341</code>
<code>calculate_percent_difference()</code> <code>(evalml.objectives.standard_metrics.AccuracyMulticlass class method), 311</code>	<code>calculate_percent_difference()</code> <code>(evalml.objectives.standard_metrics.MCCBinary class method), 342</code>
<code>calculate_percent_difference()</code> <code>(evalml.objectives.standard_metrics.AUC class method), 313</code>	<code>calculate_percent_difference()</code> <code>(evalml.objectives.standard_metrics.MCCMulticlass class method), 345</code>
<code>calculate_percent_difference()</code> <code>(evalml.objectives.standard_metrics.AUCMacro class method), 315</code>	<code>calculate_percent_difference()</code> <code>(evalml.objectives.standard_metrics.MeanSquaredLogError class method), 346</code>
<code>calculate_percent_difference()</code> <code>(evalml.objectives.standard_metrics.AUCMicro class method), 316</code>	<code>calculate_percent_difference()</code> <code>(evalml.objectives.standard_metrics.MedianAE class method), 348</code>
<code>calculate_percent_difference()</code> <code>(evalml.objectives.standard_metrics.AUCWeighted class method), 318</code>	<code>calculate_percent_difference()</code> <code>(evalml.objectives.standard_metrics.MSE class method), 349</code>
<code>calculate_percent_difference()</code> <code>(evalml.objectives.standard_metrics.BalancedAccuracyBinary class method), 320</code>	<code>calculate_percent_difference()</code> <code>(evalml.objectives.standard_metrics.Precision class method), 351</code>
<code>calculate_percent_difference()</code> <code>(evalml.objectives.standard_metrics.BalancedAccuracyMulticlass class method), 322</code>	<code>calculate_percent_difference()</code> <code>(evalml.objectives.standard_metrics.PrecisionMacro class method), 353</code>
<code>calculate_percent_difference()</code>	<code>calculate_percent_difference()</code>



(`evalml.objectives.standard_metrics.PrecisionMicro` class method), 355

`calculate_percent_difference()` (`evalml.objectives.standard_metrics.PrecisionWeighted` class method), 356

`calculate_percent_difference()` (`evalml.objectives.standard_metrics.R2` class method), 358

`calculate_percent_difference()` (`evalml.objectives.standard_metrics.Recall` class method), 359

`calculate_percent_difference()` (`evalml.objectives.standard_metrics.RecallMacro` class method), 361

`calculate_percent_difference()` (`evalml.objectives.standard_metrics.RecallMicro` class method), 363

`calculate_percent_difference()` (`evalml.objectives.standard_metrics.RecallWeighted` class method), 365

`calculate_percent_difference()` (`evalml.objectives.standard_metrics.RootMeanSquaredError` class method), 366

`calculate_percent_difference()` (`evalml.objectives.standard_metrics.RootMeanSquaredLogError` class method), 368

`calculate_percent_difference()` (`evalml.objectives.time_series_regression_objective.TimeSeriesRegressionObjective` class method), 370

`calculate_permutation_importance()` (in module `evalml.model_understanding`), 277

`calculate_permutation_importance()` (in module `evalml.model_understanding.permutation_importance`), 274

`calculate_permutation_importance_one_column()` (in module `evalml.model_understanding`), 277

`calculate_permutation_importance_one_column()` (in module `evalml.model_understanding.permutation_importance`), 274

`can_optimize_threshold()` (`evalml.objectives.AccuracyBinary` property), 375

`can_optimize_threshold()` (`evalml.objectives.AUC` property), 379

`can_optimize_threshold()` (`evalml.objectives.BalancedAccuracyBinary` property), 386

`can_optimize_threshold()` (`evalml.objectives.binary_classification_objective.BinaryClassificationObjective` property), 288

`can_optimize_threshold()` (`evalml.objectives.BinaryClassificationObjective` property), 389

`can_optimize_threshold()` (`evalml.objectives.cost_benefit_matrix.CostBenefitMatrix` property), 291

`can_optimize_threshold()` (`evalml.objectives.CostBenefitMatrix` property), 392

`can_optimize_threshold()` (`evalml.objectives.F1` property), 396

`can_optimize_threshold()` (`evalml.objectives.fraud_cost.FraudCost` property), 294

`can_optimize_threshold()` (`evalml.objectives.FraudCost` property), 403

`can_optimize_threshold()` (`evalml.objectives.Gini` property), 406

`can_optimize_threshold()` (`evalml.objectives.lead_scoring.LeadScoring` property), 296

`can_optimize_threshold()` (`evalml.objectives.LeadScoring` property), 408

`can_optimize_threshold()` (`evalml.objectives.LogLossBinary` property), 411

`can_optimize_threshold()` (`evalml.objectives.MCCBinary` property), 419

`can_optimize_threshold()` (`evalml.objectives.Precision` property), 431

`can_optimize_threshold()` (`evalml.objectives.Recall` property), 440

`can_optimize_threshold()` (`evalml.objectives.sensitivity_low_alert.SensitivityLowAlert` property), 306

`can_optimize_threshold()` (`evalml.objectives.SensitivityLowAlert` property), 452

`can_optimize_threshold()` (`evalml.objectives.standard_metrics.AccuracyBinary` property), 309

`can_optimize_threshold()` (`evalml.objectives.standard_metrics.AUC` property), 313

`can_optimize_threshold()` (`evalml.objectives.standard_metrics.BalancedAccuracyBinary` property), 320

`can_optimize_threshold()` (`evalml.objectives.standard_metrics.F1` property), 325

`can_optimize_threshold()` (`evalml.objectives.standard_metrics.Gini` property), 332

`can_optimize_threshold()` (`evalml.objectives.standard_metrics.LogLossBinary`

[property](#)), 334  
[can\\_optimize\\_threshold\(\)](#) ([evalml.objectives.standard\\_metrics.MCCBinary](#) [property](#)), 343  
[can\\_optimize\\_threshold\(\)](#) ([evalml.objectives.standard\\_metrics.Precision](#) [property](#)), 351  
[can\\_optimize\\_threshold\(\)](#) ([evalml.objectives.standard\\_metrics.Recall](#) [property](#)), 360  
[can\\_tune\\_threshold\\_with\\_objective\(\)](#) ([evalml.pipelines.binary\\_classification\\_pipeline.BinaryClassificationPipeline](#) [method](#)), 1058  
[can\\_tune\\_threshold\\_with\\_objective\(\)](#) ([evalml.pipelines.classification\\_pipeline.ClassificationPipeline](#) [method](#)), 1063  
[can\\_tune\\_threshold\\_with\\_objective\(\)](#) ([evalml.pipelines.multiclass\\_classification\\_pipeline.MulticlassClassificationPipeline](#) [method](#)), 1071  
[can\\_tune\\_threshold\\_with\\_objective\(\)](#) ([evalml.pipelines.MulticlassClassificationPipeline](#) [method](#)), 1155  
[can\\_tune\\_threshold\\_with\\_objective\(\)](#) ([evalml.pipelines.pipeline\\_base.PipelineBase](#) [method](#)), 1076  
[can\\_tune\\_threshold\\_with\\_objective\(\)](#) ([evalml.pipelines.PipelineBase](#) [method](#)), 1164  
[can\\_tune\\_threshold\\_with\\_objective\(\)](#) ([evalml.pipelines.regression\\_pipeline.RegressionPipeline](#) [method](#)), 1081  
[can\\_tune\\_threshold\\_with\\_objective\(\)](#) ([evalml.pipelines.RegressionPipeline](#) [method](#)), 1175  
[can\\_tune\\_threshold\\_with\\_objective\(\)](#) ([evalml.pipelines.time\\_series\\_classification\\_pipelines.TimeSeriesBinaryClassificationPipeline](#) [method](#)), 1086  
[can\\_tune\\_threshold\\_with\\_objective\(\)](#) ([evalml.pipelines.time\\_series\\_classification\\_pipelines.TimeSeriesMulticlassClassificationPipeline](#) [method](#)), 1091  
[can\\_tune\\_threshold\\_with\\_objective\(\)](#) ([evalml.pipelines.time\\_series\\_classification\\_pipelines.TimeSeriesRegressionPipeline](#) [method](#)), 1095  
[can\\_tune\\_threshold\\_with\\_objective\(\)](#) ([evalml.pipelines.time\\_series\\_regression\\_pipeline.TimeSeriesRegressionPipeline](#) [method](#)), 1099  
[can\\_tune\\_threshold\\_with\\_objective\(\)](#) ([evalml.pipelines.TimeSeriesBinaryClassificationPipeline](#) [method](#)), 1200  
[can\\_tune\\_threshold\\_with\\_objective\(\)](#) ([evalml.pipelines.TimeSeriesClassificationPipeline](#) [method](#)), 1205  
[can\\_tune\\_threshold\\_with\\_objective\(\)](#) ([evalml.pipelines.TimeSeriesMulticlassClassificationPipeline](#) [method](#)), 1209  
[can\\_tune\\_threshold\\_with\\_objective\(\)](#) ([evalml.pipelines.TimeSeriesRegressionPipeline](#) [method](#)), 1213  
[cancel\(\)](#) ([evalml.automl.engine.cf\\_engine.CFComputation](#) [method](#)), 167  
[cancel\(\)](#) ([evalml.automl.engine.dask\\_engine.DaskComputation](#) [method](#)), 169  
[cancel\(\)](#) ([evalml.automl.engine.engine\\_base.EngineComputation](#) [method](#)), 172  
[cancel\(\)](#) ([evalml.automl.engine.EngineComputation](#) [method](#)), 178  
[cancel\(\)](#) ([evalml.automl.engine.sequential\\_engine.SequentialComputation](#) [method](#)), 174  
[CatBoostClassifier](#) (class in [evalml.pipelines](#)), 1107  
[CatBoostClassifier](#) (class in [evalml.pipelines.components](#)), 924  
[CatBoostClassifier](#) (class in [evalml.pipelines.components.estimators](#)), 619  
[CatBoostClassifier](#) (class in [evalml.pipelines.components.estimators.classifiers](#)), 507  
[CatBoostClassifier](#) (class in [evalml.pipelines.components.estimators.classifiers.catboost\\_classifiers](#)), 474  
[CatBoostRegressor](#) (class in [evalml.pipelines](#)), 1110  
[CatBoostRegressor](#) (class in [evalml.pipelines.components](#)), 926  
[CatBoostRegressor](#) (class in [evalml.pipelines.components.estimators](#)), 622  
[CatBoostRegressor](#) (class in [evalml.pipelines.components.estimators.regressors](#)), 579  
[CatBoostRegressor](#) (class in [evalml.pipelines.components.estimators.regressors.catboost\\_regressors](#)), 542  
[categories\(\)](#) ([evalml.pipelines.components.OneHotEncoder](#) [method](#)), 689  
[categories\(\)](#) ([evalml.pipelines.components.transformers.encoders.one\\_hot\\_encoder.OneHotEncoder](#) [method](#)), 689  
[categories\(\)](#) ([evalml.pipelines.components.transformers.encoders.OneHotEncoder](#) [method](#)), 696  
[categories\(\)](#) ([evalml.pipelines.components.transformers.OneHotEncoder](#) [method](#)), 863  
[categories\(\)](#) ([evalml.pipelines.OneHotEncoder](#) [method](#)), 1159  
[CFClient](#) (class in [evalml.automl.engine.cf\\_engine](#)), 166  
[CFComputation](#) (class in [evalml.automl.engine](#)), 175



CFEngine (class in evalml.engine.cf_engine),	clone()	(evalml.pipelines.ARIMARegressor method),
167		1106
check_all_pipeline_names_unique()	(in clone())	(evalml.pipelines.binary_classification_pipeline.BinaryClassificationPipeline method),
module evalml.automl.utils), 189		1058
check_for_fit()	(evalml.pipelines.components.component_base.BaseComponent method),	909
class method), 909		1108
check_for_fit()	(evalml.pipelines.components.ComponentBase method),	931
class method), 931		1111
check_for_fit()	(evalml.pipelines.components.transformers.set_of_delayed_pipeline_classes.OfferingPipeline method),	691
class method), 691		1063
check_for_fit()	(evalml.pipelines.components.transformers.set_of_delayed_pipeline_classes.OfferingPipeline method),	729
class method), 729		918
check_for_fit()	(evalml.pipelines.pipeline_meta.PipelineBase method),	1079
class method), 1079		920
check_for_fit()	(evalml.pipelines.pipeline_meta.TimeSeriesPipelineBase method),	1080
class method), 1080		922
classes_()	(evalml.pipelines.binary_classification_pipeline.BinaryClassificationPipeline property),	1058
property), 1058		925
classes_()	(evalml.pipelines.classification_pipeline.ClassificationPipeline property),	1063
property), 1063		927
classes_()	(evalml.pipelines.components.BaselineClassifier property),	920
property), 920		908
classes_()	(evalml.pipelines.components.estimators.BaselineClassifier property),	615
property), 615		929
classes_()	(evalml.pipelines.components.estimators.classifiers.baseline_classifier.BaselineClassifier property),	472
property), 472		932
classes_()	(evalml.pipelines.components.estimators.classifiers.baseline_classifier.BaselineClassifier property),	505
property), 505		935
classes_()	(evalml.pipelines.multiclass_classification_pipeline.MulticlassClassificationPipeline property),	1071
property), 1071		938
classes_()	(evalml.pipelines.MulticlassClassificationPipeline property),	1155
property), 1155		940
classes_()	(evalml.pipelines.time_series_classification_pipeline.TimeSeriesClassificationPipeline property),	1086
property), 1086		943
classes_()	(evalml.pipelines.time_series_classification_pipeline.TimeSeriesClassificationPipeline property),	1091
property), 1091		945
classes_()	(evalml.pipelines.time_series_classification_pipeline.TimeSeriesClassificationPipeline property),	1095
property), 1095		947
classes_()	(evalml.pipelines.TimeSeriesBinaryClassificationPipeline property),	1201
property), 1201		950
classes_()	(evalml.pipelines.TimeSeriesClassificationPipeline property),	1205
property), 1205		952
classes_()	(evalml.pipelines.TimeSeriesMulticlassClassificationPipeline property),	1209
property), 1209		954
ClassificationPipeline (class in clone())	(evalml.pipelines.components.ensemble.sklearn_stacked_ensemble method),	1062
evalml.pipelines.classification_pipeline),		455
1062		458
ClassImbalanceDataCheck (class in clone())	(evalml.pipelines.components.ensemble.sklearn_stacked_ensemble method),	227
evalml.data_checks), 227		461
ClassImbalanceDataCheck (class in clone())	(evalml.pipelines.components.ensemble.sklearn_stacked_ensemble method),	199
evalml.data_checks.class_imbalance_data_check),		464
199		467
classproperty (class in evalml.utils), 1259		467
classproperty (class in evalml.utils.gen_utils), 1253		467

clone() (evalml.pipelines.components.ensemble.SklearnStackedEnsembleClassifier method), 469

clone() (evalml.pipelines.components.ensemble.XGBoostPipeline method), 503

clone() (evalml.pipelines.components.Estimator method), 956

clone() (evalml.pipelines.components.estimators.ARIMARegressor method), 613

clone() (evalml.pipelines.components.estimators.BaselineClassifier method), 615

clone() (evalml.pipelines.components.estimators.BaselineRegressor method), 618

clone() (evalml.pipelines.components.estimators.CatBoostClassifier method), 620

clone() (evalml.pipelines.components.estimators.CatBoostRegressor method), 623

clone() (evalml.pipelines.components.estimators.classifiers.baseline\_classifier method), 472

clone() (evalml.pipelines.components.estimators.classifiers.baseline\_classifier method), 472

clone() (evalml.pipelines.components.estimators.classifiers.BaselineClassifier method), 506

clone() (evalml.pipelines.components.estimators.classifiers.catboost\_classifier method), 475

clone() (evalml.pipelines.components.estimators.classifiers.CatBoostClassifier method), 508

clone() (evalml.pipelines.components.estimators.classifiers.DecisionTreeClassifier method), 478

clone() (evalml.pipelines.components.estimators.classifiers.DecisionTreeClassifier method), 511

clone() (evalml.pipelines.components.estimators.classifiers.elasticnet\_classifier method), 481

clone() (evalml.pipelines.components.estimators.classifiers.ElasticNetClassifier method), 514

clone() (evalml.pipelines.components.estimators.classifiers.elasticnet\_classifier method), 485

clone() (evalml.pipelines.components.estimators.classifiers.ExtraTreesClassifier method), 517

clone() (evalml.pipelines.components.estimators.classifiers.kneighbors\_classifier method), 488

clone() (evalml.pipelines.components.estimators.classifiers.KNeighborsClassifier method), 520

clone() (evalml.pipelines.components.estimators.classifiers.lightgbm\_classifier method), 491

clone() (evalml.pipelines.components.estimators.classifiers.LightGBMClassifier method), 523

clone() (evalml.pipelines.components.estimators.classifiers.logistic\_regression\_classifier method), 494

clone() (evalml.pipelines.components.estimators.classifiers.LogisticRegressionClassifier method), 526

clone() (evalml.pipelines.components.estimators.classifiers.RandomForestClassifier method), 528

clone() (evalml.pipelines.components.estimators.classifiers.rf\_classifier method), 497

clone() (evalml.pipelines.components.estimators.classifiers.svm\_classifier method), 500

clone() (evalml.pipelines.components.estimators.classifiers.SVMClassifier method), 531

clone() (evalml.pipelines.components.estimators.classifiers.XGBoostClassifier method), 534

clone() (evalml.pipelines.components.estimators.DecisionTreeRegressor method), 626

clone() (evalml.pipelines.components.estimators.DecisionTreeRegressor method), 629

clone() (evalml.pipelines.components.estimators.ElasticNetClassifier method), 632

clone() (evalml.pipelines.components.estimators.ElasticNetRegressor method), 635

clone() (evalml.pipelines.components.estimators.Estimator method), 637

clone() (evalml.pipelines.components.estimators.estimator.Estimator method), 609

clone() (evalml.pipelines.components.estimators.ExtraTreesClassifier method), 640

clone() (evalml.pipelines.components.estimators.ExtraTreesRegressor method), 643

clone() (evalml.pipelines.components.estimators.KNeighborsClassifier method), 645

clone() (evalml.pipelines.components.estimators.LightGBMClassifier method), 648

clone() (evalml.pipelines.components.estimators.LightGBMRegressor method), 651

clone() (evalml.pipelines.components.estimators.LinearRegressor method), 654

clone() (evalml.pipelines.components.estimators.LogisticRegressionClassifier method), 656

clone() (evalml.pipelines.components.estimators.ProphetRegressor method), 659

clone() (evalml.pipelines.components.estimators.RandomForestClassifier method), 661

clone() (evalml.pipelines.components.estimators.RandomForestRegressor method), 663

clone() (evalml.pipelines.components.estimators.regressors.arima\_regressor method), 537

clone() (evalml.pipelines.components.estimators.regressors.ARIMARegressor method), 575

clone() (evalml.pipelines.components.estimators.regressors.baseline\_regressor method), 540

clone() (evalml.pipelines.components.estimators.regressors.baseline\_regressor method), 578

clone() (evalml.pipelines.components.estimators.regressors.catboost\_regressor method), 543

clone() (evalml.pipelines.components.estimators.regressors.CatBoostRegressor method), 580

clone() (evalml.pipelines.components.estimators.regressors.decision\_tree\_regressor method), 546

clone() (evalml.pipelines.components.estimators.regressors.DecisionTreeRegressor method), 583

clone() (evalml.pipelines.components.estimators.regressors.elasticnet\_regressor method), 549

clone () (evalml.pipelines.components.estimators.regressors.ExtraTreesRegressor method), 586	clone () (evalml.pipelines.components.LightGBMClassifier method), 972
clone () (evalml.pipelines.components.estimators.regressors.LinearRegression method), 552	clone () (evalml.pipelines.components.LightGBMRegressor method), 975
clone () (evalml.pipelines.components.estimators.regressors.ExtendedTreesRegressor method), 589	clone () (evalml.pipelines.components.LogisticRegression method), 977
clone () (evalml.pipelines.components.estimators.regressors.LightGBMRegressor method), 555	clone () (evalml.pipelines.components.LinearDiscriminantAnalysis method), 980
clone () (evalml.pipelines.components.estimators.regressors.LightGBMRegressor method), 592	clone () (evalml.pipelines.components.LogisticRegressionClassifier method), 982
clone () (evalml.pipelines.components.estimators.regressors.LinearRegression method), 558	clone () (evalml.pipelines.components.LogTransformer method), 984
clone () (evalml.pipelines.components.estimators.regressors.LinearRegression method), 594	clone () (evalml.pipelines.components.LSA method), 987
clone () (evalml.pipelines.components.estimators.regressors.ProphetRegressor method), 561	clone () (evalml.pipelines.components.OneHotEncoder method), 989
clone () (evalml.pipelines.components.estimators.regressors.ProphetRegressor method), 597	clone () (evalml.pipelines.components.PCA method), 992
clone () (evalml.pipelines.components.estimators.regressors.RandomForestRegressor method), 599	clone () (evalml.pipelines.components.PerColumnImputer method), 994
clone () (evalml.pipelines.components.estimators.regressors.SerialRegression method), 564	clone () (evalml.pipelines.components.PolynomialDetrender method), 996
clone () (evalml.pipelines.components.estimators.regressors.SVR method), 567	clone () (evalml.pipelines.components.ProphetRegressor method), 999
clone () (evalml.pipelines.components.estimators.regressors.SVMRegressor method), 602	clone () (evalml.pipelines.components.RandomForestClassifier method), 1001
clone () (evalml.pipelines.components.estimators.regressors.TimeSeriesBaselineEstimator method), 569	clone () (evalml.pipelines.components.SerialRegression method), 1003
clone () (evalml.pipelines.components.estimators.regressors.TimeSeriesBaselineEstimator method), 604	clone () (evalml.pipelines.components.RFClassifierSelectFromModel method), 1006
clone () (evalml.pipelines.components.estimators.regressors.XGBoostRegressor method), 572	clone () (evalml.pipelines.components.RFRegressorSelectFromModel method), 1009
clone () (evalml.pipelines.components.estimators.regressors.XGBoostRegressor method), 606	clone () (evalml.pipelines.components.SelectByType method), 1011
clone () (evalml.pipelines.components.estimators.SVMClassifier method), 666	clone () (evalml.pipelines.components.SelectColumns method), 1013
clone () (evalml.pipelines.components.estimators.SVMRegressor method), 668	clone () (evalml.pipelines.components.SimpleImputer method), 1015
clone () (evalml.pipelines.components.estimators.TimeSeriesBaselineEstimator method), 670	clone () (evalml.pipelines.components.StackedEnsembleClassifier method), 1018
clone () (evalml.pipelines.components.estimators.XGBoostClassifier method), 673	clone () (evalml.pipelines.components.StackedEnsembleRegressor method), 1020
clone () (evalml.pipelines.components.estimators.XGBoostRegressor method), 676	clone () (evalml.pipelines.components.SMOTENCOversampler method), 1023
clone () (evalml.pipelines.components.ExtraTreesClassifier method), 959	clone () (evalml.pipelines.components.SMOTENCOversampler method), 1025
clone () (evalml.pipelines.components.ExtraTreesRegressor method), 962	clone () (evalml.pipelines.components.SMOTEOverSampler method), 1027
clone () (evalml.pipelines.components.FeatureSelector method), 964	clone () (evalml.pipelines.components.StandardScaler method), 1029
clone () (evalml.pipelines.components.Imputer method), 967	clone () (evalml.pipelines.components.SVMClassifier method), 1032
clone () (evalml.pipelines.components.KNeighborsClassifier method), 970	clone () (evalml.pipelines.components.SVMRegressor method), 1034

`clone()` (`evalml.pipelines.components.TargetEncoder` method), 1036  
`clone()` (`evalml.pipelines.components.TargetImputer` method), 1038  
`clone()` (`evalml.pipelines.components.TextFeaturizer` method), 1041  
`clone()` (`evalml.pipelines.components.TimeSeriesBaselineEstimator` method), 1043  
`clone()` (`evalml.pipelines.components.Transformer` method), 1045  
`clone()` (`evalml.pipelines.components.transformers.column_selector.CalibratedSelector` method), 824  
`clone()` (`evalml.pipelines.components.transformers.column_selector.DropColumns` method), 826  
`clone()` (`evalml.pipelines.components.transformers.column_selector.SelectByType` method), 828  
`clone()` (`evalml.pipelines.components.transformers.column_selector.SelectColumns` method), 830  
`clone()` (`evalml.pipelines.components.transformers.DateTimeFeaturizer` method), 838  
`clone()` (`evalml.pipelines.components.transformers.DelayedFeatureTransformer` method), 841  
`clone()` (`evalml.pipelines.components.transformers.DFSTransformer` method), 843  
`clone()` (`evalml.pipelines.components.transformers.dimensionality_reduction.LinearDiscriminantAnalysis` method), 678  
`clone()` (`evalml.pipelines.components.transformers.dimensionality_reduction.LogTransformer` method), 683  
`clone()` (`evalml.pipelines.components.transformers.dimensionality_reduction.LSA` method), 685  
`clone()` (`evalml.pipelines.components.transformers.dimensionality_reduction.OneHotEncoder` method), 681  
`clone()` (`evalml.pipelines.components.transformers.DropColumns` method), 845  
`clone()` (`evalml.pipelines.components.transformers.DropSubColumns` method), 847  
`clone()` (`evalml.pipelines.components.transformers.EmailFeaturizer` method), 849  
`clone()` (`evalml.pipelines.components.transformers.encoder.OneHotEncoder` method), 689  
`clone()` (`evalml.pipelines.components.transformers.encoder.OneHotEncoder` method), 696  
`clone()` (`evalml.pipelines.components.transformers.encoder.target_encoder.TargetEncoder` method), 693  
`clone()` (`evalml.pipelines.components.transformers.encoder.TargetEncoder` method), 699  
`clone()` (`evalml.pipelines.components.transformers.feature_selection.FeatureSelector` method), 701  
`clone()` (`evalml.pipelines.components.transformers.feature_selection.FeatureSelector` method), 711  
`clone()` (`evalml.pipelines.components.transformers.feature_selection.FeatureSelector` method), 704  
`clone()` (`evalml.pipelines.components.transformers.feature_selection.FeatureSelector` method), 708  
`clone()` (`evalml.pipelines.components.transformers.feature_selection.RFClassifierSelector` method), 713  
`clone()` (`evalml.pipelines.components.transformers.feature_selection.RFClassifierSelector` method), 716  
`clone()` (`evalml.pipelines.components.transformers.FeatureSelector` method), 851  
`clone()` (`evalml.pipelines.components.transformers.Imputer` method), 854  
`clone()` (`evalml.pipelines.components.transformers.imputers.Imputer` method), 731  
`clone()` (`evalml.pipelines.components.transformers.imputers.imputer.Imputer` method), 719  
`clone()` (`evalml.pipelines.components.transformers.imputers.per_column.PerColumnImputer` method), 722  
`clone()` (`evalml.pipelines.components.transformers.imputers.PerColumnImputer` method), 733  
`clone()` (`evalml.pipelines.components.transformers.imputers.simple_imp.SimpleImputer` method), 725  
`clone()` (`evalml.pipelines.components.transformers.imputers.SimpleImputer` method), 736  
`clone()` (`evalml.pipelines.components.transformers.imputers.target_imp.TargetImputer` method), 727  
`clone()` (`evalml.pipelines.components.transformers.imputers.TargetImputer` method), 738  
`clone()` (`evalml.pipelines.components.transformers.linear_discriminant.LinearDiscriminantAnalysis` method), 856  
`clone()` (`evalml.pipelines.components.transformers.LogTransformer` method), 858  
`clone()` (`evalml.pipelines.components.transformers.LSA` method), 860  
`clone()` (`evalml.pipelines.components.transformers.OneHotEncoder` method), 863  
`clone()` (`evalml.pipelines.components.transformers.PCA` method), 865  
`clone()` (`evalml.pipelines.components.transformers.PCA` method), 865  
`clone()` (`evalml.pipelines.components.transformers.PerColumnImputer` method), 868  
`clone()` (`evalml.pipelines.components.transformers.PolynomialDetrender` method), 870  
`clone()` (`evalml.pipelines.components.transformers.preprocessing.datetimetarget_encoder.TargetEncoder` method), 741  
`clone()` (`evalml.pipelines.components.transformers.preprocessing.DateTransformer` method), 769  
`clone()` (`evalml.pipelines.components.transformers.preprocessing.delayed_target_encoder.TargetEncoder` method), 744  
`clone()` (`evalml.pipelines.components.transformers.preprocessing.DelayedFeatureTransformer` method), 772  
`clone()` (`evalml.pipelines.components.transformers.preprocessing.DFSTransformer` method), 774  
`clone()` (`evalml.pipelines.components.transformers.preprocessing.drop_columns.DropColumns` method), 746  
`clone()` (`evalml.pipelines.components.transformers.preprocessing.DropSubColumns` method), 776  
`clone()` (`evalml.pipelines.components.transformers.preprocessing.FeatureSelector` method), 778





`method)`, 1134  
`clone()` (`evalml.pipelines.ExtraTreesRegressor` `method`), 1137  
`clone()` (`evalml.pipelines.FeatureSelector` `method`), 1139  
`clone()` (`evalml.pipelines.KNeighborsClassifier` `method`), 1142  
`clone()` (`evalml.pipelines.LightGBMClassifier` `method`), 1145  
`clone()` (`evalml.pipelines.LightGBMRegressor` `method`), 1147  
`clone()` (`evalml.pipelines.LinearRegressor` `method`), 1150  
`clone()` (`evalml.pipelines.LogisticRegressionClassifier` `method`), 1152  
`clone()` (`evalml.pipelines.multiclass_classification_pipeline.MulticlassClassificationPipeline` `method`), 1071  
`clone()` (`evalml.pipelines.MulticlassClassificationPipeline` `method`), 1155  
`clone()` (`evalml.pipelines.OneHotEncoder` `method`), 1159  
`clone()` (`evalml.pipelines.PerColumnImputer` `method`), 1162  
`clone()` (`evalml.pipelines.pipeline_base.PipelineBase` `method`), 1076  
`clone()` (`evalml.pipelines.PipelineBase` `method`), 1164  
`clone()` (`evalml.pipelines.ProphetRegressor` `method`), 1168  
`clone()` (`evalml.pipelines.RandomForestClassifier` `method`), 1170  
`clone()` (`evalml.pipelines.RandomForestRegressor` `method`), 1172  
`clone()` (`evalml.pipelines.regression_pipeline.RegressionPipeline` `method`), 1082  
`clone()` (`evalml.pipelines.RegressionPipeline` `method`), 1175  
`clone()` (`evalml.pipelines.RFClassifierSelectFromModel` `method`), 1179  
`clone()` (`evalml.pipelines.RFRegressorSelectFromModel` `method`), 1181  
`clone()` (`evalml.pipelines.SimpleImputer` `method`), 1184  
`clone()` (`evalml.pipelines.SklearnStackedEnsembleClassifier` `method`), 1186  
`clone()` (`evalml.pipelines.SklearnStackedEnsembleRegressor` `method`), 1189  
`clone()` (`evalml.pipelines.StandardScaler` `method`), 1191  
`clone()` (`evalml.pipelines.SVMClassifier` `method`), 1193  
`clone()` (`evalml.pipelines.SVMRegressor` `method`), 1195  
`clone()` (`evalml.pipelines.TargetEncoder` `method`), 1198  
`clone()` (`evalml.pipelines.time_series_classification_pipelines.TimeSeriesClassificationPipeline` `method`), 1086  
`clone()` (`evalml.pipelines.time_series_classification_pipelines.TimeSeriesClassificationPipeline` `method`), 1091  
`clone()` (`evalml.pipelines.time_series_classification_pipelines.TimeSeriesClassificationPipeline` `method`), 1095  
`clone()` (`evalml.pipelines.time_series_regression_pipeline.TimeSeriesRegressionPipeline` `method`), 1100  
`clone()` (`evalml.pipelines.TimeSeriesBinaryClassificationPipeline` `method`), 1201  
`clone()` (`evalml.pipelines.TimeSeriesClassificationPipeline` `method`), 1205  
`clone()` (`evalml.pipelines.TimeSeriesMulticlassClassificationPipeline` `method`), 1209  
`clone()` (`evalml.pipelines.TimeSeriesRegressionPipeline` `method`), 1217  
`clone()` (`evalml.pipelines.Transformer` `method`), 1217  
`clone()` (`evalml.pipelines.XGBoostClassifier` `method`), 1219  
`clone()` (`evalml.pipelines.XGBoostRegressor` `method`), 1222  
`ColumnSelector` (class in `evalml.pipelines.components.transformers.column_selectors`), 823  
`ComponentBase` (class in `evalml.pipelines.components`), 929  
`ComponentBase` (class in `evalml.pipelines.components.component_base`), 907  
`ComponentBaseMeta` (class in `evalml.pipelines.components`), 930  
`ComponentBaseMeta` (class in `evalml.pipelines.components.component_base_meta`), 909  
`ComponentGraph` (class in `evalml.pipelines`), 1112  
`ComponentGraph` (class in `evalml.pipelines.component_graph`), 1067  
`ComponentNotYetFittedError`, 250, 252  
`compute_estimator_features()` (`evalml.pipelines.binary_classification_pipeline.BinaryClassificationPipeline` `method`), 1058  
`compute_estimator_features()` (`evalml.pipelines.classification_pipeline.ClassificationPipeline` `method`), 1063  
`compute_estimator_features()` (`evalml.pipelines.multiclass_classification_pipeline.MulticlassClassificationPipeline` `method`), 1071  
`compute_estimator_features()` (`evalml.pipelines.MulticlassClassificationPipeline` `method`), 1155  
`compute_estimator_features()` (`evalml.pipelines.pipeline_base.PipelineBase` `method`), 1076  
`compute_estimator_features()`

`(evalml.pipelines.PipelineBase method), create_objectives()`  
 1165 `(evalml.pipelines.binary_classification_pipeline.BinaryClassificationPipeline static method), 1058`  
`compute_estimator_features()`  
`(evalml.pipelines.regression_pipeline.RegressionPipeline create_objectives()`  
`method), 1082` `(evalml.pipelines.classification_pipeline.ClassificationPipeline static method), 1064`  
`compute_estimator_features()`  
`(evalml.pipelines.RegressionPipeline method), create_objectives()`  
 1175 `(evalml.pipelines.multiclass_classification_pipeline.MulticlassClassificationPipeline static method), 1072`  
`compute_estimator_features()`  
`(evalml.pipelines.time_series_classification_pipeline.TimeSeriesBinaryClassificationPipeline method), 1086` `(evalml.pipelines.MulticlassClassificationPipeline static method), 1155`  
`compute_estimator_features()`  
`(evalml.pipelines.time_series_classification_pipeline.TimeSeriesClassificationPipeline method), 1091` `(evalml.pipelines.pipeline_base.PipelineBase static method), 1076`  
`compute_estimator_features()`  
`(evalml.pipelines.time_series_classification_pipeline.TimeSeriesMulticlassClassificationPipeline method), 1095` `(evalml.pipelines.PipelineBase static method), 1165`  
`compute_estimator_features()`  
`(evalml.pipelines.time_series_regression_pipeline.TimeSeriesRegressionPipeline method), 1100` `(evalml.pipelines.regression_pipeline.RegressionPipeline static method), 1082`  
`compute_estimator_features()`  
`(evalml.pipelines.TimeSeriesBinaryClassificationPipeline create_objectives()`  
`method), 1201` `(evalml.pipelines.RegressionPipeline static method), 1175`  
`compute_estimator_features()`  
`(evalml.pipelines.TimeSeriesClassificationPipeline create_objectives()`  
`method), 1205` `(evalml.pipelines.time_series_classification_pipeline.TimeSeriesClassificationPipeline static method), 1086`  
`compute_estimator_features()`  
`(evalml.pipelines.TimeSeriesMulticlassClassificationPipeline create_objectives()`  
`method), 1209` `(evalml.pipelines.time_series_classification_pipeline.TimeSeriesMulticlassClassificationPipeline static method), 1091`  
`compute_estimator_features()`  
`(evalml.pipelines.TimeSeriesRegressionPipeline create_objectives()`  
`method), 1214` `(evalml.pipelines.time_series_classification_pipeline.TimeSeriesRegressionPipeline static method), 1095`  
`compute_final_component_features()`  
`(evalml.pipelines.component_graph.ComponentGraph create_objectives()`  
`method), 1068` `(evalml.pipelines.time_series_regression_pipeline.TimeSeriesRegressionPipeline static method), 1100`  
`compute_final_component_features()`  
`(evalml.pipelines.ComponentGraph method), create_objectives()`  
 1113 `(evalml.pipelines.TimeSeriesBinaryClassificationPipeline static method), 1201`  
`compute_order()` `(evalml.pipelines.component_graph.ComponentGraph static method), 1068`  
`property), 1068` `create_objectives()`  
`compute_order()` `(evalml.pipelines.ComponentGraph method), create_objectives()`  
`property), 1114` `(evalml.pipelines.TimeSeriesClassificationPipeline static method), 1205`  
`confusion_matrix()` `(in module evalml.model_understanding), 278` `(evalml.pipelines.TimeSeriesMulticlassClassificationPipeline static method), 1210`  
`confusion_matrix()` `(in module evalml.model_understanding.graphs), 265` `create_objectives()`  
`convert_to_seconds()` `(in module evalml.utils), 1259` `(evalml.pipelines.TimeSeriesRegressionPipeline static method), 1214`  
`convert_to_seconds()` `(in module evalml.utils.gen_utils), 1254` `cross_entropy()` `(in module evalml.model_understanding.prediction_explanations.explainers), 257`  
`CostBenefitMatrix` `(class in evalml.objectives), 391` `custom_name()` `(evalml.pipelines.binary_classification_pipeline.BinaryClassificationPipeline property), 1058`  
`CostBenefitMatrix` `(class in evalml.objectives.cost_benefit_matrix), 290` `custom_name()` `(evalml.pipelines.classification_pipeline.ClassificationPipeline property), 1064`

[property](#)), 1064  
[custom\\_name\(\)](#) ([evalml.pipelines.multiclass\\_classification\\_pipeline](#)), 231  
[property](#)), 1072  
[custom\\_name\(\)](#) ([evalml.pipelines.MulticlassClassificationPipeline](#)), 203  
[property](#)), 1155  
[custom\\_name\(\)](#) ([evalml.pipelines.pipeline\\_base.PipelineBase](#)), 230  
[property](#)), 1076  
[custom\\_name\(\)](#) ([evalml.pipelines.PipelineBase](#)), 230  
[property](#)), 1165  
[custom\\_name\(\)](#) ([evalml.pipelines.regression\\_pipeline.RegressionPipeline](#)), 231  
[property](#)), 1082  
[custom\\_name\(\)](#) ([evalml.pipelines.RegressionPipeline](#)), 231  
[property](#)), 1175  
[custom\\_name\(\)](#) ([evalml.pipelines.time\\_series\\_classification\\_pipeline.TimeSeriesBinaryClassificationPipeline](#)), 206  
[property](#)), 1086  
[custom\\_name\(\)](#) ([evalml.pipelines.time\\_series\\_classification\\_pipeline.TimeSeriesMulticlassClassificationPipeline](#)), 232  
[property](#)), 1091  
[custom\\_name\(\)](#) ([evalml.pipelines.time\\_series\\_classification\\_pipeline.TimeSeriesMulticlassClassificationPipeline](#)), 232  
[property](#)), 1095  
[custom\\_name\(\)](#) ([evalml.pipelines.time\\_series\\_regression\\_pipeline.TimeSeriesRegressionPipeline](#)), 232  
[property](#)), 1100  
[custom\\_name\(\)](#) ([evalml.pipelines.TimeSeriesBinaryClassificationPipeline](#)), 203  
[property](#)), 1201  
[custom\\_name\(\)](#) ([evalml.pipelines.TimeSeriesClassificationPipeline](#)), 232  
[property](#)), 1205  
[custom\\_name\(\)](#) ([evalml.pipelines.TimeSeriesMulticlassClassificationPipeline](#)), 232  
[property](#)), 1210  
[custom\\_name\(\)](#) ([evalml.pipelines.TimeSeriesRegressionPipeline](#)), 232  
[property](#)), 1214

**D**  
[DaskComputation](#) (class in [evalml.automl.engine.dask\\_engine](#)), 169  
[DaskEngine](#) (class in [evalml.automl.engine](#)), 176  
[DaskEngine](#) (class in [evalml.automl.engine.dask\\_engine](#)), 169  
[DataCheck](#) (class in [evalml.data\\_checks](#)), 228  
[DataCheck](#) (class in [evalml.data\\_checks.data\\_check](#)), 200  
[DataCheckAction](#) (class in [evalml.data\\_checks](#)), 229  
[DataCheckAction](#) (class in [evalml.data\\_checks.data\\_check\\_action](#)), 201  
[DataCheckActionCode](#) (class in [evalml.data\\_checks](#)), 229  
[DataCheckActionCode](#) (class in [evalml.data\\_checks.data\\_check\\_action\\_code](#)), 202  
[DataCheckError](#) (class in [evalml.data\\_checks](#)), 229  
[DataCheckError](#) (class in [evalml.data\\_checks.data\\_check\\_message](#)), 203  
[DataCheckInitError](#), 250, 252

[DataCheckMessage](#) (class in [evalml.data\\_checks](#)), 230  
[MulticlassClassificationPipeline](#)  
[DataCheckMessage](#) (class in [evalml.data\\_checks.data\\_check\\_message](#)), 203  
[DataCheckMessageCode](#) (class in [evalml.data\\_checks](#)), 230  
[DataCheckMessageCode](#) (class in [evalml.data\\_checks.data\\_check\\_message\\_code](#)), 204  
[DataCheckMessageType](#) (class in [evalml.data\\_checks](#)), 231  
[DataCheckMessageType](#) (class in [evalml.data\\_checks.data\\_check\\_message\\_type](#)), 206  
[DataChecks](#) (class in [evalml.pipelines.components](#)), 931  
[DataCheckWarning](#) (class in [evalml.data\\_checks](#)), 231  
[DataCheckWarning](#) (class in [evalml.data\\_checks.data\\_check\\_message](#)), 203  
[DataFeaturizer](#) (class in [evalml.pipelines.components](#)), 931  
[DataFeaturizer](#) (class in [evalml.pipelines.components.transformers](#)), 837  
[DateTimeFeaturizer](#) (class in [evalml.pipelines.components.transformers.preprocessing](#)), 768  
[DateTimeFeaturizer](#) (class in [evalml.pipelines.components.transformers.preprocessing.datetime](#)), 740  
[DateTimeFormatDataCheck](#) (class in [evalml.data\\_checks](#)), 232  
[DateTimeFormatDataCheck](#) (class in [evalml.data\\_checks.datetime\\_format\\_data\\_check](#)), 207  
[DateTimeNaNDataCheck](#) (class in [evalml.data\\_checks](#)), 233  
[DateTimeNaNDataCheck](#) (class in [evalml.data\\_checks.datetime\\_nan\\_data\\_check](#)), 209  
[debug\(\)](#) ([evalml.automl.engine.engine\\_base.JobLogger](#) method), 172  
[decision\\_function\(\)](#) ([evalml.objectives.AccuracyBinary](#) method), 375  
[decision\\_function\(\)](#) ([evalml.objectives.AUC](#) method), 379  
[decision\\_function\(\)](#) ([evalml.objectives.BalancedAccuracyBinary](#) method), 386



`decision_function()` (`evalml.objectives.binary_classification_objective.BinaryClassificationObjective` method), 288  
`decision_function()` (`evalml.objectives.BinaryClassificationObjective` method), 389  
`decision_function()` (`evalml.objectives.cost_benefit_matrix.CostBenefitMatrix` method), 291  
`decision_function()` (`evalml.objectives.CostBenefitMatrix` method), 392  
`decision_function()` (`evalml.objectives.F1` method), 396  
`decision_function()` (`evalml.objectives.fraud_cost.FraudCost` method), 294  
`decision_function()` (`evalml.objectives.FraudCost` method), 403  
`decision_function()` (`evalml.objectives.Gini` method), 406  
`decision_function()` (`evalml.objectives.lead_scoring.LeadScoring` method), 296  
`decision_function()` (`evalml.objectives.LeadScoring` method), 408  
`decision_function()` (`evalml.objectives.LogLossBinary` method), 411  
`decision_function()` (`evalml.objectives.MCCBinary` method), 419  
`decision_function()` (`evalml.objectives.Precision` method), 432  
`decision_function()` (`evalml.objectives.Recall` method), 440  
`decision_function()` (`evalml.objectives.sensitivity_low_alert.SensitivityLowAlert` method), 306  
`decision_function()` (`evalml.objectives.SensitivityLowAlert` method), 452  
`decision_function()` (`evalml.objectives.standard_metrics.AccuracyBinary` method), 309  
`decision_function()` (`evalml.objectives.standard_metrics.AUC` method), 313  
`decision_function()` (`evalml.objectives.standard_metrics.BalancedAccuracyBinary` method), 320  
`decision_function()` (`evalml.objectives.standard_metrics.F1` method), 325  
`decision_function()` (`evalml.objectives.standard_metrics.Gini` method), 332  
`decision_function()` (`evalml.objectives.standard_metrics.LogLossBinary` method), 334  
`decision_function()` (`evalml.objectives.standard_metrics.MCCBinary` method), 343  
`decision_function()` (`evalml.objectives.standard_metrics.Precision` method), 351  
`decision_function()` (`evalml.objectives.standard_metrics.Recall` method), 360  
`decision_tree_data_from_estimator()` (in module `evalml.model_understanding.graphs`), 266  
`decision_tree_data_from_pipeline()` (in module `evalml.model_understanding.graphs`), 266  
`DecisionTreeClassifier` (class in `evalml.pipelines`), 1115  
`DecisionTreeClassifier` (class in `evalml.pipelines.components`), 934  
`DecisionTreeClassifier` (class in `evalml.pipelines.components.estimators`), 624  
`DecisionTreeClassifier` (class in `evalml.pipelines.components.estimators.classifiers`), 510  
`DecisionTreeClassifier` (class in `evalml.pipelines.components.estimators.classifiers.decision_tree`), 477  
`DecisionTreeRegressor` (class in `evalml.pipelines`), 1118  
`DecisionTreeRegressor` (class in `evalml.pipelines.components`), 936  
`DecisionTreeRegressor` (class in `evalml.pipelines.components.estimators`), 627  
`DecisionTreeRegressor` (class in `evalml.pipelines.components.estimators.regressors`), 582  
`DecisionTreeRegressor` (class in `evalml.pipelines.components.estimators.regressors.decision_tree`), 545  
`DEFAULT_METRICS` (in module `evalml.model_understanding.prediction_explanations.explainers`), 257  
`default_parameters()` (`evalml.pipelines.ARIMARegressor` method), 1106

<code>default_parameters()</code> ( <i>evalml.pipelines.CatBoostClassifier</i> method), 1109	<code>default_parameters()</code> ( <i>evalml.pipelines.components.ElasticNetClassifier</i> method), 950
<code>default_parameters()</code> ( <i>evalml.pipelines.CatBoostRegressor</i> method), 1111	<code>default_parameters()</code> ( <i>evalml.pipelines.components.ElasticNetRegressor</i> method), 952
<code>default_parameters()</code> ( <i>evalml.pipelines.component_graph.ComponentGraph</i> property), 1068	<code>default_parameters()</code> ( <i>evalml.pipelines.components.EmailFeaturizer</i> method), 954
<code>default_parameters()</code> ( <i>evalml.pipelines.ComponentGraph</i> prop- erty), 1114	<code>default_parameters()</code> ( <i>evalml.pipelines.components.ensemble.sklearn_stacked_ensemble</i> method), 455
<code>default_parameters()</code> ( <i>evalml.pipelines.components.ARIMARegressor</i> method), 918	<code>default_parameters()</code> ( <i>evalml.pipelines.components.ensemble.sklearn_stacked_ensemble</i> method), 458
<code>default_parameters()</code> ( <i>evalml.pipelines.components.BaselineClassifier</i> method), 920	<code>default_parameters()</code> ( <i>evalml.pipelines.components.ensemble.sklearn_stacked_ensemble</i> method), 461
<code>default_parameters()</code> ( <i>evalml.pipelines.components.BaselineRegressor</i> method), 922	<code>default_parameters()</code> ( <i>evalml.pipelines.components.ensemble.SklearnStackedEnsemble</i> method), 464
<code>default_parameters()</code> ( <i>evalml.pipelines.components.CatBoostClassifier</i> method), 925	<code>default_parameters()</code> ( <i>evalml.pipelines.components.ensemble.SklearnStackedEnsemble</i> method), 467
<code>default_parameters()</code> ( <i>evalml.pipelines.components.CatBoostRegressor</i> method), 927	<code>default_parameters()</code> ( <i>evalml.pipelines.components.ensemble.SklearnStackedEnsemble</i> method), 469
<code>default_parameters()</code> ( <i>evalml.pipelines.components.component_base.ComponentBase</i> method), 908	<code>default_parameters()</code> ( <i>evalml.pipelines.components.Estimator</i> method), 956
<code>default_parameters()</code> ( <i>evalml.pipelines.components.ComponentBase</i> method), 929	<code>default_parameters()</code> ( <i>evalml.pipelines.components.estimators.ARIMARegressor</i> method), 613
<code>default_parameters()</code> ( <i>evalml.pipelines.components.DateTimeFeaturizer</i> method), 932	<code>default_parameters()</code> ( <i>evalml.pipelines.components.estimators.BaselineClassifier</i> method), 615
<code>default_parameters()</code> ( <i>evalml.pipelines.components.DecisionTreeClassifier</i> method), 935	<code>default_parameters()</code> ( <i>evalml.pipelines.components.estimators.BaselineRegressor</i> method), 618
<code>default_parameters()</code> ( <i>evalml.pipelines.components.DecisionTreeRegressor</i> method), 938	<code>default_parameters()</code> ( <i>evalml.pipelines.components.estimators.CatBoostClassifier</i> method), 620
<code>default_parameters()</code> ( <i>evalml.pipelines.components.DelayedFeatureTransformer</i> method), 941	<code>default_parameters()</code> ( <i>evalml.pipelines.components.estimators.CatBoostRegressor</i> method), 623
<code>default_parameters()</code> ( <i>evalml.pipelines.components.DFSTransformer</i> method), 943	<code>default_parameters()</code> ( <i>evalml.pipelines.components.estimators.classifiers.baseline_class</i> method), 472
<code>default_parameters()</code> ( <i>evalml.pipelines.components.DropColumns</i> method), 945	<code>default_parameters()</code> ( <i>evalml.pipelines.components.estimators.classifiers.BaselineClass</i> method), 506
<code>default_parameters()</code> ( <i>evalml.pipelines.components.DropNullColumns</i> method), 947	<code>default_parameters()</code> ( <i>evalml.pipelines.components.estimators.classifiers.catboost_class</i> method), 475

<code>default_parameters()</code> ( <code>evalml.pipelines.components.estimators.classifiers.CatBoostClassifier</code> method), 508	<code>default_parameters()</code> ( <code>evalml.pipelines.components.estimators.classifiers.XGBoostClassifier</code> method), 534
<code>default_parameters()</code> ( <code>evalml.pipelines.components.estimators.classifiers.decision_tree.DecisionTreeClassifier</code> method), 478	<code>default_parameters()</code> ( <code>evalml.pipelines.components.estimators.classifiers.decision_tree.DecisionTreeClassifier</code> method), 626
<code>default_parameters()</code> ( <code>evalml.pipelines.components.estimators.classifiers.DecisionTreeRegressor</code> method), 511	<code>default_parameters()</code> ( <code>evalml.pipelines.components.estimators.DecisionTreeRegressor</code> method), 629
<code>default_parameters()</code> ( <code>evalml.pipelines.components.estimators.classifiers.elasticnet.ElasticNetClassifier</code> method), 481	<code>default_parameters()</code> ( <code>evalml.pipelines.components.estimators.ElasticNetClassifier</code> method), 632
<code>default_parameters()</code> ( <code>evalml.pipelines.components.estimators.classifiers.ElasticNetRegressor</code> method), 514	<code>default_parameters()</code> ( <code>evalml.pipelines.components.estimators.ElasticNetRegressor</code> method), 635
<code>default_parameters()</code> ( <code>evalml.pipelines.components.estimators.classifiers.et_classifier.ExtraTreesClassifier</code> method), 485	<code>default_parameters()</code> ( <code>evalml.pipelines.components.estimators.Estimator</code> method), 637
<code>default_parameters()</code> ( <code>evalml.pipelines.components.estimators.classifiers.ExtraTreesClassifier</code> method), 517	<code>default_parameters()</code> ( <code>evalml.pipelines.components.estimator.Estimator</code> method), 609
<code>default_parameters()</code> ( <code>evalml.pipelines.components.estimators.classifiers.kneighbors.KNeighborsClassifier</code> method), 488	<code>default_parameters()</code> ( <code>evalml.pipelines.components.estimators.ExtraTreesClassifier</code> method), 640
<code>default_parameters()</code> ( <code>evalml.pipelines.components.estimators.classifiers.KNeighborsClassifier</code> method), 520	<code>default_parameters()</code> ( <code>evalml.pipelines.components.estimators.ExtraTreesRegressor</code> method), 643
<code>default_parameters()</code> ( <code>evalml.pipelines.components.estimators.classifiers.lightgbm.LightGBMClassifier</code> method), 491	<code>default_parameters()</code> ( <code>evalml.pipelines.components.estimators.KNeighborsClassifier</code> method), 646
<code>default_parameters()</code> ( <code>evalml.pipelines.components.estimators.classifiers.LightGBMClassifier</code> method), 523	<code>default_parameters()</code> ( <code>evalml.pipelines.components.estimators.LightGBMClassifier</code> method), 648
<code>default_parameters()</code> ( <code>evalml.pipelines.components.estimators.classifiers.logistic_regression.LogisticRegressionClassifier</code> method), 494	<code>default_parameters()</code> ( <code>evalml.pipelines.components.estimators.LogisticRegressionClassifier</code> method), 651
<code>default_parameters()</code> ( <code>evalml.pipelines.components.estimators.classifiers.LogisticRegressionClassifier</code> method), 526	<code>default_parameters()</code> ( <code>evalml.pipelines.components.estimators.LinearRegressor</code> method), 654
<code>default_parameters()</code> ( <code>evalml.pipelines.components.estimators.classifiers.RandomForestClassifier</code> method), 529	<code>default_parameters()</code> ( <code>evalml.pipelines.components.estimators.LogisticRegressionClassifier</code> method), 656
<code>default_parameters()</code> ( <code>evalml.pipelines.components.estimators.classifiers.rf_classifier.RandomForestClassifier</code> method), 497	<code>default_parameters()</code> ( <code>evalml.pipelines.components.estimators.ProphetRegressor</code> method), 659
<code>default_parameters()</code> ( <code>evalml.pipelines.components.estimators.classifiers.svm_classifier.SVCClassifier</code> method), 500	<code>default_parameters()</code> ( <code>evalml.pipelines.components.estimators.RandomForestClassifier</code> method), 661
<code>default_parameters()</code> ( <code>evalml.pipelines.components.estimators.classifiers.SVMClassifier</code> method), 531	<code>default_parameters()</code> ( <code>evalml.pipelines.components.estimators.RandomForestRegressor</code> method), 663
<code>default_parameters()</code> ( <code>evalml.pipelines.components.estimators.classifiers.xgboost.XGBoostClassifier</code> method), 503	<code>default_parameters()</code> ( <code>evalml.pipelines.components.estimators.regressors.arima_regressor.ArimaRegressor</code> method), 537

default_parameters() (evalml.pipelines.components.estimators.regressors.ARIMARegressor. method), 576	default_parameters() (evalml.pipelines.components.estimators.regressors.rf_regressor. method), 564
default_parameters() (evalml.pipelines.components.estimators.regressors.baseline. method), 540	default_parameters() (evalml.pipelines.components.estimators.regressors.svm_regressor. method), 567
default_parameters() (evalml.pipelines.components.estimators.regressors.BaselineRegressor. method), 578	default_parameters() (evalml.pipelines.components.estimators.regressors.SVMRegressor. method), 602
default_parameters() (evalml.pipelines.components.estimators.regressors.catboost. method), 543	default_parameters() (evalml.pipelines.components.estimators.regressors.time_series_ method), 569
default_parameters() (evalml.pipelines.components.estimators.regressors.CatBoostRegressor. method), 580	default_parameters() (evalml.pipelines.components.estimators.regressors.TimeSeriesBa method), 604
default_parameters() (evalml.pipelines.components.estimators.regressors.decision. method), 546	default_parameters() (evalml.pipelines.components.estimators.regressors.xgboost_regr method), 572
default_parameters() (evalml.pipelines.components.estimators.regressors.DecisionTreeRegressor. method), 583	default_parameters() (evalml.pipelines.components.estimators.regressors.XGBoostRegr method), 606
default_parameters() (evalml.pipelines.components.estimators.regressors.elasticnet. method), 549	default_parameters() (evalml.pipelines.components.estimators.SVMClassifier method), 666
default_parameters() (evalml.pipelines.components.estimators.regressors.ElasticNetRegressor. method), 586	default_parameters() (evalml.pipelines.components.estimators.SVMRegressor method), 668
default_parameters() (evalml.pipelines.components.estimators.regressors.et_regressor. method), 552	default_parameters() (evalml.pipelines.components.estimators.TimeSeriesBaselineEstim method), 670
default_parameters() (evalml.pipelines.components.estimators.regressors.ExtraTreesRegressor. method), 589	default_parameters() (evalml.pipelines.components.estimators.XGBoostClassifier method), 673
default_parameters() (evalml.pipelines.components.estimators.regressors.lightgbm. method), 555	default_parameters() (evalml.pipelines.components.estimators.XGBoostRegressor method), 676
default_parameters() (evalml.pipelines.components.estimators.regressors.LightGBMRegressor. method), 592	default_parameters() (evalml.pipelines.components.ExtraTreesClassifier method), 959
default_parameters() (evalml.pipelines.components.estimators.regressors.linear_ method), 558	default_parameters() (evalml.pipelines.components.ExtraTreesRegressor method), 962
default_parameters() (evalml.pipelines.components.estimators.regressors.LinearRegressor. method), 594	default_parameters() (evalml.pipelines.components.FeatureSelector method), 964
default_parameters() (evalml.pipelines.components.estimators.regressors.prophet. method), 561	default_parameters() (evalml.pipelines.components.Imputer method), 967
default_parameters() (evalml.pipelines.components.estimators.regressors.ProphetRegressor. method), 597	default_parameters() (evalml.pipelines.components.KNeighborsClassifier method), 970
default_parameters() (evalml.pipelines.components.estimators.regressors.RandomForestRegressor. method), 599	default_parameters() (evalml.pipelines.components.LightGBMClassifier method), 972

<code>default_parameters()</code> ( <i>evalml.pipelines.components.LightGBMRegressor</i> <i>method</i> ), 975	<code>default_parameters()</code> ( <i>evalml.pipelines.components.SklearnStackedEnsembleClassifier</i> <i>method</i> ), 1018
<code>default_parameters()</code> ( <i>evalml.pipelines.components.LinearDiscriminantAnalysis</i> <i>method</i> ), 977	<code>default_parameters()</code> ( <i>evalml.pipelines.components.SklearnStackedEnsembleRegressor</i> <i>method</i> ), 1021
<code>default_parameters()</code> ( <i>evalml.pipelines.components.LinearRegressor</i> <i>method</i> ), 980	<code>default_parameters()</code> ( <i>evalml.pipelines.components.SMOTENCOversampler</i> <i>method</i> ), 1023
<code>default_parameters()</code> ( <i>evalml.pipelines.components.LogisticRegressionClassifier</i> <i>method</i> ), 982	<code>default_parameters()</code> ( <i>evalml.pipelines.components.SMOTENCOversampler</i> <i>method</i> ), 1025
<code>default_parameters()</code> ( <i>evalml.pipelines.components.LogTransformer</i> <i>method</i> ), 984	<code>default_parameters()</code> ( <i>evalml.pipelines.components.SMOTEOversampler</i> <i>method</i> ), 1027
<code>default_parameters()</code> ( <i>evalml.pipelines.components.LSA</i> <i>method</i> ), 987	<code>default_parameters()</code> ( <i>evalml.pipelines.components.StandardScaler</i> <i>method</i> ), 1029
<code>default_parameters()</code> ( <i>evalml.pipelines.components.OneHotEncoder</i> <i>method</i> ), 989	<code>default_parameters()</code> ( <i>evalml.pipelines.components.SVMClassifier</i> <i>method</i> ), 1032
<code>default_parameters()</code> ( <i>evalml.pipelines.components.PCA</i> <i>method</i> ), 992	<code>default_parameters()</code> ( <i>evalml.pipelines.components.SVMRegressor</i> <i>method</i> ), 1034
<code>default_parameters()</code> ( <i>evalml.pipelines.components.PerColumnImputer</i> <i>method</i> ), 994	<code>default_parameters()</code> ( <i>evalml.pipelines.components.TargetEncoder</i> <i>method</i> ), 1036
<code>default_parameters()</code> ( <i>evalml.pipelines.components.PolynomialDetrender</i> <i>method</i> ), 996	<code>default_parameters()</code> ( <i>evalml.pipelines.components.TargetImputer</i> <i>method</i> ), 1039
<code>default_parameters()</code> ( <i>evalml.pipelines.components.ProphetRegressor</i> <i>method</i> ), 999	<code>default_parameters()</code> ( <i>evalml.pipelines.components.TextFeaturizer</i> <i>method</i> ), 1041
<code>default_parameters()</code> ( <i>evalml.pipelines.components.RandomForestClassifier</i> <i>method</i> ), 1001	<code>default_parameters()</code> ( <i>evalml.pipelines.components.TimeSeriesBaselineEstimator</i> <i>method</i> ), 1043
<code>default_parameters()</code> ( <i>evalml.pipelines.components.RandomForestRegressor</i> <i>method</i> ), 1003	<code>default_parameters()</code> ( <i>evalml.pipelines.components.Transformer</i> <i>method</i> ), 1045
<code>default_parameters()</code> ( <i>evalml.pipelines.components.RFClassifierSelectFromModel</i> <i>method</i> ), 1006	<code>default_parameters()</code> ( <i>evalml.pipelines.components.transformers.column_selectors.ColumnSelector</i> <i>method</i> ), 824
<code>default_parameters()</code> ( <i>evalml.pipelines.components.RFRegressorSelectFromModel</i> <i>method</i> ), 1009	<code>default_parameters()</code> ( <i>evalml.pipelines.components.transformers.column_selectors.DropColumnSelector</i> <i>method</i> ), 826
<code>default_parameters()</code> ( <i>evalml.pipelines.components.SelectByType</i> <i>method</i> ), 1011	<code>default_parameters()</code> ( <i>evalml.pipelines.components.transformers.column_selectors.SelectColumnSelector</i> <i>method</i> ), 828
<code>default_parameters()</code> ( <i>evalml.pipelines.components.SelectColumns</i> <i>method</i> ), 1013	<code>default_parameters()</code> ( <i>evalml.pipelines.components.transformers.column_selectors.SelectRowSelector</i> <i>method</i> ), 830
<code>default_parameters()</code> ( <i>evalml.pipelines.components.SimpleImputer</i> <i>method</i> ), 1015	<code>default_parameters()</code> ( <i>evalml.pipelines.components.transformers.DateTimeFeaturizer</i> <i>method</i> ), 838



default_parameters() (evalml.pipelines.components.transformers.DelayedFeatureTransformerPipelineComponent), 841	default_parameters() (evalml.pipelines.components.transformers.feature_selection.RFClassifierPipelineComponent), 716
default_parameters() (evalml.pipelines.components.transformers.DFSTransformerPipelineComponent), 843	default_parameters() (evalml.pipelines.components.transformers.FeatureSelectorPipelineComponent), 851
default_parameters() (evalml.pipelines.components.transformers.dimensionality_reduction_pipeline_base.DiscretizedAsynchrImputerPipelineComponent), 678	default_parameters() (evalml.pipelines.components.transformers.imputers.ImputerPipelineComponent), 854
default_parameters() (evalml.pipelines.components.transformers.dimensionality_reduction_pipeline_base.DiscretizedAsynchrImputerPipelineComponent), 683	default_parameters() (evalml.pipelines.components.transformers.imputers.imputers.ImputerPipelineComponent), 731
default_parameters() (evalml.pipelines.components.transformers.dimensionality_reduction_pipeline_base.DiscretizedAsynchrImputerPipelineComponent), 686	default_parameters() (evalml.pipelines.components.transformers.imputers.imputer.ImputerPipelineComponent), 719
default_parameters() (evalml.pipelines.components.transformers.dimensionality_reduction_pipeline_base.DiscretizedAsynchrImputerPipelineComponent), 681	default_parameters() (evalml.pipelines.components.transformers.imputers.per_column.ImputerPipelineComponent), 722
default_parameters() (evalml.pipelines.components.transformers.DropColumnsPipelineComponent), 845	default_parameters() (evalml.pipelines.components.transformers.imputers.PerColumnImputerPipelineComponent), 733
default_parameters() (evalml.pipelines.components.transformers.DropNullColumnsPipelineComponent), 847	default_parameters() (evalml.pipelines.components.transformers.imputers.simple_imputer.ImputerPipelineComponent), 725
default_parameters() (evalml.pipelines.components.transformers.EmailFeaturizerPipelineComponent), 849	default_parameters() (evalml.pipelines.components.transformers.imputers.SimpleImputerPipelineComponent), 736
default_parameters() (evalml.pipelines.components.transformers.encoders.one_hot_encoder.OneHotEncoderPipelineComponent), 689	default_parameters() (evalml.pipelines.components.transformers.imputers.target_imputer.ImputerPipelineComponent), 728
default_parameters() (evalml.pipelines.components.transformers.encoders.OneHotEncoderPipelineComponent), 696	default_parameters() (evalml.pipelines.components.transformers.imputers.TargetImputerPipelineComponent), 738
default_parameters() (evalml.pipelines.components.transformers.encoders.target_encoder.TargetEncoderPipelineComponent), 693	default_parameters() (evalml.pipelines.components.transformers.LinearDiscriminantAnalysisPipelineComponent), 856
default_parameters() (evalml.pipelines.components.transformers.encoders.TargetEncoderPipelineComponent), 699	default_parameters() (evalml.pipelines.components.transformers.LogTransformerPipelineComponent), 858
default_parameters() (evalml.pipelines.components.transformers.feature_selection.feature_selector.FeatureSelectorPipelineComponent), 701	default_parameters() (evalml.pipelines.components.transformers.LSAPipelineComponent), 860
default_parameters() (evalml.pipelines.components.transformers.feature_selection.feature_selector.FeatureSelectorPipelineComponent), 711	default_parameters() (evalml.pipelines.components.transformers.OneHotEncoderPipelineComponent), 863
default_parameters() (evalml.pipelines.components.transformers.feature_selection.feature_selector.FeatureSelectorPipelineComponent), 705	default_parameters() (evalml.pipelines.components.transformers.RFClassifierPipelineComponent), 865
default_parameters() (evalml.pipelines.components.transformers.feature_selection.feature_selector.FeatureSelectorPipelineComponent), 708	default_parameters() (evalml.pipelines.components.transformers.RFRegressorPipelineComponent), 868
default_parameters() (evalml.pipelines.components.transformers.feature_selection.feature_selector.FeatureSelectorPipelineComponent), 714	default_parameters() (evalml.pipelines.components.transformers.PolynomialDetrenderPipelineComponent), 870

Index 1339

default_parameters() (evalml.pipelines.components.transformers.SelectByType method), 878	default_parameters() (evalml.pipelines.components.XGBoostRegressor method), 1055
default_parameters() (evalml.pipelines.components.transformers.SelectColumns method), 880	default_parameters() (evalml.pipelines.DecisionTreeClassifier method), 1117
default_parameters() (evalml.pipelines.components.transformers.SimpleImputer method), 882	default_parameters() (evalml.pipelines.DecisionTreeRegressor method), 1119
default_parameters() (evalml.pipelines.components.transformers.SMOTENCOversampler method), 885	default_parameters() (evalml.pipelines.DelayedFeatureTransformer method), 1122
default_parameters() (evalml.pipelines.components.transformers.SMOTENCOversampler method), 887	default_parameters() (evalml.pipelines.DFSTransformer method), 1124
default_parameters() (evalml.pipelines.components.transformers.SMOTEOverSampler method), 889	default_parameters() (evalml.pipelines.ElasticNetClassifier method), 1127
default_parameters() (evalml.pipelines.components.transformers.StandardScaler method), 891	default_parameters() (evalml.pipelines.ElasticNetRegressor method), 1129
default_parameters() (evalml.pipelines.components.transformers.TargetEncoder method), 894	default_parameters() (evalml.pipelines.Estimator method), 1131
default_parameters() (evalml.pipelines.components.transformers.TargetImputer method), 896	default_parameters() (evalml.pipelines.ExtraTreesClassifier method), 1134
default_parameters() (evalml.pipelines.components.transformers.TextFeaturizer method), 898	default_parameters() (evalml.pipelines.ExtraTreesRegressor method), 1137
default_parameters() (evalml.pipelines.components.transformers.Transformer method), 901	default_parameters() (evalml.pipelines.FeatureSelector method), 1139
default_parameters() (evalml.pipelines.components.transformers.transformer.TargetEncoder method), 832	default_parameters() (evalml.pipelines.KNeighborsClassifier method), 1142
default_parameters() (evalml.pipelines.components.transformers.transformer.Transformer method), 835	default_parameters() (evalml.pipelines.LightGBMClassifier method), 1145
default_parameters() (evalml.pipelines.components.transformers.Undersampler method), 903	default_parameters() (evalml.pipelines.LightGBMRegressor method), 1148
default_parameters() (evalml.pipelines.components.transformers.URLFeaturizer method), 905	default_parameters() (evalml.pipelines.LinearRegressor method), 1150
default_parameters() (evalml.pipelines.components.Undersampler method), 1048	default_parameters() (evalml.pipelines.LogisticRegressionClassifier method), 1152
default_parameters() (evalml.pipelines.components.URLFeaturizer method), 1050	default_parameters() (evalml.pipelines.OneHotEncoder method), 1159
default_parameters() (evalml.pipelines.components.XGBoostClassifier method), 1052	default_parameters() (evalml.pipelines.PerColumnImputer method), 1162
	default_parameters()



(evalml.pipelines.ProphetRegressor method), 1168  
 default\_parameters() (evalml.pipelines.RandomForestClassifier method), 1170  
 default\_parameters() (evalml.pipelines.RandomForestRegressor method), 1172  
 default\_parameters() (evalml.pipelines.RFClassifierSelectFromModel method), 1179  
 default\_parameters() (evalml.pipelines.RFRegressorSelectFromModel method), 1181  
 default\_parameters() (evalml.pipelines.SimpleImputer method), 1184  
 default\_parameters() (evalml.pipelines.SklearnStackedEnsembleClassifier method), 1186  
 default\_parameters() (evalml.pipelines.SklearnStackedEnsembleRegressor method), 1189  
 default\_parameters() (evalml.pipelines.StandardScaler method), 1191  
 default\_parameters() (evalml.pipelines.SVMClassifier method), 1193  
 default\_parameters() (evalml.pipelines.SVMRegressor method), 1195  
 default\_parameters() (evalml.pipelines.TargetEncoder method), 1198  
 default\_parameters() (evalml.pipelines.Transformer method), 1217  
 default\_parameters() (evalml.pipelines.XGBoostClassifier method), 1219  
 default\_parameters() (evalml.pipelines.XGBoostRegressor method), 1222  
 DefaultDataChecks (class in evalml.data\_checks), 234  
 DefaultDataChecks (class in evalml.data\_checks.default\_data\_checks), 210  
 DelayedFeatureTransformer (class in evalml.pipelines), 1121  
 DelayedFeatureTransformer (class in evalml.pipelines.components), 939  
 DelayedFeatureTransformer (class in evalml.pipelines.components.transformers), 840  
 DelayedFeatureTransformer (class in evalml.pipelines.components.transformers.preprocessing), 771  
 DelayedFeatureTransformer (class in evalml.pipelines.components.transformers.preprocessing.delayed), 743  
 deprecate\_arg() (in module evalml.utils), 1259  
 deprecate\_arg() (in module evalml.utils.gen\_utils), 1254  
 describe() (evalml.pipelines.ARIMARegressor method), 1106  
 describe() (evalml.pipelines.binary\_classification\_pipeline.BinaryClassifier method), 1058  
 describe() (evalml.pipelines.CatBoostClassifier method), 1109  
 describe() (evalml.pipelines.CatBoostRegressor method), 1111  
 describe() (evalml.pipelines.classification\_pipeline.ClassificationPipeline method), 1064  
 describe() (evalml.pipelines.component\_graph.ComponentGraph method), 1068  
 describe() (evalml.pipelines.ComponentGraph method), 1114  
 describe() (evalml.pipelines.components.ARIMARegressor method), 918  
 describe() (evalml.pipelines.components.BaselineClassifier method), 920  
 describe() (evalml.pipelines.components.BaselineRegressor method), 922  
 describe() (evalml.pipelines.components.CatBoostClassifier method), 925  
 describe() (evalml.pipelines.components.CatBoostRegressor method), 927  
 describe() (evalml.pipelines.components.component\_base.ComponentBase method), 908  
 describe() (evalml.pipelines.components.ComponentBase method), 929  
 describe() (evalml.pipelines.components.DateTimeFeaturizer method), 932  
 describe() (evalml.pipelines.components.DecisionTreeClassifier method), 935  
 describe() (evalml.pipelines.components.DecisionTreeRegressor method), 938  
 describe() (evalml.pipelines.components.DelayedFeatureTransformer method), 941  
 describe() (evalml.pipelines.components.DFSTransformer method), 943  
 describe() (evalml.pipelines.components.DropColumns method), 945  
 describe() (evalml.pipelines.components.DropNullColumns method), 947  
 describe() (evalml.pipelines.components.ElasticNetClassifier method), 950

`describe()` (`evalml.pipelines.components.ElasticNetRegressor` method), 952

`describe()` (`evalml.pipelines.components.EmailFeaturizer` method), 954

`describe()` (`evalml.pipelines.components.ensemble.sklearn_stacked_ensemble_base` method), 455

`describe()` (`evalml.pipelines.components.ensemble.sklearn_stacked_ensemble_classifier` method), 458

`describe()` (`evalml.pipelines.components.ensemble.sklearn_stacked_ensemble_regressor` method), 461

`describe()` (`evalml.pipelines.components.ensemble.SklearnStackedEnsembleClassifier` method), 464

`describe()` (`evalml.pipelines.components.ensemble.SklearnStackedEnsembleClassifier` method), 467

`describe()` (`evalml.pipelines.components.ensemble.SklearnStackedEnsembleRegressor` method), 470

`describe()` (`evalml.pipelines.components.Estimator` method), 957

`describe()` (`evalml.pipelines.components.estimators.ARIMARegressor` method), 613

`describe()` (`evalml.pipelines.components.estimators.BaseClassifier` method), 616

`describe()` (`evalml.pipelines.components.estimators.BaseRegressor` method), 618

`describe()` (`evalml.pipelines.components.estimators.CatBoostClassifier` method), 621

`describe()` (`evalml.pipelines.components.estimators.CatBoostRegressor` method), 623

`describe()` (`evalml.pipelines.components.estimators.classifiers.base_line_classifier` method), 472

`describe()` (`evalml.pipelines.components.estimators.classifiers.BaseLineClassifier` method), 506

`describe()` (`evalml.pipelines.components.estimators.classifiers.boosted_classifier` method), 475

`describe()` (`evalml.pipelines.components.estimators.classifiers.CatBoostClassifier` method), 508

`describe()` (`evalml.pipelines.components.estimators.classifiers.decision_tree_classifier` method), 478

`describe()` (`evalml.pipelines.components.estimators.classifiers.DecisionTreeClassifier` method), 511

`describe()` (`evalml.pipelines.components.estimators.classifiers.elastic_net_classifier` method), 482

`describe()` (`evalml.pipelines.components.estimators.classifiers.ElasticNetClassifier` method), 514

`describe()` (`evalml.pipelines.components.estimators.classifiers.extreme_tree_classifier` method), 485

`describe()` (`evalml.pipelines.components.estimators.classifiers.ExtraTreesClassifier` method), 517

`describe()` (`evalml.pipelines.components.estimators.classifiers.kneighbors_classifier` method), 488

`describe()` (`evalml.pipelines.components.estimators.classifiers.KNeighborsClassifier` method), 520

`describe()` (`evalml.pipelines.components.estimators.classifiers.lightgbm_classifier` method), 491

`describe()` (`evalml.pipelines.components.estimators.classifiers.LightGBMClassifier` method), 523

`describe()` (`evalml.pipelines.components.estimators.classifiers.logistic_regression_classifier` method), 494

`describe()` (`evalml.pipelines.components.estimators.classifiers.LogisticRegressionClassifier` method), 526

`describe()` (`evalml.pipelines.components.estimators.classifiers.RandomForestClassifier` method), 529

`describe()` (`evalml.pipelines.components.estimators.classifiers.svm_classifier` method), 497

`describe()` (`evalml.pipelines.components.estimators.classifiers.SVMClassifier` method), 500

`describe()` (`evalml.pipelines.components.estimators.classifiers.SVMClassifier` method), 531

`describe()` (`evalml.pipelines.components.estimators.classifiers.xgboost_classifier` method), 503

`describe()` (`evalml.pipelines.components.estimators.classifiers.XGBoostClassifier` method), 534

`describe()` (`evalml.pipelines.components.estimators.DecisionTreeRegressor` method), 626

`describe()` (`evalml.pipelines.components.estimators.DecisionTreeRegressor` method), 629

`describe()` (`evalml.pipelines.components.estimators.ElasticNetClassifier` method), 632

`describe()` (`evalml.pipelines.components.estimators.ElasticNetRegressor` method), 635

`describe()` (`evalml.pipelines.components.estimators.Estimator` method), 637

`describe()` (`evalml.pipelines.components.estimators.estimator.Estimator` method), 609

`describe()` (`evalml.pipelines.components.estimators.ExtraTreesClassifier` method), 640

`describe()` (`evalml.pipelines.components.estimators.ExtraTreesClassifier` method), 643

`describe()` (`evalml.pipelines.components.estimators.KNeighborsClassifier` method), 646

`describe()` (`evalml.pipelines.components.estimators.KNeighborsClassifier` method), 649

`describe()` (`evalml.pipelines.components.estimators.LightGBMClassifier` method), 651

`describe()` (`evalml.pipelines.components.estimators.LightGBMRegressor` method), 654

`describe()` (`evalml.pipelines.components.estimators.LogisticRegressionClassifier` method), 656

`describe()` (`evalml.pipelines.components.estimators.ProphetRegressor` method), 659

`describe()` (`evalml.pipelines.components.estimators.RandomForestClassifier` method), 661

`describe()` (`evalml.pipelines.components.estimators.RandomForestRegressor` method), 664

`describe()` (`evalml.pipelines.components.estimators.regressors.arima_regressor` method), 537

`describe()` (`evalml.pipelines.components.estimators.regressors.ARIMARegressor` method), 576

`describe()` (`evalml.pipelines.components.estimators.regressor.baseline` (`evalml.pipelines.components.estimators.XGBoostClassifier` method), 540  
`describe()` (`evalml.pipelines.components.estimators.regressor.baseline` (`evalml.pipelines.components.estimators.XGBoostRegressor` method), 578  
`describe()` (`evalml.pipelines.components.estimators.regressor.catboost` (`evalml.pipelines.components.ExtraTreesClassifier` method), 543  
`describe()` (`evalml.pipelines.components.estimators.regressor.catboost` (`evalml.pipelines.components.ExtraTreesRegressor` method), 581  
`describe()` (`evalml.pipelines.components.estimators.regressor.decisiontree` (`evalml.pipelines.components.RegressorSelector` method), 546  
`describe()` (`evalml.pipelines.components.estimators.regressor.decisiontree` (`evalml.pipelines.components.Imputer` method), 583  
`describe()` (`evalml.pipelines.components.estimators.regressor.elasticnet` (`evalml.pipelines.components.KNeighborsClassifier` method), 549  
`describe()` (`evalml.pipelines.components.estimators.regressor.elasticnet` (`evalml.pipelines.components.LightGBMClassifier` method), 586  
`describe()` (`evalml.pipelines.components.estimators.regressor.elasticnet` (`evalml.pipelines.components.LightGBMRegressor` method), 552  
`describe()` (`evalml.pipelines.components.estimators.regressor.extra_trees` (`evalml.pipelines.components.LinearDiscriminantAnalysis` method), 589  
`describe()` (`evalml.pipelines.components.estimators.regressor.lightgbm` (`evalml.pipelines.components.LinearRegressor` method), 555  
`describe()` (`evalml.pipelines.components.estimators.regressor.lightgbm` (`evalml.pipelines.components.LogisticRegressionClassifier` method), 592  
`describe()` (`evalml.pipelines.components.estimators.regressor.linear` (`evalml.pipelines.components.LogTransformer` method), 558  
`describe()` (`evalml.pipelines.components.estimators.regressor.linear` (`evalml.pipelines.components.LSA` method), 594  
`describe()` (`evalml.pipelines.components.estimators.regressor.prophet` (`evalml.pipelines.components.OneHotEncoder` method), 561  
`describe()` (`evalml.pipelines.components.estimators.regressor.prophet` (`evalml.pipelines.components.PCA` method), 597  
`describe()` (`evalml.pipelines.components.estimators.regressor.random_forest` (`evalml.pipelines.components.PerColumnImputer` method), 599  
`describe()` (`evalml.pipelines.components.estimators.regressor.if_regression` (`evalml.pipelines.components.PolynomialDetrender` method), 564  
`describe()` (`evalml.pipelines.components.estimators.regressor.svm` (`evalml.pipelines.components.ProphetRegressor` method), 567  
`describe()` (`evalml.pipelines.components.estimators.regressor.svm` (`evalml.pipelines.components.RandomForestClassifier` method), 602  
`describe()` (`evalml.pipelines.components.estimators.regressor.time_series` (`evalml.pipelines.components.RandomBaselineRegressor` method), 569  
`describe()` (`evalml.pipelines.components.estimators.regressor.time_series` (`evalml.pipelines.components.RFClassifierSelectFromModel` method), 604  
`describe()` (`evalml.pipelines.components.estimators.regressor.xgboost` (`evalml.pipelines.components.RFRegressorSelectFromModel` method), 572  
`describe()` (`evalml.pipelines.components.estimators.regressor.xgboost` (`evalml.pipelines.components.SelectByType` method), 606  
`describe()` (`evalml.pipelines.components.estimators.SVMClassifier` (`evalml.pipelines.components.SelectColumns` method), 666  
`describe()` (`evalml.pipelines.components.estimators.SVMRegressor` (`evalml.pipelines.components.SimpleImputer` method), 668  
`describe()` (`evalml.pipelines.components.estimators.TimeSeriesBaseline` (`evalml.pipelines.components.SklearnStackedEnsembleClassifier` method), 671

`describe()` (`evalml.pipelines.components.SklearnStackedEnsembleRegressor`), 1021  
`describe()` (`evalml.pipelines.components.SMOTEOverSampler`), 1023  
`describe()` (`evalml.pipelines.components.SMOTEOverSampler`), 1025  
`describe()` (`evalml.pipelines.components.SMOTEOverSampler`), 1027  
`describe()` (`evalml.pipelines.components.StandardScaler`), 1029  
`describe()` (`evalml.pipelines.components.SVMClassifier`), 1032  
`describe()` (`evalml.pipelines.components.SVMRegressor`), 1034  
`describe()` (`evalml.pipelines.components.TargetEncoder`), 1036  
`describe()` (`evalml.pipelines.components.TargetImputer`), 1039  
`describe()` (`evalml.pipelines.components.TextFeaturizer`), 1041  
`describe()` (`evalml.pipelines.components.TimeSeriesBaselineEstimator`), 1043  
`describe()` (`evalml.pipelines.components.Transformer`), 1045  
`describe()` (`evalml.pipelines.components.transformers.ColumnSelector`), 824  
`describe()` (`evalml.pipelines.components.transformers.ColumnSelector`), 826  
`describe()` (`evalml.pipelines.components.transformers.ColumnSelector`), 828  
`describe()` (`evalml.pipelines.components.transformers.ColumnSelector`), 830  
`describe()` (`evalml.pipelines.components.transformers.DateTimeFeaturizer`), 838  
`describe()` (`evalml.pipelines.components.transformers.DelayedFeatureTransformer`), 841  
`describe()` (`evalml.pipelines.components.transformers.DESTransformer`), 843  
`describe()` (`evalml.pipelines.components.transformers.DimensionalityReductionPipeline`), 678  
`describe()` (`evalml.pipelines.components.transformers.DimensionalityReductionPipeline`), 683  
`describe()` (`evalml.pipelines.components.transformers.DimensionalityReductionPipeline`), 686  
`describe()` (`evalml.pipelines.components.transformers.DimensionalityReductionPipeline`), 681  
`describe()` (`evalml.pipelines.components.transformers.DropColumns`), 845  
`describe()` (`evalml.pipelines.components.transformers.DropNullColumns`), 847  
`describe()` (`evalml.pipelines.components.transformers.ElasticFeature`), 849  
`describe()` (`evalml.pipelines.components.transformers.Encoder`), 689  
`describe()` (`evalml.pipelines.components.transformers.encoders.OneHotEncoder`), 696  
`describe()` (`evalml.pipelines.components.transformers.encoders.target_encoder`), 693  
`describe()` (`evalml.pipelines.components.transformers.encoders.TargetEncoder`), 699  
`describe()` (`evalml.pipelines.components.transformers.feature_selection`), 702  
`describe()` (`evalml.pipelines.components.transformers.feature_selection`), 711  
`describe()` (`evalml.pipelines.components.transformers.feature_selection`), 705  
`describe()` (`evalml.pipelines.components.transformers.feature_selection`), 708  
`describe()` (`evalml.pipelines.components.transformers.feature_selection`), 714  
`describe()` (`evalml.pipelines.components.transformers.feature_selection`), 716  
`describe()` (`evalml.pipelines.components.transformers.FeatureSelector`), 852  
`describe()` (`evalml.pipelines.components.transformers.Imputer`), 854  
`describe()` (`evalml.pipelines.components.transformers.imputers.Imputer`), 731  
`describe()` (`evalml.pipelines.components.transformers.imputers.impute`), 719  
`describe()` (`evalml.pipelines.components.transformers.imputers.per_column`), 722  
`describe()` (`evalml.pipelines.components.transformers.imputers.PerColumn`), 734  
`describe()` (`evalml.pipelines.components.transformers.imputers.simple`), 725  
`describe()` (`evalml.pipelines.components.transformers.imputers.Simple`), 736  
`describe()` (`evalml.pipelines.components.transformers.imputers.target`), 728  
`describe()` (`evalml.pipelines.components.transformers.imputers.Target`), 738  
`describe()` (`evalml.pipelines.components.transformers.LinearDiscriminantAnalysis`), 856  
`describe()` (`evalml.pipelines.components.transformers.LogTransformer`), 858  
`describe()` (`evalml.pipelines.components.transformers.LSA`), 860  
`describe()` (`evalml.pipelines.components.transformers.OneHotEncoder`), 863  
`describe()` (`evalml.pipelines.components.transformers.PCA`), 866  
`describe()` (`evalml.pipelines.components.transformers.PerColumnImputer`), 868  
`describe()` (`evalml.pipelines.components.transformers.PolynomialDetector`), 870  
`describe()` (`evalml.pipelines.components.transformers.preprocessing`), 741



`describe()` (`evalml.pipelines.components.transformers.preprocessing.DelayedFeatureTransformer` components.transformers.samplers.oversampler), 769  
`method`), 804  
`describe()` (`evalml.pipelines.components.transformers.preprocessing.DelayedFeatureTransformer` components.transformers.samplers.SMOTE), 744  
`method`), 809  
`describe()` (`evalml.pipelines.components.transformers.preprocessing.DelayedFeatureTransformer` components.transformers.samplers.SMOTE), 772  
`method`), 812  
`describe()` (`evalml.pipelines.components.transformers.preprocessing.DelayedFeatureTransformer` components.transformers.samplers.SMOTE), 774  
`method`), 814  
`describe()` (`evalml.pipelines.components.transformers.preprocessing.DropNullColumns` components.transformers.samplers.UnderSampler), 747  
`method`), 817  
`describe()` (`evalml.pipelines.components.transformers.preprocessing.DropNullColumns` components.transformers.samplers.UnderSampler), 776  
`method`), 807  
`describe()` (`evalml.pipelines.components.transformers.preprocessing.EvalMLPipeline` components.transformers.samplers.UnderSampler), 778  
`method`), 819  
`describe()` (`evalml.pipelines.components.transformers.preprocessing.FeatureSelector` components.transformers.samplers.UnderSampler), 749  
`method`), 821  
`describe()` (`evalml.pipelines.components.transformers.preprocessing.LogTransformer` components.transformers.samplers.UnderSampler), 751  
`method`), 878  
`describe()` (`evalml.pipelines.components.transformers.preprocessing.LogTransformer` components.transformers.samplers.UnderSampler), 780  
`method`), 880  
`describe()` (`evalml.pipelines.components.transformers.preprocessing.LSA` components.transformers.samplers.UnderSampler), 782  
`method`), 882  
`describe()` (`evalml.pipelines.components.transformers.preprocessing.LSML` components.transformers.samplers.UnderSampler), 754  
`method`), 885  
`describe()` (`evalml.pipelines.components.transformers.preprocessing.PolynomialDefender` components.transformers.samplers.UnderSampler), 756  
`method`), 887  
`describe()` (`evalml.pipelines.components.transformers.preprocessing.PolynomialDefender` components.transformers.samplers.UnderSampler), 785  
`method`), 889  
`describe()` (`evalml.pipelines.components.transformers.preprocessing.TargetEncoder` components.transformers.samplers.UnderSampler), 759  
`method`), 891  
`describe()` (`evalml.pipelines.components.transformers.preprocessing.TargetEncoder` components.transformers.samplers.UnderSampler), 762  
`method`), 894  
`describe()` (`evalml.pipelines.components.transformers.preprocessing.TargetImputer` components.transformers.samplers.UnderSampler), 787  
`method`), 896  
`describe()` (`evalml.pipelines.components.transformers.preprocessing.TargetImputer` components.transformers.samplers.UnderSampler), 789  
`method`), 898  
`describe()` (`evalml.pipelines.components.transformers.preprocessing.TargetImputer` components.transformers.samplers.UnderSampler), 764  
`method`), 901  
`describe()` (`evalml.pipelines.components.transformers.preprocessing.TargetImputer` components.transformers.samplers.UnderSampler), 766  
`method`), 832  
`describe()` (`evalml.pipelines.components.transformers.preprocessing.TargetImputer` components.transformers.samplers.UnderSampler), 791  
`method`), 835  
`describe()` (`evalml.pipelines.components.transformers.preprocessing.TargetImputer` components.transformers.samplers.UnderSampler), 873  
`method`), 903  
`describe()` (`evalml.pipelines.components.transformers.preprocessing.TargetImputer` components.transformers.samplers.UnderSampler), 876  
`method`), 905  
`describe()` (`evalml.pipelines.components.transformers.preprocessing.TargetImputer` components.transformers.samplers.UnderSampler), 794  
`method`), 1048  
`describe()` (`evalml.pipelines.components.transformers.preprocessing.TargetImputer` components.transformers.samplers.UnderSampler), 796  
`method`), 1050  
`describe()` (`evalml.pipelines.components.transformers.preprocessing.TargetImputer` components.transformers.samplers.UnderSampler), 799  
`method`), 1052  
`describe()` (`evalml.pipelines.components.transformers.preprocessing.TargetImputer` components.transformers.samplers.UnderSampler), 802  
`method`), 1055

<code>describe()</code> ( <i>evalml.pipelines.DecisionTreeClassifier method</i> ), 1117	<code>describe()</code> ( <i>evalml.pipelines.RFRegressorSelectFromModel method</i> ), 1182
<code>describe()</code> ( <i>evalml.pipelines.DecisionTreeRegressor method</i> ), 1120	<code>describe()</code> ( <i>evalml.pipelines.SimpleImputer method</i> ), 1184
<code>describe()</code> ( <i>evalml.pipelines.DelayedFeatureTransformer method</i> ), 1122	<code>describe()</code> ( <i>evalml.pipelines.SklearnStackedEnsembleClassifier method</i> ), 1186
<code>describe()</code> ( <i>evalml.pipelines.DFSTransformer method</i> ), 1124	<code>describe()</code> ( <i>evalml.pipelines.SklearnStackedEnsembleRegressor method</i> ), 1189
<code>describe()</code> ( <i>evalml.pipelines.ElasticNetClassifier method</i> ), 1127	<code>describe()</code> ( <i>evalml.pipelines.StandardScaler method</i> ), 1191
<code>describe()</code> ( <i>evalml.pipelines.ElasticNetRegressor method</i> ), 1129	<code>describe()</code> ( <i>evalml.pipelines.SVMClassifier method</i> ), 1193
<code>describe()</code> ( <i>evalml.pipelines.Estimator method</i> ), 1132	<code>describe()</code> ( <i>evalml.pipelines.SVMRegressor method</i> ), 1196
<code>describe()</code> ( <i>evalml.pipelines.ExtraTreesClassifier method</i> ), 1134	<code>describe()</code> ( <i>evalml.pipelines.TargetEncoder method</i> ), 1198
<code>describe()</code> ( <i>evalml.pipelines.ExtraTreesRegressor method</i> ), 1137	<code>describe()</code> ( <i>evalml.pipelines.time_series_classification_pipelines.TimeSeriesClassificationPipeline method</i> ), 1086
<code>describe()</code> ( <i>evalml.pipelines.FeatureSelector method</i> ), 1139	<code>describe()</code> ( <i>evalml.pipelines.time_series_classification_pipelines.TimeSeriesClassificationPipeline method</i> ), 1091
<code>describe()</code> ( <i>evalml.pipelines.KNeighborsClassifier method</i> ), 1142	<code>describe()</code> ( <i>evalml.pipelines.time_series_classification_pipelines.TimeSeriesClassificationPipeline method</i> ), 1095
<code>describe()</code> ( <i>evalml.pipelines.LightGBMClassifier method</i> ), 1145	<code>describe()</code> ( <i>evalml.pipelines.time_series_regression_pipeline.TimeSeriesRegressionPipeline method</i> ), 1100
<code>describe()</code> ( <i>evalml.pipelines.LightGBMRegressor method</i> ), 1148	<code>describe()</code> ( <i>evalml.pipelines.TimeSeriesBinaryClassificationPipeline method</i> ), 1201
<code>describe()</code> ( <i>evalml.pipelines.LinearRegressor method</i> ), 1150	<code>describe()</code> ( <i>evalml.pipelines.TimeSeriesClassificationPipeline method</i> ), 1205
<code>describe()</code> ( <i>evalml.pipelines.LogisticRegressionClassifier method</i> ), 1153	<code>describe()</code> ( <i>evalml.pipelines.TimeSeriesMulticlassClassificationPipeline method</i> ), 1210
<code>describe()</code> ( <i>evalml.pipelines.multiclass_classification_pipeline.MulticlassClassificationPipeline method</i> ), 1072	<code>describe()</code> ( <i>evalml.pipelines.TimeSeriesRegressionPipeline method</i> ), 1214
<code>describe()</code> ( <i>evalml.pipelines.MulticlassClassificationPipeline method</i> ), 1156	<code>describe()</code> ( <i>evalml.pipelines.Transformer method</i> ), 1217
<code>describe()</code> ( <i>evalml.pipelines.OneHotEncoder method</i> ), 1159	<code>describe()</code> ( <i>evalml.pipelines.XGBoostClassifier method</i> ), 1220
<code>describe()</code> ( <i>evalml.pipelines.PerColumnImputer method</i> ), 1162	<code>describe()</code> ( <i>evalml.pipelines.XGBoostRegressor method</i> ), 1222
<code>describe()</code> ( <i>evalml.pipelines.pipeline_base.PipelineBase method</i> ), 1076	<code>describe_pipeline()</code> ( <i>evalml.automl.automl_search.AutoMLSearch method</i> ), 184
<code>describe()</code> ( <i>evalml.pipelines.PipelineBase method</i> ), 1165	<code>describe_pipeline()</code> ( <i>evalml.automl.AutoMLSearch method</i> ), 194
<code>describe()</code> ( <i>evalml.pipelines.ProphetRegressor method</i> ), 1168	<code>describe_pipeline()</code> ( <i>evalml.AutoMLSearch method</i> ), 1265
<code>describe()</code> ( <i>evalml.pipelines.RandomForestClassifier method</i> ), 1170	<code>detect_problem_type()</code> (in module <i>evalml.problem_types</i> ), 1239
<code>describe()</code> ( <i>evalml.pipelines.RandomForestRegressor method</i> ), 1173	<code>detect_problem_type()</code> (in module <i>evalml.problem_types.utils</i> ), 1237
<code>describe()</code> ( <i>evalml.pipelines.regression_pipeline.RegressionPipeline method</i> ), 1082	<code>DFSTransformer</code> (class in <i>evalml.pipelines</i> ), 1123
<code>describe()</code> ( <i>evalml.pipelines.RegressionPipeline method</i> ), 1175	<code>DFSTransformer</code> (class in <i>evalml.pipelines.components</i> ), 942
<code>describe()</code> ( <i>evalml.pipelines.RFClassifierSelectFromModel method</i> ), 1179	<code>DFSTransformer</code> (class in <i>evalml.pipelines.components</i> ), 942

[evalml.pipelines.components.transformers\), 842](#)  
[ElasticNetClassifier \(class in evalml.pipelines.components.estimators.classifiers\), 512](#)  
[DFSTransformer \(class in evalml.pipelines.components.transformers.preprocessing\), 773](#)  
[DFSTransformer \(class in evalml.pipelines.components.estimators.classifiers.elasticnet\\_classifier\), 480](#)  
[DFSTransformer \(class in evalml.pipelines.components.transformers.preprocessing\), 748](#)  
[ElasticNetRegressor \(class in evalml.pipelines\), 1128](#)  
[done\(\) \(evalml.automl.engine.cf\\_engine.CFComputation method\), 167](#)  
[done\(\) \(evalml.automl.engine.dask\\_engine.DaskComputation method\), 169](#)  
[done\(\) \(evalml.automl.engine.engine\\_base.EngineComputation method\), 172](#)  
[done\(\) \(evalml.automl.engine.EngineComputation method\), 178](#)  
[done\(\) \(evalml.automl.engine.sequential\\_engine.SequentialComputation method\), 174](#)  
[drop\\_nan\\_target\\_rows\(\) \(in module evalml.preprocessing\), 1233](#)  
[drop\\_nan\\_target\\_rows\(\) \(in module evalml.preprocessing.utils\), 1231](#)  
[drop\\_rows\\_with\\_nans\(\) \(in module evalml.utils\), 1259](#)  
[drop\\_rows\\_with\\_nans\(\) \(in module evalml.utils.gen\\_utils\), 1254](#)  
[DropColumns \(class in evalml.pipelines.components\), 944](#)  
[DropColumns \(class in evalml.pipelines.components.transformers\), 844](#)  
[DropColumns \(class in evalml.pipelines.components.transformers.column\\_selectors\), 825](#)  
[DropNullColumns \(class in evalml.pipelines.components\), 946](#)  
[DropNullColumns \(class in evalml.pipelines.components.transformers\), 846](#)  
[DropNullColumns \(class in evalml.pipelines.components.transformers.preprocessing\), 775](#)  
[DropNullColumns \(class in evalml.pipelines.components.transformers.preprocessing.drop\\_null\\_columns\), 746](#)  
**E**  
[ElasticNetClassifier \(class in evalml.pipelines\), 1125](#)  
[ElasticNetClassifier \(class in evalml.pipelines.components\), 948](#)  
[ElasticNetClassifier \(class in evalml.pipelines.components.estimators\), 630](#)  
[ElasticNetClassifier \(class in evalml.pipelines.components.estimators.classifiers\), 512](#)  
[ElasticNetClassifier \(class in evalml.pipelines.components.estimators.classifiers.elasticnet\\_classifier\), 480](#)  
[ElasticNetRegressor \(class in evalml.pipelines\), 1128](#)  
[ElasticNetRegressor \(class in evalml.pipelines.components\), 951](#)  
[ElasticNetRegressor \(class in evalml.pipelines.components.estimators\), 633](#)  
[ElasticNetRegressor \(class in evalml.pipelines.components.estimators.regressors\), 585](#)  
[ElasticNetRegressor \(class in evalml.pipelines.components.estimators.regressors.elasticnet\\_regressor\), 548](#)  
[EmailFeaturizer \(class in evalml.pipelines.components\), 953](#)  
[EmailFeaturizer \(class in evalml.pipelines.components.transformers\), 848](#)  
[EmailFeaturizer \(class in evalml.pipelines.components.transformers.preprocessing\), 777](#)  
[EmailFeaturizer \(class in evalml.pipelines.components.transformers.preprocessing.transformers\), 764](#)  
[EmptyDataChecks \(class in evalml.data\\_checks\), 235](#)  
[EmptyDataChecks \(class in evalml.data\\_checks.utils\), 225](#)  
[EngineBase \(class in evalml.automl\), 196](#)  
[EngineBase \(class in evalml.automl.engine\), 178](#)  
[EngineBase \(class in evalml.automl.engine.engine\\_base\), 171](#)  
[EngineComputation \(class in evalml.automl.engine\), 178](#)  
[EngineComputation \(class in evalml.automl.engine.engine\\_base\), 171](#)  
[EnsembleMissingPipelinesError, 250, 252](#)  
[error\(\) \(evalml.automl.engine.engine\\_base.JobLogger method\), 250](#)  
[error\\_contains\\_nan \(in module evalml.data\\_checks.datetime\\_nan\\_data\\_check\), 209](#)  
[error\\_contains\\_nan \(in module evalml.data\\_checks.natural\\_language\\_nan\\_data\\_check\), 216](#)  
[Estimator \(class in evalml.pipelines\), 1130](#)  
[Estimator \(class in evalml.pipelines.components\), 955](#)  
[Estimator \(class in evalml.pipelines.components.estimators\), 630](#)

636  
 Estimator (class in `evalml.pipelines.components.estimators.estimator`)  
 608

`evalml`  
 module, 156

`evalml.automl`  
 module, 156

`evalml.automl.automl_algorithm`  
 module, 156

`evalml.automl.automl_algorithm.automl_algorithm`  
 module, 156

`evalml.automl.automl_algorithm.evalml_algorithm`  
 module, 157

`evalml.automl.automl_algorithm.iterative_algorithm`  
 module, 159

`evalml.automl.automl_search`  
 module, 180

`evalml.automl.callbacks`  
 module, 187

`evalml.automl.engine`  
 module, 166

`evalml.automl.engine.cf_engine`  
 module, 166

`evalml.automl.engine.dask_engine`  
 module, 168

`evalml.automl.engine.engine_base`  
 module, 171

`evalml.automl.engine.sequential_engine`  
 module, 174

`evalml.automl.pipeline_search_plots`  
 module, 187

`evalml.automl.utils`  
 module, 188

`evalml.data_checks`  
 module, 198

`evalml.data_checks.class_imbalance_data_check`  
 module, 198

`evalml.data_checks.data_check`  
 module, 200

`evalml.data_checks.data_check_action`  
 module, 201

`evalml.data_checks.data_check_action_code`  
 module, 202

`evalml.data_checks.data_check_message`  
 module, 202

`evalml.data_checks.data_check_message_code`  
 module, 204

`evalml.data_checks.data_check_message_type`  
 module, 206

`evalml.data_checks.data_checks`  
 module, 206

`evalml.data_checks.datetime_format_data_check`  
 module, 207

`evalml.data_checks.datetime_nan_data_check`  
 module, 208

`evalml.data_checks.default_data_checks`  
 module, 210

`evalml.data_checks.highly_null_data_check`  
 module, 211

`evalml.data_checks.id_columns_data_check`  
 module, 213

`evalml.data_checks.invalid_targets_data_check`  
 module, 214

`evalml.data_checks.multicollinearity_data_check`  
 module, 215

`evalml.data_checks.natural_language_nan_data_check`  
 module, 216

`evalml.data_checks.no_variance_data_check`  
 module, 217

`evalml.data_checks.outliers_data_check`  
 module, 218

`evalml.data_checks.sparsity_data_check`  
 module, 219

`evalml.data_checks.target_distribution_data_check`  
 module, 221

`evalml.data_checks.target_leakage_data_check`  
 module, 222

`evalml.data_checks.uniqueness_data_check`  
 module, 223

`evalml.data_checks.utils`  
 module, 225

`evalml.demos`  
 module, 246

`evalml.demos.breast_cancer`  
 module, 246

`evalml.demos.churn`  
 module, 246

`evalml.demos.diabetes`  
 module, 247

`evalml.demos.fraud`  
 module, 247

`evalml.demos.wine`  
 module, 248

`evalml.exceptions`  
 module, 249

`evalml.exceptions.exceptions`  
 module, 249

`evalml.model_family`  
 module, 254

`evalml.model_family.model_family`  
 module, 254

`evalml.model_family.utils`  
 module, 255

`evalml.model_understanding`  
 module, 256

`evalml.model_understanding.force_plots`  
 module, 263



<code>evalml.model_understanding.graphs</code>	<code>evalml.pipelines.components.ensemble.sklearn_stack</code>
module, 264	module, 460
<code>evalml.model_understanding.permutation_importance</code>	<code>evalml.pipelines.components.estimators</code>
module, 274	module, 471
<code>evalml.model_understanding.prediction_explanations</code>	<code>evalml.pipelines.components.estimators.classifiers</code>
module, 256	module, 471
<code>evalml.model_understanding.prediction_explanations</code>	<code>evalml.pipelines.components.estimators.classifiers</code>
module, 256	module, 471
<code>evalml.objectives</code>	<code>evalml.pipelines.components.estimators.classifiers</code>
module, 287	module, 474
<code>evalml.objectives.binary_classification_objective</code>	<code>evalml.pipelines.components.estimators.classifiers</code>
module, 287	module, 477
<code>evalml.objectives.cost_benefit_matrix</code>	<code>evalml.pipelines.components.estimators.classifiers</code>
module, 290	module, 480
<code>evalml.objectives.fraud_cost</code>	<code>evalml.pipelines.components.estimators.classifiers</code>
module, 292	module, 483
<code>evalml.objectives.lead_scoring</code>	<code>evalml.pipelines.components.estimators.classifiers</code>
module, 295	module, 486
<code>evalml.objectives.multiclass_classification_objective</code>	<code>evalml.pipelines.components.estimators.classifiers</code>
module, 298	module, 489
<code>evalml.objectives.objective_base</code>	<code>evalml.pipelines.components.estimators.classifiers</code>
module, 300	module, 493
<code>evalml.objectives.regression_objective</code>	<code>evalml.pipelines.components.estimators.classifiers</code>
module, 302	module, 496
<code>evalml.objectives.sensitivity_low_alert</code>	<code>evalml.pipelines.components.estimators.classifiers</code>
module, 305	module, 498
<code>evalml.objectives.standard_metrics</code>	<code>evalml.pipelines.components.estimators.classifiers</code>
module, 307	module, 501
<code>evalml.objectives.time_series_regression_objective</code>	<code>evalml.pipelines.components.estimators.estimator</code>
module, 369	module, 608
<code>evalml.objectives.utils</code>	<code>evalml.pipelines.components.estimators.regressors</code>
module, 371	module, 535
<code>evalml.pipelines</code>	<code>evalml.pipelines.components.estimators.regressors.a</code>
module, 453	module, 535
<code>evalml.pipelines.binary_classification_pipeline</code>	<code>evalml.pipelines.components.estimators.regressors.b</code>
module, 1056	module, 539
<code>evalml.pipelines.binary_classification_pipeline</code>	<code>evalml.pipelines.components.estimators.regressors.c</code>
module, 1061	module, 541
<code>evalml.pipelines.classification_pipeline</code>	<code>evalml.pipelines.components.estimators.regressors.c</code>
module, 1062	module, 544
<code>evalml.pipelines.component_graph</code>	<code>evalml.pipelines.components.estimators.regressors.e</code>
module, 1066	module, 548
<code>evalml.pipelines.components</code>	<code>evalml.pipelines.components.estimators.regressors.e</code>
module, 453	module, 550
<code>evalml.pipelines.components.component_base</code>	<code>evalml.pipelines.components.estimators.regressors.f</code>
module, 907	module, 554
<code>evalml.pipelines.components.component_base</code>	<code>evalml.pipelines.components.estimators.regressors.f</code>
module, 909	module, 557
<code>evalml.pipelines.components.ensemble</code>	<code>evalml.pipelines.components.estimators.regressors.p</code>
module, 453	module, 560
<code>evalml.pipelines.components.ensemble.sklearn_stacked_ensemble_base</code>	<code>evalml.pipelines.components.estimators.regressors.r</code>
module, 454	module, 563
<code>evalml.pipelines.components.ensemble.sklearn_stacked_ensemble_classifier</code>	<code>evalml.pipelines.components.estimators.regressors.s</code>
module, 457	module, 565

[evalml.pipelines.components.estimators.regressor\\_pipeline\\_components.transformers.preprocessors](#)  
 module, 568

[evalml.pipelines.components.estimators.regressor\\_pipeline\\_components.transformers.preprocessors](#)  
 module, 571

[evalml.pipelines.components.transformer\\_pipeline\\_components.transformers.preprocessors](#)  
 module, 677

[evalml.pipelines.components.transformer\\_pipeline\\_components.transformers.samplers](#)  
 module, 823

[evalml.pipelines.components.transformer\\_pipeline\\_components.transformers.samplers.k](#)  
 module, 677

[evalml.pipelines.components.transformer\\_pipeline\\_components.transformers.samplers.k](#)  
 module, 677

[evalml.pipelines.components.transformer\\_pipeline\\_components.transformers.samplers.k](#)  
 module, 680

[evalml.pipelines.components.transformer\\_pipeline\\_components.transformers.scalers](#)  
 module, 687

[evalml.pipelines.components.transformer\\_pipeline\\_components.transformers.scalers.st](#)  
 module, 687

[evalml.pipelines.components.transformer\\_pipeline\\_components.transformers.transforme](#)  
 module, 691

[evalml.pipelines.components.transformer\\_pipeline\\_components.utils](#)  
 module, 700

[evalml.pipelines.components.transformer\\_pipeline\\_components.feature\\_selection\\_pipeline](#)  
 module, 700

[evalml.pipelines.components.transformer\\_pipeline\\_components.feature\\_selector](#)  
 module, 703

[evalml.pipelines.components.transformer\\_pipeline\\_components.feature\\_selector](#)  
 module, 706

[evalml.pipelines.components.transformer\\_pipeline\\_components.regression\\_pipeline](#)  
 module, 718

[evalml.pipelines.components.transformer\\_pipeline\\_components.time\\_series\\_classification\\_pipeline](#)  
 module, 718

[evalml.pipelines.components.transformer\\_pipeline\\_components.time\\_series\\_regression\\_pipeline](#)  
 module, 721

[evalml.pipelines.components.transformer\\_pipeline\\_components.imputer](#)  
 module, 724

[evalml.pipelines.components.transformer\\_pipeline\\_components.preprocessing](#)  
 module, 726

[evalml.pipelines.components.transformer\\_pipeline\\_components.preprocessing.data\\_splitters](#)  
 module, 740

[evalml.pipelines.components.transformer\\_pipeline\\_components.preprocessing.data\\_splitters.balanced\\_class](#)  
 module, 740

[evalml.pipelines.components.transformer\\_pipeline\\_components.preprocessing.data\\_splitters.balanced\\_class](#)  
 module, 743

[evalml.pipelines.components.transformer\\_pipeline\\_components.preprocessing.data\\_splitters.time\\_series\\_sp](#)  
 module, 745

[evalml.pipelines.components.transformer\\_pipeline\\_components.preprocessing.data\\_splitters.training\\_valida](#)  
 module, 748

[evalml.pipelines.components.transformer\\_pipeline\\_components.preprocessing.data\\_splitters.training\\_valida](#)  
 module, 750

[evalml.pipelines.components.transformer\\_pipeline\\_components.preprocessing.data\\_splitters.training\\_valida](#)  
 module, 753

[evalml.pipelines.components.transformer\\_pipeline\\_components.preprocessing.data\\_splitters.training\\_valida](#)  
 module, 755

[evalml.problem\\_types.utils](#)  
     module, [1236](#)  
[evalml.tuners](#)  
     module, [1240](#)  
[evalml.tuners.grid\\_search\\_tuner](#)  
     module, [1240](#)  
[evalml.tuners.random\\_search\\_tuner](#)  
     module, [1242](#)  
[evalml.tuners.skopt\\_tuner](#)  
     module, [1243](#)  
[evalml.tuners.tuner](#)  
     module, [1245](#)  
[evalml.tuners.tuner\\_exceptions](#)  
     module, [1246](#)  
[evalml.utils](#)  
     module, [1250](#)  
[evalml.utils.base\\_meta](#)  
     module, [1250](#)  
[evalml.utils.cli\\_utils](#)  
     module, [1251](#)  
[evalml.utils.gen\\_utils](#)  
     module, [1252](#)  
[evalml.utils.logger](#)  
     module, [1256](#)  
[evalml.utils.update\\_checker](#)  
     module, [1257](#)  
[evalml.utils.woodwork\\_utils](#)  
     module, [1257](#)  
[EvalMLAlgorithm](#) (class in [evalml.automl.automl\\_algorithm](#)), [163](#)  
[EvalMLAlgorithm](#) (class in [evalml.automl.automl\\_algorithm.evalml\\_algorithm](#)), [158](#)  
[evaluate\\_pipeline\(\)](#) (in module [evalml.automl.engine](#)), [178](#)  
[evaluate\\_pipeline\(\)](#) (in module [evalml.automl.engine.engine\\_base](#)), [172](#)  
[explain\\_predictions\(\)](#) (in module [evalml.model\\_understanding](#)), [278](#)  
[explain\\_predictions\(\)](#) (in module [evalml.model\\_understanding.prediction\\_explanations](#)), [261](#)  
[explain\\_predictions\(\)](#) (in module [evalml.model\\_understanding.prediction\\_explanations.explainers](#)), [257](#)  
[explain\\_predictions\\_best\\_worst\(\)](#) (in module [evalml.model\\_understanding](#)), [279](#)  
[explain\\_predictions\\_best\\_worst\(\)](#) (in module [evalml.model\\_understanding.prediction\\_explanations](#)), [261](#)  
[explain\\_predictions\\_best\\_worst\(\)](#) (in module [evalml.model\\_understanding.prediction\\_explanations.explainers](#)), [258](#)  
[ExplainPredictionsStage](#) (class in [evalml.model\\_understanding.prediction\\_explanations.explainers](#)), [259](#)  
[ExpVariance](#) (class in [evalml.objectives](#)), [393](#)  
[ExpVariance](#) (class in [evalml.objectives.standard\\_metrics](#)), [323](#)  
[ExtraTreesClassifier](#) (class in [evalml.pipelines](#)), [1133](#)  
[ExtraTreesClassifier](#) (class in [evalml.pipelines.components](#)), [958](#)  
[ExtraTreesClassifier](#) (class in [evalml.pipelines.components.estimators](#)), [638](#)  
[ExtraTreesClassifier](#) (class in [evalml.pipelines.components.estimators.classifiers](#)), [515](#)  
[ExtraTreesClassifier](#) (class in [evalml.pipelines.components.estimators.classifiers.et\\_classifier](#)), [483](#)  
[ExtraTreesRegressor](#) (class in [evalml.pipelines](#)), [1136](#)  
[ExtraTreesRegressor](#) (class in [evalml.pipelines.components](#)), [961](#)  
[ExtraTreesRegressor](#) (class in [evalml.pipelines.components.estimators](#)), [641](#)  
[ExtraTreesRegressor](#) (class in [evalml.pipelines.components.estimators.regressors](#)), [587](#)  
[ExtraTreesRegressor](#) (class in [evalml.pipelines.components.estimators.regressors.et\\_regressor](#)), [551](#)  
[F1](#) (class in [evalml.objectives](#)), [395](#)  
[F1](#) (class in [evalml.objectives.standard\\_metrics](#)), [324](#)  
[F1Macro](#) (class in [evalml.objectives](#)), [397](#)  
[F1Macro](#) (class in [evalml.objectives.standard\\_metrics](#)), [327](#)  
[F1Micro](#) (class in [evalml.objectives](#)), [399](#)  
[F1Micro](#) (class in [evalml.objectives.standard\\_metrics](#)), [328](#)  
[F1Weighted](#) (class in [evalml.objectives](#)), [400](#)  
[F1Weighted](#) (class in [evalml.objectives.standard\\_metrics](#)), [330](#)  
[feature\\_importance\(\)](#) ([evalml.pipelines.ARIMARegressor](#) property), [1107](#)  
[feature\\_importance\(\)](#) ([evalml.pipelines.binary\\_classification\\_pipeline.BinaryClassifier](#) property), [1058](#)  
[feature\\_importance\(\)](#) ([evalml.pipelines.CatBoostClassifier](#) property), [1109](#)  
[feature\\_importance\(\)](#)

<code>(evalml.pipelines.CatBoostRegressor</code> <code>property)</code> , 1112	<code>(evalml.pipelines.components.estimators.ARIMARegressor</code> <code>property)</code> , 614
<code>feature_importance()</code> <code>(evalml.pipelines.classification_pipeline.ClassificationPipeline</code> <code>property)</code> , 1064	<code>feature_importance()</code> <code>(evalml.pipelines.components.estimators.BaselineClassifier</code> <code>property)</code> , 616
<code>feature_importance()</code> <code>(evalml.pipelines.components.ARIMARegressor</code> <code>property)</code> , 918	<code>feature_importance()</code> <code>(evalml.pipelines.components.estimators.BaselineRegressor</code> <code>property)</code> , 618
<code>feature_importance()</code> <code>(evalml.pipelines.components.BaselineClassifier</code> <code>property)</code> , 920	<code>feature_importance()</code> <code>(evalml.pipelines.components.estimators.CatBoostClassifier</code> <code>property)</code> , 621
<code>feature_importance()</code> <code>(evalml.pipelines.components.BaselineRegressor</code> <code>property)</code> , 923	<code>feature_importance()</code> <code>(evalml.pipelines.components.estimators.CatBoostRegressor</code> <code>property)</code> , 624
<code>feature_importance()</code> <code>(evalml.pipelines.components.CatBoostClassifier</code> <code>property)</code> , 925	<code>feature_importance()</code> <code>(evalml.pipelines.components.estimators.classifiers.baseline_class</code> <code>property)</code> , 473
<code>feature_importance()</code> <code>(evalml.pipelines.components.CatBoostRegressor</code> <code>property)</code> , 928	<code>feature_importance()</code> <code>(evalml.pipelines.components.estimators.classifiers.BaselineClassifier</code> <code>property)</code> , 506
<code>feature_importance()</code> <code>(evalml.pipelines.components.DecisionTreeClassifier</code> <code>property)</code> , 936	<code>feature_importance()</code> <code>(evalml.pipelines.components.estimators.classifiers.catboost_classifier</code> <code>property)</code> , 476
<code>feature_importance()</code> <code>(evalml.pipelines.components.DecisionTreeRegressor</code> <code>property)</code> , 939	<code>feature_importance()</code> <code>(evalml.pipelines.components.estimators.classifiers.CatBoostClassifier</code> <code>property)</code> , 509
<code>feature_importance()</code> <code>(evalml.pipelines.components.ElasticNetClassifier</code> <code>property)</code> , 950	<code>feature_importance()</code> <code>(evalml.pipelines.components.estimators.classifiers.decision_tree_classifier</code> <code>property)</code> , 479
<code>feature_importance()</code> <code>(evalml.pipelines.components.ElasticNetRegressor</code> <code>property)</code> , 953	<code>feature_importance()</code> <code>(evalml.pipelines.components.estimators.classifiers.DecisionTreeRegressor</code> <code>property)</code> , 512
<code>feature_importance()</code> <code>(evalml.pipelines.components.ensemble.sklearn_stacked_ensemble_classifier</code> <code>property)</code> , 455	<code>feature_importance()</code> <code>(evalml.pipelines.components.estimators.classifiers.elasticnet_classifier</code> <code>property)</code> , 482
<code>feature_importance()</code> <code>(evalml.pipelines.components.ensemble.sklearn_stacked_ensemble_classifier</code> <code>property)</code> , 459	<code>feature_importance()</code> <code>(evalml.pipelines.components.estimators.classifiers.sklearn_stacked_ensemble_classifier</code> <code>property)</code> , 515
<code>feature_importance()</code> <code>(evalml.pipelines.components.ensemble.sklearn_stacked_ensemble_regressor</code> <code>property)</code> , 462	<code>feature_importance()</code> <code>(evalml.pipelines.components.estimators.classifiers.sklearn_stacked_ensemble_regressor</code> <code>property)</code> , 485
<code>feature_importance()</code> <code>(evalml.pipelines.components.ensemble.SklearnStackedEnsembleClassifier</code> <code>property)</code> , 464	<code>feature_importance()</code> <code>(evalml.pipelines.components.estimators.classifiers.ExtraTreesClassifier</code> <code>property)</code> , 518
<code>feature_importance()</code> <code>(evalml.pipelines.components.ensemble.SklearnStackedEnsembleClassifier</code> <code>property)</code> , 467	<code>feature_importance()</code> <code>(evalml.pipelines.components.estimators.classifiers.kneighbors_classifier</code> <code>property)</code> , 488
<code>feature_importance()</code> <code>(evalml.pipelines.components.ensemble.SklearnStackedEnsembleRegressor</code> <code>property)</code> , 470	<code>feature_importance()</code> <code>(evalml.pipelines.components.estimators.classifiers.KNeighborsClassifier</code> <code>property)</code> , 521
<code>feature_importance()</code> <code>(evalml.pipelines.components.Estimator</code> <code>property)</code> , 957	<code>feature_importance()</code> <code>(evalml.pipelines.components.estimators.classifiers.lightgbm_classifier</code> <code>property)</code> , 492
<code>feature_importance()</code>	<code>feature_importance()</code>

<code>(evalml.pipelines.components.estimators.classifiers.LightGBMClassifier</code> <code>property), 524</code>	<code>(evalml.pipelines.components.estimators.classifiers.LightGBMClassifier</code> <code>property), 649</code>
<code>feature_importance()</code> <code>(evalml.pipelines.components.estimators.classifiers.logistic_regression_pipeline_classification.LogisticRegressionClassifier</code> <code>property), 495</code>	<code>feature_importance()</code> <code>(evalml.pipelines.components.estimators.classifiers.logistic_regression_pipeline_classification.LogisticRegressionClassifier</code> <code>property), 652</code>
<code>feature_importance()</code> <code>(evalml.pipelines.components.estimators.classifiers.LogisticRegressionClassifier</code> <code>property), 527</code>	<code>feature_importance()</code> <code>(evalml.pipelines.components.estimators.LinearRegressor</code> <code>property), 654</code>
<code>feature_importance()</code> <code>(evalml.pipelines.components.estimators.classifiers.RandomForestClassifier</code> <code>property), 529</code>	<code>feature_importance()</code> <code>(evalml.pipelines.components.estimators.LogisticRegressionClassifier</code> <code>property), 657</code>
<code>feature_importance()</code> <code>(evalml.pipelines.components.estimators.classifiers.rf_classifier.RandomForestClassifier</code> <code>property), 497</code>	<code>feature_importance()</code> <code>(evalml.pipelines.components.estimators.ProphetRegressor</code> <code>property), 659</code>
<code>feature_importance()</code> <code>(evalml.pipelines.components.estimators.classifiers.svm_classifier.SVMClassifier</code> <code>property), 500</code>	<code>feature_importance()</code> <code>(evalml.pipelines.components.estimators.RandomForestClassifier</code> <code>property), 662</code>
<code>feature_importance()</code> <code>(evalml.pipelines.components.estimators.classifiers.SVMClassifier</code> <code>property), 532</code>	<code>feature_importance()</code> <code>(evalml.pipelines.components.estimators.RandomForestRegressor</code> <code>property), 664</code>
<code>feature_importance()</code> <code>(evalml.pipelines.components.estimators.classifiers.xgboost_classifier.XGBoostClassifier</code> <code>property), 503</code>	<code>feature_importance()</code> <code>(evalml.pipelines.components.estimators.regressors.arima_regressor.ArimaRegressor</code> <code>property), 538</code>
<code>feature_importance()</code> <code>(evalml.pipelines.components.estimators.classifiers.XGBoostClassifier</code> <code>property), 534</code>	<code>feature_importance()</code> <code>(evalml.pipelines.components.estimators.regressors.ARIMARegressor</code> <code>property), 576</code>
<code>feature_importance()</code> <code>(evalml.pipelines.components.estimators.DecisionTreeClassifier</code> <code>property), 627</code>	<code>feature_importance()</code> <code>(evalml.pipelines.components.estimators.regressors.baseline_regressor.BaselineRegressor</code> <code>property), 540</code>
<code>feature_importance()</code> <code>(evalml.pipelines.components.estimators.DecisionTreeRegressor</code> <code>property), 630</code>	<code>feature_importance()</code> <code>(evalml.pipelines.components.estimators.regressors.BaselineRegressor</code> <code>property), 578</code>
<code>feature_importance()</code> <code>(evalml.pipelines.components.estimators.ElasticNetClassifier</code> <code>property), 633</code>	<code>feature_importance()</code> <code>(evalml.pipelines.components.estimators.regressors.catboost_regressor.CatBoostRegressor</code> <code>property), 543</code>
<code>feature_importance()</code> <code>(evalml.pipelines.components.estimators.ElasticNetRegressor</code> <code>property), 635</code>	<code>feature_importance()</code> <code>(evalml.pipelines.components.estimators.regressors.CatBoostRegressor</code> <code>property), 581</code>
<code>feature_importance()</code> <code>(evalml.pipelines.components.estimators.Estimator</code> <code>property), 637</code>	<code>feature_importance()</code> <code>(evalml.pipelines.components.estimators.regressors.decision_tree_regressor.DecisionTreeRegressor</code> <code>property), 547</code>
<code>feature_importance()</code> <code>(evalml.pipelines.components.estimators.estimator.Estimator</code> <code>property), 609</code>	<code>feature_importance()</code> <code>(evalml.pipelines.components.estimators.regressors.DecisionTreeRegressor</code> <code>property), 584</code>
<code>feature_importance()</code> <code>(evalml.pipelines.components.estimators.ExtraTreesClassifier</code> <code>property), 640</code>	<code>feature_importance()</code> <code>(evalml.pipelines.components.estimators.regressors.elasticnet_regressor.ElasticNetRegressor</code> <code>property), 549</code>
<code>feature_importance()</code> <code>(evalml.pipelines.components.estimators.ExtraTreesRegressor</code> <code>property), 643</code>	<code>feature_importance()</code> <code>(evalml.pipelines.components.estimators.regressors.ElasticNetRegressor</code> <code>property), 586</code>
<code>feature_importance()</code> <code>(evalml.pipelines.components.estimators.KNeighborsClassifier</code> <code>property), 646</code>	<code>feature_importance()</code> <code>(evalml.pipelines.components.estimators.regressors.et_regressor.ExtraTreesRegressor</code> <code>property), 553</code>
<code>feature_importance()</code>	<code>feature_importance()</code>

```

(evalml.pipelines.components.estimators.regressors.ExtraTreesRegressor
property), 589
feature_importance()
(evalml.pipelines.components.estimators.regressors.lightgbm.LightGBMRegressor
property), 556
feature_importance()
(evalml.pipelines.components.estimators.regressors.LightGBMRegressor
property), 592
feature_importance()
(evalml.pipelines.components.estimators.regressors.linear_regression.LinearRegression
property), 559
feature_importance()
(evalml.pipelines.components.estimators.regressors.LinearRegression
property), 595
feature_importance()
(evalml.pipelines.components.estimators.regressors.prophet.ProphetRegressor
property), 562
feature_importance()
(evalml.pipelines.components.estimators.regressors.ProphetRegressor
property), 597
feature_importance()
(evalml.pipelines.components.estimators.regressors.RandomForestRegressor
property), 600
feature_importance()
(evalml.pipelines.components.estimators.regressors.rf_regression.RandomForestRegressor
property), 564
feature_importance()
(evalml.pipelines.components.estimators.regressors.svm_regression.SVMRegressor
property), 567
feature_importance()
(evalml.pipelines.components.estimators.regressors.SVMRegressor
property), 602
feature_importance()
(evalml.pipelines.components.estimators.regressors.time_series.TimeSeriesBaselineEstimator
property), 570
feature_importance()
(evalml.pipelines.components.estimators.regressors.TimeSeriesBaselineEstimator
property), 604
feature_importance()
(evalml.pipelines.components.estimators.regressors.xgboost.XGBRegressor
property), 573
feature_importance()
(evalml.pipelines.components.estimators.regressors.XGBRegressor
property), 607
feature_importance()
(evalml.pipelines.components.estimators.SVMClassifier
property), 666
feature_importance()
(evalml.pipelines.components.estimators.SVMRegressor
property), 669
feature_importance()
(evalml.pipelines.components.estimators.TimeSeriesBaselineEstimator
property), 671
feature_importance()

(evalml.pipelines.components.estimators.XGBoostClassifier
property), 674
feature_importance()
(evalml.pipelines.components.estimators.XGBoostRegressor
property), 676
feature_importance()
(evalml.pipelines.components.ExtraTreesClassifier
property), 960
feature_importance()
(evalml.pipelines.components.ExtraTreesRegressor
property), 963
feature_importance()
(evalml.pipelines.components.KNeighborsClassifier
property), 970
feature_importance()
(evalml.pipelines.components.LightGBMClassifier
property), 973
feature_importance()
(evalml.pipelines.components.LightGBMRegressor
property), 976
feature_importance()
(evalml.pipelines.components.LinearRegressor
property), 980
feature_importance()
(evalml.pipelines.components.LogisticRegressionClassifier
property), 983
feature_importance()
(evalml.pipelines.components.ProphetRegressor
property), 999
feature_importance()
(evalml.pipelines.components.RandomForestClassifier
property), 1002
feature_importance()
(evalml.pipelines.components.SklearnStackedEnsembleClassifier
property), 1018
feature_importance()
(evalml.pipelines.components.SklearnStackedEnsembleRegressor
property), 1021
feature_importance()
(evalml.pipelines.components.SVMClassifier
property), 1032
feature_importance()
(evalml.pipelines.components.SVMRegressor
property), 1034
feature_importance()
(evalml.pipelines.components.TimeSeriesBaselineEstimator
property), 1043
feature_importance()
(evalml.pipelines.components.XGBoostClassifier
property), 1053
feature_importance()

```



[\(evalml.pipelines.components.XGBoostRegressor property\), 1055](#)  
[feature\\_importance\(\) \(evalml.pipelines.DecisionTreeClassifier property\), 1117](#)  
[feature\\_importance\(\) \(evalml.pipelines.DecisionTreeRegressor property\), 1120](#)  
[feature\\_importance\(\) \(evalml.pipelines.ElasticNetClassifier property\), 1127](#)  
[feature\\_importance\(\) \(evalml.pipelines.ElasticNetRegressor property\), 1130](#)  
[feature\\_importance\(\) \(evalml.pipelines.Estimator property\), 1132](#)  
[feature\\_importance\(\) \(evalml.pipelines.ExtraTreesClassifier property\), 1135](#)  
[feature\\_importance\(\) \(evalml.pipelines.ExtraTreesRegressor property\), 1138](#)  
[feature\\_importance\(\) \(evalml.pipelines.KNeighborsClassifier property\), 1143](#)  
[feature\\_importance\(\) \(evalml.pipelines.LightGBMClassifier property\), 1145](#)  
[feature\\_importance\(\) \(evalml.pipelines.LightGBMRegressor property\), 1148](#)  
[feature\\_importance\(\) \(evalml.pipelines.LinearRegressor property\), 1150](#)  
[feature\\_importance\(\) \(evalml.pipelines.LogisticRegressionClassifier property\), 1153](#)  
[feature\\_importance\(\) \(evalml.pipelines.multiclass\\_classification\\_pipeline.MulticlassClassificationPipeline property\), 1072](#)  
[feature\\_importance\(\) \(evalml.pipelines.MulticlassClassificationPipeline property\), 1156](#)  
[feature\\_importance\(\) \(evalml.pipelines.pipeline\\_base.PipelineBase property\), 1076](#)  
[feature\\_importance\(\) \(evalml.pipelines.PipelineBase property\), 1165](#)  
[feature\\_importance\(\) \(evalml.pipelines.ProphetRegressor property\), 1168](#)  
[feature\\_importance\(\) \(evalml.pipelines.RandomForestClassifier property\), 1171](#)  
[feature\\_importance\(\) \(evalml.pipelines.RandomForestRegressor property\), 1173](#)  
[feature\\_importance\(\) \(evalml.pipelines.regression\\_pipeline.RegressionPipeline property\), 1082](#)  
[feature\\_importance\(\) \(evalml.pipelines.RegressionPipeline property\), 1175](#)  
[feature\\_importance\(\) \(evalml.pipelines.SklearnStackedEnsembleClassifier property\), 1187](#)  
[feature\\_importance\(\) \(evalml.pipelines.SklearnStackedEnsembleRegressor property\), 1189](#)  
[feature\\_importance\(\) \(evalml.pipelines.SVMClassifier property\), 1194](#)  
[feature\\_importance\(\) \(evalml.pipelines.SVMRegressor property\), 1196](#)  
[feature\\_importance\(\) \(evalml.pipelines.time\\_series\\_classification\\_pipelines.TimeSeriesClassificationPipeline property\), 1087](#)  
[feature\\_importance\(\) \(evalml.pipelines.time\\_series\\_classification\\_pipelines.TimeSeriesClassificationPipeline property\), 1091](#)  
[feature\\_importance\(\) \(evalml.pipelines.time\\_series\\_classification\\_pipelines.TimeSeriesClassificationPipeline property\), 1096](#)  
[feature\\_importance\(\) \(evalml.pipelines.time\\_series\\_regression\\_pipeline.TimeSeriesRegressionPipeline property\), 1100](#)  
[feature\\_importance\(\) \(evalml.pipelines.TimeSeriesBinaryClassificationPipeline property\), 1201](#)  
[feature\\_importance\(\) \(evalml.pipelines.TimeSeriesClassificationPipeline property\), 1206](#)  
[feature\\_importance\(\) \(evalml.pipelines.TimeSeriesMulticlassClassificationPipeline property\), 1210](#)  
[feature\\_importance\(\) \(evalml.pipelines.TimeSeriesRegressionPipeline property\), 1214](#)  
[feature\\_importance\(\) \(evalml.pipelines.XGBoostClassifier property\), 1220](#)  
[feature\\_importance\(\) \(evalml.pipelines.XGBoostRegressor property\), 1222](#)  
[FeatureSelector \(class in evalml.pipelines\), 1138](#)  
[FeatureSelector \(class in](#)

<code>evalml.pipelines.components</code> ), 963	<code>fit()</code> ( <code>evalml.pipelines.components.ElasticNetRegressor</code>
<code>FeatureSelector</code> (class in <code>evalml.pipelines.components.transformers</code> ), 850	<code>method</code> ), 953
<code>FeatureSelector</code> (class in <code>evalml.pipelines.components.transformers.feature_selection</code> ), 710	<code>fit()</code> ( <code>evalml.pipelines.components.EmailFeaturizer</code> <code>method</code> ), 955
<code>FeatureSelector</code> (class in <code>evalml.pipelines.components.transformers.feature_selection</code> ), 701	<code>fit()</code> ( <code>evalml.pipelines.components.ensemble.sklearn_stacked_ensemble</code> <code>method</code> ), 455
<code>FeatureSelector</code> (class in <code>evalml.pipelines.components.transformers.feature_selection</code> ), 701	<code>fit()</code> ( <code>evalml.pipelines.components.ensemble.sklearn_stacked_ensemble</code> <code>method</code> ), 459
<code>fit()</code> ( <code>evalml.pipelines.ARIMARegressor</code> <code>method</code> ), 1107	<code>fit()</code> ( <code>evalml.pipelines.components.ensemble.sklearn_stacked_ensemble</code> <code>method</code> ), 462
<code>fit()</code> ( <code>evalml.pipelines.binary_classification_pipeline.BinaryClassificationPipeline</code> <code>method</code> ), 1058	<code>fit()</code> ( <code>evalml.pipelines.components.ensemble.SklearnStackedEnsembleBa</code> <code>method</code> ), 464
<code>fit()</code> ( <code>evalml.pipelines.CatBoostClassifier</code> <code>method</code> ), 1109	<code>fit()</code> ( <code>evalml.pipelines.components.ensemble.SklearnStackedEnsembleCl</code> <code>method</code> ), 467
<code>fit()</code> ( <code>evalml.pipelines.CatBoostRegressor</code> <code>method</code> ), 1112	<code>fit()</code> ( <code>evalml.pipelines.components.ensemble.SklearnStackedEnsembleRe</code> <code>method</code> ), 470
<code>fit()</code> ( <code>evalml.pipelines.classification_pipeline.ClassificationPipeline</code> <code>method</code> ), 1064	<code>fit()</code> ( <code>evalml.pipelines.components.Estimator</code> <code>method</code> ), 957
<code>fit()</code> ( <code>evalml.pipelines.component_graph.ComponentGraph</code> <code>method</code> ), 1068	<code>fit()</code> ( <code>evalml.pipelines.components.estimators.ARIMARegressor</code> <code>method</code> ), 614
<code>fit()</code> ( <code>evalml.pipelines.ComponentGraph</code> <code>method</code> ), 1114	<code>fit()</code> ( <code>evalml.pipelines.components.estimators.BaselineClassifier</code> <code>method</code> ), 616
<code>fit()</code> ( <code>evalml.pipelines.components.ARIMARegressor</code> <code>method</code> ), 918	<code>fit()</code> ( <code>evalml.pipelines.components.estimators.BaselineRegressor</code> <code>method</code> ), 618
<code>fit()</code> ( <code>evalml.pipelines.components.BaselineClassifier</code> <code>method</code> ), 920	<code>fit()</code> ( <code>evalml.pipelines.components.estimators.CatBoostClassifier</code> <code>method</code> ), 621
<code>fit()</code> ( <code>evalml.pipelines.components.BaselineRegressor</code> <code>method</code> ), 923	<code>fit()</code> ( <code>evalml.pipelines.components.estimators.CatBoostRegressor</code> <code>method</code> ), 624
<code>fit()</code> ( <code>evalml.pipelines.components.CatBoostClassifier</code> <code>method</code> ), 925	<code>fit()</code> ( <code>evalml.pipelines.components.estimators.classifiers.baseline_classi</code> <code>method</code> ), 473
<code>fit()</code> ( <code>evalml.pipelines.components.CatBoostRegressor</code> <code>method</code> ), 928	<code>fit()</code> ( <code>evalml.pipelines.components.estimators.classifiers.BaselineClassif</code> <code>method</code> ), 506
<code>fit()</code> ( <code>evalml.pipelines.components.component_base.ComponentBase</code> <code>method</code> ), 908	<code>fit()</code> ( <code>evalml.pipelines.components.estimators.classifiers.catboost_classi</code> <code>method</code> ), 476
<code>fit()</code> ( <code>evalml.pipelines.components.ComponentBase</code> <code>method</code> ), 930	<code>fit()</code> ( <code>evalml.pipelines.components.estimators.classifiers.CatBoostClassi</code> <code>method</code> ), 509
<code>fit()</code> ( <code>evalml.pipelines.components.DateTimeFeaturizer</code> <code>method</code> ), 932	<code>fit()</code> ( <code>evalml.pipelines.components.estimators.classifiers.decision_tree_c</code> <code>method</code> ), 479
<code>fit()</code> ( <code>evalml.pipelines.components.DecisionTreeClassifier</code> <code>method</code> ), 936	<code>fit()</code> ( <code>evalml.pipelines.components.estimators.classifiers.DecisionTreeCl</code> <code>method</code> ), 512
<code>fit()</code> ( <code>evalml.pipelines.components.DecisionTreeRegressor</code> <code>method</code> ), 939	<code>fit()</code> ( <code>evalml.pipelines.components.estimators.classifiers.elasticnet_class</code> <code>method</code> ), 482
<code>fit()</code> ( <code>evalml.pipelines.components.DelayedFeatureTransformer</code> <code>method</code> ), 941	<code>fit()</code> ( <code>evalml.pipelines.components.estimators.classifiers.ElasticNetClass</code> <code>method</code> ), 515
<code>fit()</code> ( <code>evalml.pipelines.components.DFSTransformer</code> <code>method</code> ), 943	<code>fit()</code> ( <code>evalml.pipelines.components.estimators.classifiers.et_classifier.Ext</code> <code>method</code> ), 485
<code>fit()</code> ( <code>evalml.pipelines.components.DropColumns</code> <code>method</code> ), 945	<code>fit()</code> ( <code>evalml.pipelines.components.estimators.classifiers.ExtraTreesClas</code> <code>method</code> ), 518
<code>fit()</code> ( <code>evalml.pipelines.components.DropNullColumns</code> <code>method</code> ), 947	<code>fit()</code> ( <code>evalml.pipelines.components.estimators.classifiers.kneighbors_cla</code> <code>method</code> ), 488
<code>fit()</code> ( <code>evalml.pipelines.components.ElasticNetClassifier</code> <code>method</code> ), 950	<code>fit()</code> ( <code>evalml.pipelines.components.estimators.classifiers.KNeighborsCla</code> <code>method</code> ), 521
	<code>fit()</code> ( <code>evalml.pipelines.components.estimators.classifiers.lightgbm_classi</code> <code>method</code> ), 492



`fit()` (`evalml.pipelines.components.estimators.classifiers.EightClassClassifier`), 524

`fit()` (`evalml.pipelines.components.estimators.classifiers.EightClassClassifier`), 540

`fit()` (`evalml.pipelines.components.estimators.classifiers.LogisticRegressionClassifier`), 495

`fit()` (`evalml.pipelines.components.estimators.classifiers.LogisticRegressionClassifier`), 578

`fit()` (`evalml.pipelines.components.estimators.classifiers.LogisticRegressionClassifier`), 527

`fit()` (`evalml.pipelines.components.estimators.classifiers.LogisticRegressionClassifier`), 543

`fit()` (`evalml.pipelines.components.estimators.classifiers.RandomForestClassifier`), 529

`fit()` (`evalml.pipelines.components.estimators.classifiers.RandomForestClassifier`), 581

`fit()` (`evalml.pipelines.components.estimators.classifiers.RandomForestClassifier`), 497

`fit()` (`evalml.pipelines.components.estimators.classifiers.RandomForestClassifier`), 547

`fit()` (`evalml.pipelines.components.estimators.classifiers.RandomForestClassifier`), 500

`fit()` (`evalml.pipelines.components.estimators.classifiers.RandomForestClassifier`), 584

`fit()` (`evalml.pipelines.components.estimators.classifiers.SVMClassifier`), 532

`fit()` (`evalml.pipelines.components.estimators.classifiers.SVMClassifier`), 549

`fit()` (`evalml.pipelines.components.estimators.classifiers.XGBoostClassifier`), 503

`fit()` (`evalml.pipelines.components.estimators.classifiers.XGBoostClassifier`), 586

`fit()` (`evalml.pipelines.components.estimators.classifiers.XGBoostClassifier`), 534

`fit()` (`evalml.pipelines.components.estimators.classifiers.XGBoostClassifier`), 553

`fit()` (`evalml.pipelines.components.estimators.DecisionTreeClassifier`), 627

`fit()` (`evalml.pipelines.components.estimators.DecisionTreeClassifier`), 589

`fit()` (`evalml.pipelines.components.estimators.DecisionTreeRegressor`), 630

`fit()` (`evalml.pipelines.components.estimators.DecisionTreeRegressor`), 556

`fit()` (`evalml.pipelines.components.estimators.ElasticNetClassifier`), 633

`fit()` (`evalml.pipelines.components.estimators.ElasticNetClassifier`), 592

`fit()` (`evalml.pipelines.components.estimators.ElasticNetRegressor`), 635

`fit()` (`evalml.pipelines.components.estimators.ElasticNetRegressor`), 559

`fit()` (`evalml.pipelines.components.estimators.Estimator`), 637

`fit()` (`evalml.pipelines.components.estimators.Estimator`), 595

`fit()` (`evalml.pipelines.components.estimators.estimator.Estimator`), 610

`fit()` (`evalml.pipelines.components.estimators.estimator.Estimator`), 562

`fit()` (`evalml.pipelines.components.estimators.ExtraTreesClassifier`), 640

`fit()` (`evalml.pipelines.components.estimators.ExtraTreesClassifier`), 597

`fit()` (`evalml.pipelines.components.estimators.ExtraTreesRegressor`), 643

`fit()` (`evalml.pipelines.components.estimators.ExtraTreesRegressor`), 600

`fit()` (`evalml.pipelines.components.estimators.KNeighborsClassifier`), 646

`fit()` (`evalml.pipelines.components.estimators.KNeighborsClassifier`), 564

`fit()` (`evalml.pipelines.components.estimators.LightGBMClassifier`), 649

`fit()` (`evalml.pipelines.components.estimators.LightGBMClassifier`), 567

`fit()` (`evalml.pipelines.components.estimators.LightGBMRegressor`), 652

`fit()` (`evalml.pipelines.components.estimators.LightGBMRegressor`), 602

`fit()` (`evalml.pipelines.components.estimators.LinearRegression`), 654

`fit()` (`evalml.pipelines.components.estimators.LinearRegression`), 570

`fit()` (`evalml.pipelines.components.estimators.LogisticRegressionClassifier`), 657

`fit()` (`evalml.pipelines.components.estimators.LogisticRegressionClassifier`), 604

`fit()` (`evalml.pipelines.components.estimators.ProphetRegressor`), 659

`fit()` (`evalml.pipelines.components.estimators.ProphetRegressor`), 573

`fit()` (`evalml.pipelines.components.estimators.RandomForestClassifier`), 662

`fit()` (`evalml.pipelines.components.estimators.RandomForestClassifier`), 607

`fit()` (`evalml.pipelines.components.estimators.RandomForestRegressor`), 664

`fit()` (`evalml.pipelines.components.estimators.RandomForestRegressor`), 666

`fit()` (`evalml.pipelines.components.estimators.regressors.untimed_regressor`), 538

`fit()` (`evalml.pipelines.components.estimators.regressors.untimed_regressor`), 669

`fit()` (`evalml.pipelines.components.estimators.regressors.ARIMARegressor`), 576

`fit()` (`evalml.pipelines.components.estimators.regressors.ARIMARegressor`), 671

`fit()` (`evalml.pipelines.components.estimators.XGBoostClassifier` method), 674  
`fit()` (`evalml.pipelines.components.estimators.XGBoostRegressor` method), 676  
`fit()` (`evalml.pipelines.components.ExtraTreesClassifier` method), 960  
`fit()` (`evalml.pipelines.components.ExtraTreesRegressor` method), 963  
`fit()` (`evalml.pipelines.components.FeatureSelector` method), 965  
`fit()` (`evalml.pipelines.components.Imputer` method), 967  
`fit()` (`evalml.pipelines.components.KNeighborsClassifier` method), 970  
`fit()` (`evalml.pipelines.components.LightGBMClassifier` method), 973  
`fit()` (`evalml.pipelines.components.LightGBMRegressor` method), 976  
`fit()` (`evalml.pipelines.components.LinearDiscriminantAnalysis` method), 978  
`fit()` (`evalml.pipelines.components.LinearRegressor` method), 980  
`fit()` (`evalml.pipelines.components.LogisticRegressionClassifier` method), 983  
`fit()` (`evalml.pipelines.components.LogTransformer` method), 985  
`fit()` (`evalml.pipelines.components.LSA` method), 987  
`fit()` (`evalml.pipelines.components.OneHotEncoder` method), 990  
`fit()` (`evalml.pipelines.components.PCA` method), 992  
`fit()` (`evalml.pipelines.components.PerColumnImputer` method), 994  
`fit()` (`evalml.pipelines.components.PolynomialDetrender` method), 997  
`fit()` (`evalml.pipelines.components.ProphetRegressor` method), 999  
`fit()` (`evalml.pipelines.components.RandomForestClassifier` method), 1002  
`fit()` (`evalml.pipelines.components.RandomForestRegressor` method), 1004  
`fit()` (`evalml.pipelines.components.RFClassifierSelectFromModel` method), 1006  
`fit()` (`evalml.pipelines.components.RFRegressorSelectFromModel` method), 1009  
`fit()` (`evalml.pipelines.components.SelectByType` method), 1011  
`fit()` (`evalml.pipelines.components.SelectColumns` method), 1013  
`fit()` (`evalml.pipelines.components.SimpleImputer` method), 1016  
`fit()` (`evalml.pipelines.components.SklearnStackedEnsembleClassifier` method), 1018  
`fit()` (`evalml.pipelines.components.SklearnStackedEnsembleRegressor` method), 1021  
`fit()` (`evalml.pipelines.components.SMOTEOverSampler` method), 1023  
`fit()` (`evalml.pipelines.components.SMOTENOverSampler` method), 1025  
`fit()` (`evalml.pipelines.components.SMOTEOverSampler` method), 1028  
`fit()` (`evalml.pipelines.components.StandardScaler` method), 1030  
`fit()` (`evalml.pipelines.components.SVMClassifier` method), 1032  
`fit()` (`evalml.pipelines.components.SVMRegressor` method), 1034  
`fit()` (`evalml.pipelines.components.TargetEncoder` method), 1037  
`fit()` (`evalml.pipelines.components.TargetImputer` method), 1039  
`fit()` (`evalml.pipelines.components.TextFeaturizer` method), 1041  
`fit()` (`evalml.pipelines.components.TimeSeriesBaselineEstimator` method), 1043  
`fit()` (`evalml.pipelines.components.Transformer` method), 1045  
`fit()` (`evalml.pipelines.components.transformers.column_selectors.ColumnSelector` method), 824  
`fit()` (`evalml.pipelines.components.transformers.column_selectors.DropColumns` method), 826  
`fit()` (`evalml.pipelines.components.transformers.column_selectors.SelectColumns` method), 828  
`fit()` (`evalml.pipelines.components.transformers.column_selectors.SelectColumnsByType` method), 830  
`fit()` (`evalml.pipelines.components.transformers.DateTimeFeaturizer` method), 839  
`fit()` (`evalml.pipelines.components.transformers.DelayedFeatureTransformer` method), 841  
`fit()` (`evalml.pipelines.components.transformers.DFSTransformer` method), 843  
`fit()` (`evalml.pipelines.components.transformers.dimensionality_reduction.ReducedDim` method), 679  
`fit()` (`evalml.pipelines.components.transformers.dimensionality_reduction.ReducedDim` method), 684  
`fit()` (`evalml.pipelines.components.transformers.dimensionality_reduction.ReducedDim` method), 686  
`fit()` (`evalml.pipelines.components.transformers.dimensionality_reduction.ReducedDim` method), 681  
`fit()` (`evalml.pipelines.components.transformers.DropColumns` method), 846  
`fit()` (`evalml.pipelines.components.transformers.DropNullColumns` method), 848  
`fit()` (`evalml.pipelines.components.transformers.EmailFeaturizer` method), 850  
`fit()` (`evalml.pipelines.components.transformers.encoders.onehot_encoder.OneHotEncoder` method), 689  
`fit()` (`evalml.pipelines.components.transformers.encoders.OneHotEncoder` method), 696

[fit \(\) \(evalml.pipelines.components.transformers.encoded\\_target\\_encoder.TargetEncoder method\), 693](#)  
[fit \(\) \(evalml.pipelines.components.transformers.encoded\\_target\\_encoder.TargetEncoder method\), 699](#)  
[fit \(\) \(evalml.pipelines.components.transformers.feature\\_selection.FeatureSelector method\), 702](#)  
[fit \(\) \(evalml.pipelines.components.transformers.feature\\_selection.FeatureSelector method\), 711](#)  
[fit \(\) \(evalml.pipelines.components.transformers.feature\\_selection.FeatureSelector method\), 705](#)  
[fit \(\) \(evalml.pipelines.components.transformers.feature\\_selection.FeatureSelector method\), 708](#)  
[fit \(\) \(evalml.pipelines.components.transformers.feature\\_selection.FeatureSelector method\), 714](#)  
[fit \(\) \(evalml.pipelines.components.transformers.feature\\_selection.FeatureSelector method\), 717](#)  
[fit \(\) \(evalml.pipelines.components.transformers.FeatureSelector \(evalml.pipelines.components.transformers.preprocessing.LogTransformer method\), 852](#)  
[fit \(\) \(evalml.pipelines.components.transformers.Imputer fit \(\) \(evalml.pipelines.components.transformers.preprocessing.LSA method\), 854](#)  
[fit \(\) \(evalml.pipelines.components.transformers.imputers.Imputer \(evalml.pipelines.components.transformers.preprocessing.Lsa.LSA method\), 731](#)  
[fit \(\) \(evalml.pipelines.components.transformers.imputers.Imputer \(evalml.pipelines.components.transformers.preprocessing.polynomial method\), 720](#)  
[fit \(\) \(evalml.pipelines.components.transformers.imputers.Pier\\_Col \(evalml.pipelines.components.transformers.preprocessing.Polynomial method\), 722](#)  
[fit \(\) \(evalml.pipelines.components.transformers.imputers.Pier\\_Col \(evalml.pipelines.components.transformers.preprocessing.text\\_features method\), 734](#)  
[fit \(\) \(evalml.pipelines.components.transformers.imputers.Simple \(evalml.pipelines.components.transformers.preprocessing.text\\_transform method\), 725](#)  
[fit \(\) \(evalml.pipelines.components.transformers.imputers.Simple \(evalml.pipelines.components.transformers.preprocessing.TextFeatures method\), 736](#)  
[fit \(\) \(evalml.pipelines.components.transformers.imputers.Target \(evalml.pipelines.components.transformers.preprocessing.TextTransformer method\), 728](#)  
[fit \(\) \(evalml.pipelines.components.transformers.imputers.Target \(evalml.pipelines.components.transformers.preprocessing.transform method\), 738](#)  
[fit \(\) \(evalml.pipelines.components.transformers.LinearDiscriminantAnalysis \(evalml.pipelines.components.transformers.preprocessing.transform method\), 856](#)  
[fit \(\) \(evalml.pipelines.components.transformers.LogTransformer \(evalml.pipelines.components.transformers.preprocessing.URLFeatures method\), 858](#)  
[fit \(\) \(evalml.pipelines.components.transformers.LSA fit \(\) \(evalml.pipelines.components.transformers.RFClassifierSelectFrom method\), 861](#)  
[fit \(\) \(evalml.pipelines.components.transformers.OneHotEncoder \(evalml.pipelines.components.transformers.RFRegressorSelectFrom method\), 863](#)  
[fit \(\) \(evalml.pipelines.components.transformers.PCA fit \(\) \(evalml.pipelines.components.transformers.samplers.base\\_sampler method\), 866](#)  
[fit \(\) \(evalml.pipelines.components.transformers.PerColumnImputer \(evalml.pipelines.components.transformers.samplers.base\\_sampler method\), 868](#)  
[fit \(\) \(evalml.pipelines.components.transformers.PolynomialDegree \(evalml.pipelines.components.transformers.samplers.oversamplers method\), 870](#)  
[fit \(\) \(evalml.pipelines.components.transformers.preprocessing.delayed\\_target\\_encoder.DelayedTargetEncoder \(evalml.pipelines.components.transformers.samplers.oversamplers method\), 741](#)  
[fit \(\) \(evalml.pipelines.components.transformers.preprocessing.DelayedTargetEncoder \(evalml.pipelines.components.transformers.samplers.oversamplers method\), 770](#)

[fit \(\) \(evalml.pipelines.components.transformers.samplers.SMOTEOverSampler method\), 810](#)  
[fit \(\) \(evalml.pipelines.components.transformers.samplers.SMOTEOverSampler method\), 812](#)  
[fit \(\) \(evalml.pipelines.components.transformers.samplers.SMOTEOverSampler method\), 814](#)  
[fit \(\) \(evalml.pipelines.components.transformers.samplers.Undersampler method\), 817](#)  
[fit \(\) \(evalml.pipelines.components.transformers.samplers.Undersampler method\), 807](#)  
[fit \(\) \(evalml.pipelines.components.transformers.scalers.StandardScaler method\), 819](#)  
[fit \(\) \(evalml.pipelines.components.transformers.scalers.StandardScaler method\), 822](#)  
[fit \(\) \(evalml.pipelines.components.transformers.SelectByType\(\) method\), 878](#)  
[fit \(\) \(evalml.pipelines.components.transformers.SelectColumns method\), 880](#)  
[fit \(\) \(evalml.pipelines.components.transformers.SimpleImputer method\), 883](#)  
[fit \(\) \(evalml.pipelines.components.transformers.SMOTEOverSampler method\), 885](#)  
[fit \(\) \(evalml.pipelines.components.transformers.SMOTEOverSampler method\), 887](#)  
[fit \(\) \(evalml.pipelines.components.transformers.SMOTEOverSampler method\), 890](#)  
[fit \(\) \(evalml.pipelines.components.transformers.StandardScaler method\), 892](#)  
[fit \(\) \(evalml.pipelines.components.transformers.TargetEncoder method\), 894](#)  
[fit \(\) \(evalml.pipelines.components.transformers.TargetImputer method\), 896](#)  
[fit \(\) \(evalml.pipelines.components.transformers.TextFeaturizer method\), 899](#)  
[fit \(\) \(evalml.pipelines.components.transformers.Transformer method\), 901](#)  
[fit \(\) \(evalml.pipelines.components.transformers.transformer.TargetTransformer method\), 832](#)  
[fit \(\) \(evalml.pipelines.components.transformers.transformer.TargetTransformer method\), 835](#)  
[fit \(\) \(evalml.pipelines.components.transformers.Undersampler method\), 903](#)  
[fit \(\) \(evalml.pipelines.components.transformers.URLFeaturizer method\), 905](#)  
[fit \(\) \(evalml.pipelines.components.Undersampler method\), 1048](#)  
[fit \(\) \(evalml.pipelines.components.URLFeaturizer method\), 1050](#)  
[fit \(\) \(evalml.pipelines.components.utils.WrappedSKClassifier method\), 912](#)  
[fit \(\) \(evalml.pipelines.components.utils.WrappedSKRegressor method\), 913](#)  
[fit \(\) \(evalml.pipelines.components.XGBoostClassifier method\), 1053](#)  
[fit \(\) \(evalml.pipelines.components.XGBoostRegressor method\), 1055](#)  
[fit \(\) \(evalml.pipelines.DecisionTreeClassifier method\), 1117](#)  
[fit \(\) \(evalml.pipelines.DecisionTreeRegressor method\), 1120](#)  
[fit \(\) \(evalml.pipelines.DelayedFeatureTransformer method\), 1122](#)  
[fit \(\) \(evalml.pipelines.DelayedFeatureTransformer method\), 1124](#)  
[fit \(\) \(evalml.pipelines.ElasticNetClassifier method\), 1127](#)  
[fit \(\) \(evalml.pipelines.ElasticNetRegressor method\), 1130](#)  
[fit \(\) \(evalml.pipelines.Estimator method\), 1132](#)  
[fit \(\) \(evalml.pipelines.ExtraTreesClassifier method\), 1135](#)  
[fit \(\) \(evalml.pipelines.ExtraTreesRegressor method\), 1138](#)  
[fit \(\) \(evalml.pipelines.FeatureSelector method\), 1140](#)  
[fit \(\) \(evalml.pipelines.KNeighborsClassifier method\), 1143](#)  
[fit \(\) \(evalml.pipelines.LightGBMClassifier method\), 1145](#)  
[fit \(\) \(evalml.pipelines.LightGBMRegressor method\), 1148](#)  
[fit \(\) \(evalml.pipelines.LinearRegressor method\), 1150](#)  
[fit \(\) \(evalml.pipelines.LogisticRegressionClassifier method\), 1153](#)  
[fit \(\) \(evalml.pipelines.multiclass\\_classification\\_pipeline.MulticlassClassifier method\), 1072](#)  
[fit \(\) \(evalml.pipelines.MulticlassClassificationPipeline method\), 1156](#)  
[fit \(\) \(evalml.pipelines.OneHotEncoder method\), 1160](#)  
[fit \(\) \(evalml.pipelines.PerColumnImputer method\), 1162](#)  
[fit \(\) \(evalml.pipelines.pipeline\\_base.PipelineBase method\), 1076](#)  
[fit \(\) \(evalml.pipelines.PipelineBase method\), 1165](#)  
[fit \(\) \(evalml.pipelines.ProphetRegressor method\), 1168](#)  
[fit \(\) \(evalml.pipelines.RandomForestClassifier method\), 1171](#)  
[fit \(\) \(evalml.pipelines.RandomForestRegressor method\), 1173](#)  
[fit \(\) \(evalml.pipelines.regression\\_pipeline.RegressionPipeline method\), 1082](#)  
[fit \(\) \(evalml.pipelines.RegressionPipeline method\), 1176](#)  
[fit \(\) \(evalml.pipelines.RFClassifierSelectFromModel method\), 1179](#)  
[fit \(\) \(evalml.pipelines.RFRegressorSelectFromModel method\), 1182](#)  
[fit \(\) \(evalml.pipelines.SimpleImputer method\), 1184](#)



`fit()` (`evalml.pipelines.SklearnStackedEnsembleClassifier` method), 965  
`fit()` (`evalml.pipelines.SklearnStackedEnsembleRegressor` method), 1187  
`fit()` (`evalml.pipelines.SklearnStackedEnsembleRegressor` method), 1189  
`fit()` (`evalml.pipelines.StandardScaler` method), 1191  
`fit()` (`evalml.pipelines.SVMClassifier` method), 1194  
`fit()` (`evalml.pipelines.SVMRegressor` method), 1196  
`fit()` (`evalml.pipelines.TargetEncoder` method), 1198  
`fit()` (`evalml.pipelines.time_series_classification_pipelines.TimeSeriesBinaryClassificationPipeline` method), 1087  
`fit()` (`evalml.pipelines.time_series_classification_pipelines.TimeSeriesBinaryClassificationPipeline` method), 1091  
`fit()` (`evalml.pipelines.time_series_classification_pipelines.TimeSeriesMulticlassClassificationPipeline` method), 1096  
`fit()` (`evalml.pipelines.time_series_regression_pipeline.TimeSeriesRegressionPipeline` method), 1100  
`fit()` (`evalml.pipelines.TimeSeriesBinaryClassificationPipeline` method), 1201  
`fit()` (`evalml.pipelines.TimeSeriesClassificationPipeline` method), 1206  
`fit()` (`evalml.pipelines.TimeSeriesMulticlassClassificationPipeline` method), 1210  
`fit()` (`evalml.pipelines.TimeSeriesRegressionPipeline` method), 1214  
`fit()` (`evalml.pipelines.Transformer` method), 1217  
`fit()` (`evalml.pipelines.XGBoostClassifier` method), 1220  
`fit()` (`evalml.pipelines.XGBoostRegressor` method), 1222  
`fit_features()` (`evalml.pipelines.component_graph.ComponentGraph` method), 1068  
`fit_features()` (`evalml.pipelines.ComponentGraph` method), 1114  
`fit_resample()` (`evalml.preprocessing.data_splitters.balanced_classification_sampler.BalancedClassificationSampler` method), 1224  
`fit_resample()` (`evalml.preprocessing.data_splitters.BalancedClassificationSampler` method), 1228  
`fit_resample()` (`evalml.preprocessing.data_splitters.sampler_base.SamplerBase` method), 1225  
`fit_resample()` (`evalml.preprocessing.data_splitters.SamplerBase` method), 1229  
`fit_transform()` (`evalml.pipelines.components.DateTimeFeaturizer` method), 933  
`fit_transform()` (`evalml.pipelines.components.DelayedFeatureTransformer` method), 941  
`fit_transform()` (`evalml.pipelines.components.DFSTransformer` method), 943  
`fit_transform()` (`evalml.pipelines.components.DropColumns` method), 945  
`fit_transform()` (`evalml.pipelines.components.DropNullColumns` method), 947  
`fit_transform()` (`evalml.pipelines.components.EmailFeaturizer` method), 955  
`fit_transform()` (`evalml.pipelines.components.FeatureSelector` method), 965  
`fit_transform()` (`evalml.pipelines.components.Imputer` method), 967  
`fit_transform()` (`evalml.pipelines.components.LinearDiscriminantAnalysis` method), 978  
`fit_transform()` (`evalml.pipelines.components.LogTransformer` method), 985  
`fit_transform()` (`evalml.pipelines.components.LSA` method), 987  
`fit_transform()` (`evalml.pipelines.components.OneHotEncoder` method), 990  
`fit_transform()` (`evalml.pipelines.components.PCA` method), 993  
`fit_transform()` (`evalml.pipelines.components.PerColumnImputer` method), 997  
`fit_transform()` (`evalml.pipelines.components.PolynomialDetrender` method), 1007  
`fit_transform()` (`evalml.pipelines.components.RFClassifierSelectFromModel` method), 1009  
`fit_transform()` (`evalml.pipelines.components.RFRegressorSelectFromModel` method), 1012  
`fit_transform()` (`evalml.pipelines.components.SelectByType` method), 1014  
`fit_transform()` (`evalml.pipelines.components.SelectColumns` method), 1016  
`fit_transform()` (`evalml.pipelines.components.SimpleImputer` method), 1023  
`fit_transform()` (`evalml.pipelines.components.SMOTEOverSampler` method), 1026  
`fit_transform()` (`evalml.pipelines.components.SMOTEOverSampler` method), 1028  
`fit_transform()` (`evalml.pipelines.components.StandardScaler` method), 1030  
`fit_transform()` (`evalml.pipelines.components.TargetEncoder` method), 1037  
`fit_transform()` (`evalml.pipelines.components.TargetImputer` method), 1039  
`fit_transform()` (`evalml.pipelines.components.TextFeaturizer` method), 1041  
`fit_transform()` (`evalml.pipelines.components.Transformer` method), 1046  
`fit_transform()` (`evalml.pipelines.components.transformers.column_transformer.ColumnTransformer` method), 824  
`fit_transform()` (`evalml.pipelines.components.transformers.column_transformer.ColumnTransformer` method), 826  
`fit_transform()` (`evalml.pipelines.components.transformers.column_transformer.ColumnTransformer` method), 828  
`fit_transform()` (`evalml.pipelines.components.transformers.column_transformer.ColumnTransformer` method), 830  
`fit_transform()` (`evalml.pipelines.components.transformers.DateTimeFeaturizer` method), 839  
`fit_transform()` (`evalml.pipelines.components.transformers.DelayedFeatureTransformer` method), 841



`method`), 789  
`fit_transform()` (`evalml.pipelines.components.transformers.preprocessing.TrainTestSplitter`), 790  
`method`), 765  
`fit_transform()` (`evalml.pipelines.components.transformers.preprocessing.URLFeaturizer`), 767  
`method`), 767  
`fit_transform()` (`evalml.pipelines.components.transformers.preprocessing.URLFeaturizer`), 792  
`method`), 792  
`fit_transform()` (`evalml.pipelines.components.transformers.preprocessing.URLFeaturizer`), 874  
`method`), 874  
`fit_transform()` (`evalml.pipelines.components.transformers.preprocessing.URLFeaturizer`), 876  
`method`), 876  
`fit_transform()` (`evalml.pipelines.components.transformers.preprocessing.URLFeaturizer`), 795  
`method`), 795  
`fit_transform()` (`evalml.pipelines.components.transformers.preprocessing.URLFeaturizer`), 797  
`method`), 797  
`fit_transform()` (`evalml.pipelines.components.transformers.preprocessing.URLFeaturizer`), 800  
`method`), 800  
`fit_transform()` (`evalml.pipelines.components.transformers.preprocessing.URLFeaturizer`), 802  
`method`), 802  
`fit_transform()` (`evalml.pipelines.components.transformers.preprocessing.URLFeaturizer`), 804  
`method`), 804  
`fit_transform()` (`evalml.pipelines.components.transformers.preprocessing.URLFeaturizer`), 810  
`method`), 810  
`fit_transform()` (`evalml.pipelines.components.transformers.preprocessing.URLFeaturizer`), 812  
`method`), 812  
`fit_transform()` (`evalml.pipelines.components.transformers.preprocessing.URLFeaturizer`), 815  
`method`), 815  
`fit_transform()` (`evalml.pipelines.components.transformers.preprocessing.URLFeaturizer`), 817  
`method`), 817  
`fit_transform()` (`evalml.pipelines.components.transformers.preprocessing.URLFeaturizer`), 807  
`method`), 807  
`fit_transform()` (`evalml.pipelines.components.transformers.preprocessing.URLFeaturizer`), 819  
`method`), 819  
`fit_transform()` (`evalml.pipelines.components.transformers.preprocessing.URLFeaturizer`), 822  
`method`), 822  
`fit_transform()` (`evalml.pipelines.components.transformers.preprocessing.URLFeaturizer`), 879  
`method`), 879  
`fit_transform()` (`evalml.pipelines.components.transformers.preprocessing.URLFeaturizer`), 881  
`method`), 881  
`fit_transform()` (`evalml.pipelines.components.transformers.preprocessing.URLFeaturizer`), 883  
`method`), 883  
`fit_transform()` (`evalml.pipelines.components.transformers.preprocessing.URLFeaturizer`), 885  
`method`), 885  
`fit_transform()` (`evalml.pipelines.components.transformers.preprocessing.URLFeaturizer`), 888  
`method`), 888  
`fit_transform()` (`evalml.pipelines.components.transformers.preprocessing.URLFeaturizer`), 890  
`method`), 890  
`fit_transform()` (`evalml.pipelines.components.transformers.preprocessing.URLFeaturizer`), 892  
`method`), 892  
`fit_transform()` (`evalml.pipelines.components.transformers.preprocessing.URLFeaturizer`), 894  
`method`), 894  
`fit_transform()` (`evalml.pipelines.components.transformers.preprocessing.URLFeaturizer`), 897  
`method`), 897  
`fit_transform()` (`evalml.pipelines.components.transformers.preprocessing.URLFeaturizer`), 1068  
`method`), 1068

`generate_order()` (*evalml.pipelines.ComponentGraph* *class method*), 1114  
`generate_pipeline_code()` (in module *evalml.pipelines.utils*), 1103  
`get_all_objective_names()` (in module *evalml.objectives*), 404  
`get_all_objective_names()` (in module *evalml.objectives.utils*), 372  
`get_best_sampler_for_data()` (in module *evalml.automl.utils*), 189  
`get_component()` (*evalml.pipelines.binary\_classification\_pipeline.BinaryClassificationPipeline* *method*), 1059  
`get_component()` (*evalml.pipelines.classification\_pipeline.ClassificationPipeline* *method*), 1064  
`get_component()` (*evalml.pipelines.component\_graph.ComponentGraph* *method*), 1068  
`get_component()` (*evalml.pipelines.ComponentGraph* *method*), 1114  
`get_component()` (*evalml.pipelines.multiclass\_classification\_pipeline.MulticlassClassificationPipeline* *method*), 1072  
`get_component()` (*evalml.pipelines.MulticlassClassificationPipeline* *method*), 1156  
`get_component()` (*evalml.pipelines.pipeline\_base.PipelineBase* *method*), 1077  
`get_component()` (*evalml.pipelines.PipelineBase* *method*), 1165  
`get_component()` (*evalml.pipelines.regression\_pipeline.RegressionPipeline* *method*), 1082  
`get_component()` (*evalml.pipelines.RegressionPipeline* *method*), 1176  
`get_component()` (*evalml.pipelines.time\_series\_classification\_pipeline.TimeSeriesBinaryClassificationPipeline* *method*), 1087  
`get_component()` (*evalml.pipelines.time\_series\_classification\_pipeline.TimeSeriesClassificationPipeline* *method*), 1092  
`get_component()` (*evalml.pipelines.time\_series\_classification\_pipeline.TimeSeriesMulticlassClassificationPipeline* *method*), 1096  
`get_component()` (*evalml.pipelines.time\_series\_regression\_pipeline.TimeSeriesRegressionPipeline* *method*), 1100  
`get_component()` (*evalml.pipelines.TimeSeriesBinaryClassificationPipeline* *method*), 1201  
`get_component()` (*evalml.pipelines.TimeSeriesClassificationPipeline* *method*), 1206  
`get_component()` (*evalml.pipelines.TimeSeriesMulticlassClassificationPipeline* *method*), 1210  
`get_component()` (*evalml.pipelines.TimeSeriesRegressionPipeline* *method*), 1214  
`get_core_objective_names()` (in module *evalml.objectives*), 405  
`get_core_objective_names()` (in module *evalml.objectives.utils*), 372  
`get_core_objectives()` (in module *evalml.objectives*), 405  
`get_core_objectives()` (in module *evalml.objectives.utils*), 372  
`get_default_primary_search_objective()` (in module *evalml.automl*), 196  
`get_default_primary_search_objective()` (in module *evalml.automl.utils*), 189  
`get_estimators()` (*evalml.pipelines.component\_graph.ComponentGraph* *method*), 1069  
`get_estimators()` (*evalml.pipelines.ComponentGraph* *method*), 1114  
`get_estimators()` (in module *evalml.pipelines.components.utils*), 911  
`get_feature_names()` (*evalml.pipelines.BinaryClassificationPipeline* *method*), 1252  
`get_feature_names()` (*evalml.pipelines.components.DateTimeFeaturizer* *method*), 933  
`get_feature_names()` (*evalml.pipelines.components.OneHotEncoder* *method*), 990  
`get_feature_names()` (*evalml.pipelines.components.MulticlassClassificationPipeline* *method*), 1037  
`get_feature_names()` (*evalml.pipelines.components.TargetEncoder* *method*), 1037  
`get_feature_names()` (*evalml.pipelines.components.transformers.DateTimeFeaturizer* *method*), 839  
`get_feature_names()` (*evalml.pipelines.components.transformers.encoders.onehot\_encoder.OneHotEncoder* *method*), 690  
`get_feature_names()` (*evalml.pipelines.components.transformers.encoders.OneHotEncoder* *method*), 697  
`get_feature_names()` (*evalml.pipelines.components.transformers.target\_encoder.TargetEncoder* *method*), 700  
`get_feature_names()` (*evalml.pipelines.components.transformers.preprocessing.datetimetime\_featurizer.DateTimeFeaturizer* *method*), 742  
`get_feature_names()` (*evalml.pipelines.components.transformers.preprocessing.DateTimeFeaturizer* *method*), 770  
`get_feature_names()` (*evalml.pipelines.components.transformers.TargetEncoder* *method*), 894  
`get_feature_names()` (*evalml.pipelines.OneHotEncoder* *method*), 1160  
`get_feature_names()` (*evalml.pipelines.TargetEncoder* *method*), 1198



---

<code>get_hyperparameter_ranges()</code> ( <code>evalml.pipelines.binary_classification_pipeline.BinaryClassificationPipeline</code> method), 1059	<code>get_inputs()</code> ( <code>evalml.pipelines.ComponentGraph</code> method), 1069
<code>get_hyperparameter_ranges()</code> ( <code>evalml.pipelines.classification_pipeline.ClassificationPipeline</code> method), 1064	<code>get_installed_packages()</code> (in module <code>evalml.utils.cli_utils</code> ), 1252
<code>get_hyperparameter_ranges()</code> ( <code>evalml.pipelines.multiclass_classification_pipeline.MulticlassClassificationPipeline</code> method), 1072	<code>get_pipeline_component()</code> ( <code>evalml.pipelines.component_graph.ComponentGraph</code> method), 1069
<code>get_hyperparameter_ranges()</code> ( <code>evalml.pipelines.MulticlassClassificationPipeline</code> method), 1156	<code>get_pipeline_graph()</code> ( <code>evalml.pipelines.ComponentGraph</code> method), 1115
<code>get_hyperparameter_ranges()</code> ( <code>evalml.pipelines.pipeline_base.PipelineBase</code> method), 1077	<code>get_linear_coefficients()</code> (in module <code>evalml.model_understanding</code> ), 280
<code>get_hyperparameter_ranges()</code> ( <code>evalml.pipelines.PipelineBase</code> method), 1165	<code>get_linear_coefficients()</code> (in module <code>evalml.model_understanding.graphs</code> ), 266
<code>get_hyperparameter_ranges()</code> ( <code>evalml.pipelines.regression_pipeline.RegressionPipeline</code> method), 1082	<code>get_logger()</code> (in module <code>evalml.utils</code> ), 1260
<code>get_hyperparameter_ranges()</code> ( <code>evalml.pipelines.RegressionPipeline</code> method), 1176	<code>get_logger()</code> (in module <code>evalml.utils.logger</code> ), 1256
<code>get_hyperparameter_ranges()</code> ( <code>evalml.pipelines.time_series_classification_pipeline.TimeSeriesBinaryClassificationPipeline</code> method), 1087	<code>get_n_splits()</code> ( <code>evalml.preprocessing.data_splitters.time_series_split</code> method), 1226
<code>get_hyperparameter_ranges()</code> ( <code>evalml.pipelines.time_series_classification_pipeline.TimeSeriesMulticlassClassificationPipeline</code> method), 1092	<code>get_n_splits()</code> ( <code>evalml.preprocessing.data_splitters.TimeSeriesSplit</code> method), 1229
<code>get_hyperparameter_ranges()</code> ( <code>evalml.pipelines.time_series_regression_pipeline.TimeSeriesRegressionPipeline</code> method), 1100	<code>get_n_splits()</code> ( <code>evalml.preprocessing.data_splitters.training_validation_split</code> static method), 1227
<code>get_hyperparameter_ranges()</code> ( <code>evalml.pipelines.TimeSeriesBinaryClassificationPipeline</code> method), 1201	<code>get_n_splits()</code> ( <code>evalml.preprocessing.data_splitters.TrainingValidationSplit</code> static method), 1230
<code>get_hyperparameter_ranges()</code> ( <code>evalml.pipelines.TimeSeriesClassificationPipeline</code> method), 1206	<code>get_n_splits()</code> ( <code>evalml.preprocessing.TimeSeriesSplit</code> static method), 1234
<code>get_hyperparameter_ranges()</code> ( <code>evalml.pipelines.TimeSeriesMulticlassClassificationPipeline</code> method), 1210	<code>get_n_splits()</code> ( <code>evalml.preprocessing.TrainingValidationSplit</code> static method), 1235
<code>get_hyperparameter_ranges()</code> ( <code>evalml.pipelines.TimeSeriesRegressionPipeline</code> method), 1214	<code>get_pipeline_components()</code> ( <code>evalml.pipelines.components.FeatureSelector</code> method), 965
<code>get_importable_subclasses()</code> (in module <code>evalml.utils</code> ), 1259	<code>get_names()</code> ( <code>evalml.pipelines.components.RFClassifierSelectFromModel</code> method), 1007
<code>get_importable_subclasses()</code> (in module <code>evalml.utils.gen_utils</code> ), 1254	<code>get_names()</code> ( <code>evalml.pipelines.components.RFRegressorSelectFromModel</code> method), 1010
<code>get_inputs()</code> ( <code>evalml.pipelines.component_graph.ComponentGraph</code> method), 1069	<code>get_names()</code> ( <code>evalml.pipelines.components.transformers.feature_selection</code> method), 702
	<code>get_names()</code> ( <code>evalml.pipelines.components.transformers.feature_selection</code> method), 711
	<code>get_names()</code> ( <code>evalml.pipelines.components.transformers.feature_selection</code> method), 705
	<code>get_names()</code> ( <code>evalml.pipelines.components.transformers.feature_selection</code> method), 709
	<code>get_names()</code> ( <code>evalml.pipelines.components.transformers.feature_selection</code> method), 714
	<code>get_names()</code> ( <code>evalml.pipelines.components.transformers.feature_selection</code> method), 717
	<code>get_names()</code> ( <code>evalml.pipelines.components.transformers.FeatureSelector</code> method), 852
	<code>get_names()</code> ( <code>evalml.pipelines.components.transformers.RFClassifierSelectFromModel</code> method), 874
	<code>get_names()</code> ( <code>evalml.pipelines.components.transformers.RFRegressorSelectFromModel</code> method), 877
	<code>get_pipeline_graph()</code> ( <code>evalml.pipelines.FeatureSelector</code> method), 1140

get_names() (evalml.pipelines.RFClassifierSelectFromModel method), 1180	get_result() (evalml.automl.engine.EngineComputation method), 178
get_names() (evalml.pipelines.RFRegressorSelectFromModel method), 1182	get_result() (evalml.automl.engine.sequential_engine.SequentialComputation method), 174
get_non_core_objectives() (in module evalml.objectives), 405	get_sys_info() (in module evalml.utils.cli_utils), 1252
get_non_core_objectives() (in module evalml.objectives.utils), 372	Gini (class in evalml.objectives), 405
get_objective() (in module evalml.objectives), 405	Gini (class in evalml.objectives.standard_metrics), 331
get_objective() (in module evalml.objectives.utils), 372	graph() (evalml.pipelines.binary_classification_pipeline.BinaryClassificationPipeline method), 1059
get_params() (evalml.pipelines.components.estimators.ProphetRegressor method), 659	graph() (evalml.pipelines.classification_pipeline.ClassificationPipeline method), 1064
get_params() (evalml.pipelines.components.estimators.regressors.prophet.ProphetRegressor method), 562	graph() (evalml.pipelines.component_graph.ComponentGraph method), 1062
get_params() (evalml.pipelines.components.estimators.regressors.ProphetRegressor method), 597	graph() (evalml.pipelines.ComponentGraph method), 1062
get_params() (evalml.pipelines.components.ProphetRegressor method), 999	graph() (evalml.pipelines.multiclass_classification_pipeline.MulticlassClassificationPipeline method), 1072
get_params() (evalml.pipelines.components.utils.WrappedSKClassifier method), 912	graph() (evalml.pipelines.MulticlassClassificationPipeline method), 1156
get_params() (evalml.pipelines.components.utils.WrappedSKRegressor method), 913	graph() (evalml.pipelines.pipeline_base.PipelineBase method), 1077
get_params() (evalml.pipelines.ProphetRegressor method), 1169	graph() (evalml.pipelines.PipelineBase method), 1165
get_pipeline() (evalml.automl.automl_search.AutoMLSearch method), 184	graph() (evalml.pipelines.regression_pipeline.RegressionPipeline method), 1083
get_pipeline() (evalml.automl.AutoMLSearch method), 194	graph() (evalml.pipelines.RegressionPipeline method), 1176
get_pipeline() (evalml.AutoMLSearch method), 1265	graph() (evalml.pipelines.time_series_classification_pipelines.TimeSeriesClassificationPipeline method), 1087
get_pipelines_from_component_graphs() (in module evalml.automl.utils), 189	graph() (evalml.pipelines.time_series_classification_pipelines.TimeSeriesClassificationPipeline method), 1092
get_prediction_vs_actual_data() (in module evalml.model_understanding), 280	graph() (evalml.pipelines.time_series_classification_pipelines.TimeSeriesClassificationPipeline method), 1096
get_prediction_vs_actual_data() (in module evalml.model_understanding.graphs), 266	graph() (evalml.pipelines.time_series_regression_pipeline.TimeSeriesRegressionPipeline method), 1101
get_prediction_vs_actual_over_time_data() (in module evalml.model_understanding), 280	graph() (evalml.pipelines.TimeSeriesBinaryClassificationPipeline method), 1202
get_prediction_vs_actual_over_time_data() (in module evalml.model_understanding.graphs), 267	graph() (evalml.pipelines.TimeSeriesClassificationPipeline method), 1206
get_random_seed() (in module evalml.utils), 1260	graph() (evalml.pipelines.TimeSeriesMulticlassClassificationPipeline method), 1210
get_random_seed() (in module evalml.utils.gen_utils), 1254	graph() (evalml.pipelines.TimeSeriesRegressionPipeline method), 1214
get_random_state() (in module evalml.utils), 1260	graph_binary_objective_vs_threshold() (in module evalml.model_understanding), 280
get_random_state() (in module evalml.utils.gen_utils), 1255	graph_binary_objective_vs_threshold() (in module evalml.model_understanding.graphs), 267
get_result() (evalml.automl.engine.cf_engine.CFComputation method), 167	importance_confusion_matrix() (in module evalml.model_understanding), 281
get_result() (evalml.automl.engine.dask_engine.DaskComputation method), 169	importance_confusion_matrix() (in module evalml.model_understanding.graphs), 267
get_result() (evalml.automl.engine.engine_base.EngineComputation method), 172	importance_feature_importance() (evalml.pipelines.binary_classification_pipeline.BinaryClassificationPipeline method), 1059

`method`), 1059  
`graph_feature_importance()` (`evalml.pipelines.classification_pipeline.ClassificationPipeline` `method`), 1065  
`graph_feature_importance()` (`evalml.pipelines.multiclass_classification_pipeline.MulticlassClassificationPipeline` `method`), 1072  
`graph_feature_importance()` (`evalml.pipelines.MulticlassClassificationPipeline` `method`), 1156  
`graph_feature_importance()` (`evalml.pipelines.pipeline_base.PipelineBase` `method`), 1077  
`graph_feature_importance()` (`evalml.pipelines.PipelineBase` `method`), 1166  
`graph_feature_importance()` (`evalml.pipelines.regression_pipeline.RegressionPipeline` `method`), 1083  
`graph_feature_importance()` (`evalml.pipelines.RegressionPipeline` `method`), 1176  
`graph_feature_importance()` (`evalml.pipelines.time_series_classification_pipeline.TimeSeriesBinaryClassificationPipeline` `method`), 1087  
`graph_feature_importance()` (`evalml.pipelines.time_series_classification_pipeline.TimeSeriesMulticlassClassificationPipeline` `method`), 1092  
`graph_feature_importance()` (`evalml.pipelines.time_series_classification_pipeline.TimeSeriesRegressionPipeline` `method`), 1096  
`graph_feature_importance()` (`evalml.pipelines.time_series_regression_pipeline.TimeSeriesRegressionPipeline` `method`), 1101  
`graph_feature_importance()` (`evalml.pipelines.TimeSeriesBinaryClassificationPipeline` `method`), 1202  
`graph_feature_importance()` (`evalml.pipelines.TimeSeriesClassificationPipeline` `method`), 1206  
`graph_feature_importance()` (`evalml.pipelines.TimeSeriesMulticlassClassificationPipeline` `method`), 1211  
`graph_feature_importance()` (`evalml.pipelines.TimeSeriesRegressionPipeline` `method`), 1215  
`graph_force_plot()` (in module `evalml.model_understanding.force_plots`), 263  
`graph_partial_dependence()` (in module `evalml.model_understanding`), 281  
`graph_partial_dependence()` (in module `evalml.model_understanding.graphs`), 267  
`graph_permutation_importance()` (in module `evalml.model_understanding`), 282  
`graph_permutation_importance()` (in module `evalml.model_understanding.graphs`), 268  
`graph_precision_recall_curve()` (in module `evalml.model_understanding`), 282  
`graph_precision_recall_curve()` (in module `evalml.model_understanding.graphs`), 269  
`graph_prediction_vs_actual()` (in module `evalml.model_understanding`), 282  
`graph_prediction_vs_actual()` (in module `evalml.model_understanding.graphs`), 269  
`graph_prediction_vs_actual_over_time()` (in module `evalml.model_understanding`), 283  
`graph_prediction_vs_actual_over_time()` (in module `evalml.model_understanding.graphs`), 269  
`graph_roc_curve()` (in module `evalml.model_understanding`), 283  
`graph_roc_curve()` (in module `evalml.model_understanding.graphs`), 270  
`graph_t_sne()` (in module `evalml.model_understanding`), 283  
`graph_t_sne()` (in module `evalml.model_understanding.graphs`), 270  
`greater_is_better()` (`evalml.objectives.binary_classification_objective.BinaryClassificationObjective` `property`), 428  
`greater_is_better()` (`evalml.objectives.BinaryClassificationObjective` `property`), 301  
`greater_is_better()` (`evalml.objectives.multiclass_classification_objective.MulticlassClassificationObjective` `property`), 428  
`greater_is_better()` (`evalml.objectives.MulticlassClassificationObjective` `property`), 301  
`greater_is_better()` (`evalml.objectives.objective_base.ObjectiveBase` `property`), 301  
`greater_is_better()` (`evalml.objectives.ObjectiveBase` `property`), 301  
`greater_is_better()` (`evalml.objectives.regression_objective.RegressionObjective` `property`), 303  
`greater_is_better()` (`evalml.objectives.RegressionObjective` `property`), 447  
`greater_is_better()` (`evalml.objectives.time_series_regression_objective.TimeSeriesRegressionObjective` `property`), 370  
`GridSearchTuner` (class in `evalml.tuners`), 1246  
`GridSearchTuner` (class in `evalml.tuners.grid_search_tuner`), 1241

## H

`handle_component_class()` (in module `evalml.pipelines.components.utils`), 911  
`handle_model_family()` (in module `evalml.model_family`), 255  
`handle_model_family()` (in module `evalml.model_family.utils`), 255  
`handle_problem_types()` (in module `evalml.problem_types`), 1239  
`handle_problem_types()` (in module `evalml.problem_types.utils`), 1237  
`HighlyNullDataCheck` (class in `evalml.data_checks`), 235  
`HighlyNullDataCheck` (class in `evalml.data_checks.highly_null_data_check`), 211

I

`IDColumnsDataCheck` (class in `evalml.data_checks`), 237  
`IDColumnsDataCheck` (class in `evalml.data_checks.id_columns_data_check`), 213  
`import_or_raise()` (in module `evalml.utils`), 1260  
`import_or_raise()` (in module `evalml.utils.gen_utils`), 1255  
`Imputer` (class in `evalml.pipelines.components`), 966  
`Imputer` (class in `evalml.pipelines.components.transformers`), 853  
`Imputer` (class in `evalml.pipelines.components.transformers.imputers`), 730  
`Imputer` (class in `evalml.pipelines.components.transformers.imputers.imputer`), 718  
`infer_feature_types()` (in module `evalml.utils`), 1260  
`infer_feature_types()` (in module `evalml.utils.woodwork_utils`), 1257  
`info()` (`evalml.automl.engine.engine_base.JobLogger` method), 172  
`instantiate()` (`evalml.pipelines.component_graph.ComponentGraph` method), 1069  
`instantiate()` (`evalml.pipelines.ComponentGraph` method), 1115  
`InvalidTargetDataCheck` (class in `evalml.data_checks`), 237  
`InvalidTargetDataCheck` (class in `evalml.data_checks.invalid_targets_data_check`), 214  
`inverse_transform()` (`evalml.pipelines.binary_classification_pipeline.BinaryClassificationPipeline` method), 1059  
`inverse_transform()` (`evalml.pipelines.classification_pipeline.ClassificationPipeline` method), 1065  
`inverse_transform()` (`evalml.pipelines.component_graph.ComponentGraph` method), 1069  
`inverse_transform()` (`evalml.pipelines.ComponentGraph` method), 1115  
`inverse_transform()` (`evalml.pipelines.components.LogTransformer` method), 985  
`inverse_transform()` (`evalml.pipelines.components.PolynomialDetrender` method), 997  
`inverse_transform()` (`evalml.pipelines.components.transformers.LogTransformer` method), 859  
`inverse_transform()` (`evalml.pipelines.components.transformers.PolynomialDetrender` method), 871  
`inverse_transform()` (`evalml.pipelines.components.transformers.preprocessing.log_transformer` method), 752  
`inverse_transform()` (`evalml.pipelines.components.transformers.preprocessing.LogTransformer` method), 781  
`inverse_transform()` (`evalml.pipelines.components.transformers.preprocessing.polynomial_detrender` method), 757  
`inverse_transform()` (`evalml.pipelines.components.transformers.preprocessing.PolynomialDetrender` method), 785  
`inverse_transform()` (`evalml.pipelines.components.transformers.transformer.TargetTransformer` method), 833  
`inverse_transform()` (`evalml.pipelines.multiclass_classification_pipeline.MulticlassClassificationPipeline` method), 1073  
`inverse_transform()` (`evalml.pipelines.MulticlassClassificationPipeline` method), 1157  
`inverse_transform()` (`evalml.pipelines.pipeline_base.PipelineBase` method), 1077  
`inverse_transform()` (`evalml.pipelines.PipelineBase` method), 1166  
`inverse_transform()` (`evalml.pipelines.regression_pipeline.RegressionPipeline` method), 1083  
`inverse_transform()` (`evalml.pipelines.RegressionPipeline` method), 1176  
`inverse_transform()` (`evalml.pipelines.time_series_classification_pipelines.TimeSeriesClassificationPipeline` method), 1087



[inverse\\_transform\(\)](#) ([evalml.pipelines.time\\_series\\_classification\\_pipelines.TimeSeriesClassificationPipeline](#) method), 1092  
[inverse\\_transform\(\)](#) ([evalml.pipelines.time\\_series\\_classification\\_pipelines.TimeSeriesMulticlassClassificationPipeline](#) method), 1096  
[inverse\\_transform\(\)](#) ([evalml.pipelines.time\\_series\\_regression\\_pipeline.TimeSeriesRegressionPipeline](#) method), 1101  
[inverse\\_transform\(\)](#) ([evalml.pipelines.TimeSeriesBinaryClassificationPipeline](#) method), 1202  
[inverse\\_transform\(\)](#) ([evalml.pipelines.TimeSeriesClassificationPipeline](#) method), 1206  
[inverse\\_transform\(\)](#) ([evalml.pipelines.TimeSeriesMulticlassClassificationPipeline](#) method), 1211  
[inverse\\_transform\(\)](#) ([evalml.pipelines.TimeSeriesRegressionPipeline](#) method), 1215  
[is\\_all\\_numeric\(\)](#) (in module [evalml.utils](#)), 1260  
[is\\_all\\_numeric\(\)](#) (in module [evalml.utils.gen\\_utils](#)), 1255  
[is\\_binary\(\)](#) (in module [evalml.problem\\_types](#)), 1239  
[is\\_binary\(\)](#) (in module [evalml.problem\\_types.utils](#)), 1237  
[is\\_bounded\\_like\\_percentage\(\)](#) ([evalml.objectives.binary\\_classification\\_objective.BinaryClassificationObjective](#) property), 288  
[is\\_bounded\\_like\\_percentage\(\)](#) ([evalml.objectives.BinaryClassificationObjective](#) property), 390  
[is\\_bounded\\_like\\_percentage\(\)](#) ([evalml.objectives.multiclass\\_classification\\_objective.MulticlassClassificationObjective](#) property), 299  
[is\\_bounded\\_like\\_percentage\(\)](#) ([evalml.objectives.MulticlassClassificationObjective](#) property), 428  
[is\\_bounded\\_like\\_percentage\(\)](#) ([evalml.objectives.objective\\_base.ObjectiveBase](#) property), 301  
[is\\_bounded\\_like\\_percentage\(\)](#) ([evalml.objectives.ObjectiveBase](#) property), 430  
[is\\_bounded\\_like\\_percentage\(\)](#) ([evalml.objectives.regression\\_objective.RegressionObjective](#) property), 304  
[is\\_bounded\\_like\\_percentage\(\)](#) ([evalml.objectives.RegressionObjective](#) property), 447  
[is\\_bounded\\_like\\_percentage\(\)](#) ([evalml.objectives.time\\_series\\_regression\\_objective.TimeSeriesRegressionObjective](#) property), 370  
[is\\_cancelled\(\)](#) ([evalml.automl.engine.cf\\_engine.CFComputation](#) property), 169  
[is\\_cancelled\(\)](#) ([evalml.automl.engine.dask\\_engine.DaskComputation](#) property), 169  
[is\\_classification\(\)](#) (in module [evalml.problem\\_types](#)), 1239  
[is\\_defined\\_for\\_problem\\_type\(\)](#) ([evalml.objectives.AccuracyBinary](#) class method), 375  
[is\\_defined\\_for\\_problem\\_type\(\)](#) ([evalml.objectives.AccuracyMulticlass](#) class method), 377  
[is\\_defined\\_for\\_problem\\_type\(\)](#) ([evalml.objectives.AUC](#) class method), 379  
[is\\_defined\\_for\\_problem\\_type\(\)](#) ([evalml.objectives.AUCMacro](#) class method), 381  
[is\\_defined\\_for\\_problem\\_type\(\)](#) ([evalml.objectives.AUCMicro](#) class method), 382  
[is\\_defined\\_for\\_problem\\_type\(\)](#) ([evalml.objectives.AUCWeighted](#) class method), 384  
[is\\_defined\\_for\\_problem\\_type\(\)](#) ([evalml.objectives.BalancedAccuracyBinary](#) class method), 386  
[is\\_defined\\_for\\_problem\\_type\(\)](#) ([evalml.objectives.BalancedAccuracyMulticlass](#) class method), 388  
[is\\_defined\\_for\\_problem\\_type\(\)](#) ([evalml.objectives.binary\\_classification\\_objective.BinaryClassificationObjective](#) class method), 288  
[is\\_defined\\_for\\_problem\\_type\(\)](#) ([evalml.objectives.MulticlassClassificationObjective](#) class method), 390  
[is\\_defined\\_for\\_problem\\_type\(\)](#) ([evalml.objectives.cost\\_benefit\\_matrix.CostBenefitMatrix](#) class method), 291  
[is\\_defined\\_for\\_problem\\_type\(\)](#) ([evalml.objectives.CostBenefitMatrix](#) class method), 392  
[is\\_defined\\_for\\_problem\\_type\(\)](#) ([evalml.objectives.ExpVariance](#) class method), 394  
[is\\_defined\\_for\\_problem\\_type\(\)](#) ([evalml.objectives.F1](#) class method), 396  
[is\\_defined\\_for\\_problem\\_type\(\)](#) ([evalml.objectives.F1Macro](#) class method), 398  
[is\\_defined\\_for\\_problem\\_type\(\)](#) ([evalml.objectives.F1Micro](#) class method), 400

<code>is_defined_for_problem_type()</code> ( <i>evalml.objectives.F1Weighted</i> class method), 401	<code>is_defined_for_problem_type()</code> ( <i>evalml.objectives.ObjectiveBase</i> class method), 430
<code>is_defined_for_problem_type()</code> ( <i>evalml.objectives.fraud_cost.FraudCost</i> class method), 294	<code>is_defined_for_problem_type()</code> ( <i>evalml.objectives.Precision</i> class method), 432
<code>is_defined_for_problem_type()</code> ( <i>evalml.objectives.FraudCost</i> class method), 403	<code>is_defined_for_problem_type()</code> ( <i>evalml.objectives.PrecisionMacro</i> class method), 434
<code>is_defined_for_problem_type()</code> ( <i>evalml.objectives.Gini</i> class method), 406	<code>is_defined_for_problem_type()</code> ( <i>evalml.objectives.PrecisionMicro</i> class method), 435
<code>is_defined_for_problem_type()</code> ( <i>evalml.objectives.lead_scoring.LeadScoring</i> class method), 297	<code>is_defined_for_problem_type()</code> ( <i>evalml.objectives.PrecisionWeighted</i> class method), 437
<code>is_defined_for_problem_type()</code> ( <i>evalml.objectives.LeadScoring</i> class method), 409	<code>is_defined_for_problem_type()</code> ( <i>evalml.objectives.R2</i> class method), 438
<code>is_defined_for_problem_type()</code> ( <i>evalml.objectives.LogLossBinary</i> class method), 411	<code>is_defined_for_problem_type()</code> ( <i>evalml.objectives.Recall</i> class method), 440
<code>is_defined_for_problem_type()</code> ( <i>evalml.objectives.LogLossMulticlass</i> class method), 413	<code>is_defined_for_problem_type()</code> ( <i>evalml.objectives.RecallMacro</i> class method), 442
<code>is_defined_for_problem_type()</code> ( <i>evalml.objectives.MAE</i> class method), 414	<code>is_defined_for_problem_type()</code> ( <i>evalml.objectives.RecallMicro</i> class method), 444
<code>is_defined_for_problem_type()</code> ( <i>evalml.objectives.MAPE</i> class method), 416	<code>is_defined_for_problem_type()</code> ( <i>evalml.objectives.RecallWeighted</i> class method), 445
<code>is_defined_for_problem_type()</code> ( <i>evalml.objectives.MaxError</i> class method), 417	<code>is_defined_for_problem_type()</code> ( <i>evalml.objectives.regression_objective.RegressionObjective</i> class method), 304
<code>is_defined_for_problem_type()</code> ( <i>evalml.objectives.MCCBinary</i> class method), 419	<code>is_defined_for_problem_type()</code> ( <i>evalml.objectives.RegressionObjective</i> class method), 447
<code>is_defined_for_problem_type()</code> ( <i>evalml.objectives.MCCMulticlass</i> class method), 421	<code>is_defined_for_problem_type()</code> ( <i>evalml.objectives.RootMeanSquaredError</i> class method), 449
<code>is_defined_for_problem_type()</code> ( <i>evalml.objectives.MeanSquaredLogError</i> class method), 423	<code>is_defined_for_problem_type()</code> ( <i>evalml.objectives.RootMeanSquaredLogError</i> class method), 451
<code>is_defined_for_problem_type()</code> ( <i>evalml.objectives.MedianAE</i> class method), 424	<code>is_defined_for_problem_type()</code> ( <i>evalml.objectives.sensitivity_low_alert.SensitivityLowAlert</i> class method), 306
<code>is_defined_for_problem_type()</code> ( <i>evalml.objectives.MSE</i> class method), 426	<code>is_defined_for_problem_type()</code> ( <i>evalml.objectives.SensitivityLowAlert</i> class method), 306
<code>is_defined_for_problem_type()</code> ( <i>evalml.objectives.multiclass_classification_objective.MulticlassClassificationObjective</i> class method), 299	<code>is_defined_for_problem_type()</code> ( <i>evalml.objectives.standard_metrics.AccuracyBinary</i> class method), 309
<code>is_defined_for_problem_type()</code> ( <i>evalml.objectives.MulticlassClassificationObjective</i> class method), 428	<code>is_defined_for_problem_type()</code> ( <i>evalml.objectives.standard_metrics.AccuracyMulticlass</i> class method), 311
<code>is_defined_for_problem_type()</code> ( <i>evalml.objectives.objective_base.ObjectiveBase</i> class method), 301	<code>is_defined_for_problem_type()</code>

<code>(evalml.objectives.standard_metrics.AUC</code> <code>class method), 313</code>	<code>(evalml.objectives.standard_metrics.MCCMulticlass</code> <code>class method), 345</code>
<code>is_defined_for_problem_type()</code> <code>(evalml.objectives.standard_metrics.AUCMacro</code> <code>class method), 315</code>	<code>is_defined_for_problem_type()</code> <code>(evalml.objectives.standard_metrics.MeanSquaredLogError</code> <code>class method), 346</code>
<code>is_defined_for_problem_type()</code> <code>(evalml.objectives.standard_metrics.AUCMicro</code> <code>class method), 317</code>	<code>is_defined_for_problem_type()</code> <code>(evalml.objectives.standard_metrics.MedianAE</code> <code>class method), 348</code>
<code>is_defined_for_problem_type()</code> <code>(evalml.objectives.standard_metrics.AUCWeighted</code> <code>class method), 318</code>	<code>is_defined_for_problem_type()</code> <code>(evalml.objectives.standard_metrics.MSE</code> <code>class method), 350</code>
<code>is_defined_for_problem_type()</code> <code>(evalml.objectives.standard_metrics.BalancedAccuracyBinary</code> <code>class method), 320</code>	<code>is_defined_for_problem_type()</code> <code>(evalml.objectives.standard_metrics.Precision</code> <code>class method), 352</code>
<code>is_defined_for_problem_type()</code> <code>(evalml.objectives.standard_metrics.BalancedAccuracyMulticlass</code> <code>class method), 322</code>	<code>is_defined_for_problem_type()</code> <code>(evalml.objectives.standard_metrics.PrecisionMacro</code> <code>class method), 353</code>
<code>is_defined_for_problem_type()</code> <code>(evalml.objectives.standard_metrics.ExpVariance</code> <code>class method), 324</code>	<code>is_defined_for_problem_type()</code> <code>(evalml.objectives.standard_metrics.PrecisionMicro</code> <code>class method), 355</code>
<code>is_defined_for_problem_type()</code> <code>(evalml.objectives.standard_metrics.F1 class</code> <code>method), 326</code>	<code>is_defined_for_problem_type()</code> <code>(evalml.objectives.standard_metrics.PrecisionWeighted</code> <code>class method), 356</code>
<code>is_defined_for_problem_type()</code> <code>(evalml.objectives.standard_metrics.F1Macro</code> <code>class method), 327</code>	<code>is_defined_for_problem_type()</code> <code>(evalml.objectives.standard_metrics.R2 class</code> <code>method), 358</code>
<code>is_defined_for_problem_type()</code> <code>(evalml.objectives.standard_metrics.F1Micro</code> <code>class method), 329</code>	<code>is_defined_for_problem_type()</code> <code>(evalml.objectives.standard_metrics.Recall</code> <code>class method), 360</code>
<code>is_defined_for_problem_type()</code> <code>(evalml.objectives.standard_metrics.F1Weighted</code> <code>class method), 330</code>	<code>is_defined_for_problem_type()</code> <code>(evalml.objectives.standard_metrics.RecallMacro</code> <code>class method), 362</code>
<code>is_defined_for_problem_type()</code> <code>(evalml.objectives.standard_metrics.Gini</code> <code>class method), 332</code>	<code>is_defined_for_problem_type()</code> <code>(evalml.objectives.standard_metrics.RecallMicro</code> <code>class method), 363</code>
<code>is_defined_for_problem_type()</code> <code>(evalml.objectives.standard_metrics.LogLossBinary</code> <code>class method), 335</code>	<code>is_defined_for_problem_type()</code> <code>(evalml.objectives.standard_metrics.RecallWeighted</code> <code>class method), 365</code>
<code>is_defined_for_problem_type()</code> <code>(evalml.objectives.standard_metrics.LogLossMulticlass</code> <code>class method), 336</code>	<code>is_defined_for_problem_type()</code> <code>(evalml.objectives.standard_metrics.RootMeanSquaredError</code> <code>class method), 366</code>
<code>is_defined_for_problem_type()</code> <code>(evalml.objectives.standard_metrics.MAE</code> <code>class method), 338</code>	<code>is_defined_for_problem_type()</code> <code>(evalml.objectives.standard_metrics.RootMeanSquaredLogError</code> <code>class method), 368</code>
<code>is_defined_for_problem_type()</code> <code>(evalml.objectives.standard_metrics.MAPE</code> <code>class method), 340</code>	<code>is_defined_for_problem_type()</code> <code>(evalml.objectives.time_series_regression_objective.TimeSeriesRe</code> <code>class method), 370</code>
<code>is_defined_for_problem_type()</code> <code>(evalml.objectives.standard_metrics.MaxError</code> <code>class method), 341</code>	<code>is_multiclass() (in module evalml.problem_types),</code> <code>1239</code>
<code>is_defined_for_problem_type()</code> <code>(evalml.objectives.standard_metrics.MCCBinary</code> <code>class method), 343</code>	<code>is_multiclass() (in module</code> <code>evalml.problem_types.utils), 1237</code>
<code>is_defined_for_problem_type()</code>	<code>is_regression() (in module evalml.problem_types),</code> <code>1239</code>
<code>is_defined_for_problem_type()</code>	<code>is_regression() (in module</code> <code>evalml.problem_types.utils), 1237</code>

<code>evalml.problem_types.utils</code> ), 1238	KNeighborsClassifier (class in <code>evalml.pipelines.components.estimators.classifiers</code> ), 518
<code>is_search_space_exhausted()</code> ( <code>evalml.tuners.grid_search_tuner.GridSearchTuner</code> method), 1241	KNeighborsClassifier (class in <code>evalml.pipelines.components.estimators.classifiers.kneighbors_classifier</code> ), 486
<code>is_search_space_exhausted()</code> ( <code>evalml.tuners.GridSearchTuner</code> method), 1247	<b>L</b>
<code>is_search_space_exhausted()</code> ( <code>evalml.tuners.random_search_tuner.RandomSearchTuner</code> method), 1243	LeadScoring (class in <code>evalml.objectives</code> ), 407
<code>is_search_space_exhausted()</code> ( <code>evalml.tuners.RandomSearchTuner</code> method), 1248	LeadScoring (class in <code>evalml.objectives.lead_scoring</code> ), 295
<code>is_search_space_exhausted()</code> ( <code>evalml.tuners.skopt_tuner.SKOptTuner</code> method), 1244	LightGBMClassifier (class in <code>evalml.pipelines</code> ), 1143
<code>is_search_space_exhausted()</code> ( <code>evalml.tuners.SKOptTuner</code> method), 1249	LightGBMClassifier (class in <code>evalml.pipelines.components</code> ), 971
<code>is_search_space_exhausted()</code> ( <code>evalml.tuners.Tuner</code> method), 1250	LightGBMClassifier (class in <code>evalml.pipelines.components.estimators</code> ), 647
<code>is_search_space_exhausted()</code> ( <code>evalml.tuners.tuner.Tuner</code> method), 1245	LightGBMClassifier (class in <code>evalml.pipelines.components.estimators.classifiers</code> ), 521
<code>is_time_series()</code> (in module <code>evalml.problem_types</code> ), 1240	LightGBMClassifier (class in <code>evalml.pipelines.components.estimators.classifiers.lightgbm_classifier</code> ), 490
<code>is_time_series()</code> (in module <code>evalml.problem_types.utils</code> ), 1238	LightGBMRegressor (class in <code>evalml.pipelines</code> ), 1146
<code>is_tree_estimator()</code> ( <code>evalml.model_family.model_family.ModelFamily</code> method), 254	LightGBMRegressor (class in <code>evalml.pipelines.components</code> ), 974
<code>is_tree_estimator()</code> ( <code>evalml.model_family.ModelFamily</code> method), 256	LightGBMRegressor (class in <code>evalml.pipelines.components.estimators</code> ), 650
IterativeAlgorithm (class in <code>evalml.automl.automl_algorithm</code> ), 164	LightGBMRegressor (class in <code>evalml.pipelines.components.estimators.regressors</code> ), 590
IterativeAlgorithm (class in <code>evalml.automl.automl_algorithm.iterative_algorithm</code> ), 160	LightGBMRegressor (class in <code>evalml.pipelines.components.estimators.regressors.lightgbm_regressor</code> ), 554
<b>J</b>	LinearDiscriminantAnalysis (class in <code>evalml.pipelines.components</code> ), 977
JobLogger (class in <code>evalml.engine.engine_base</code> ), 172	LinearDiscriminantAnalysis (class in <code>evalml.pipelines.components.transformers</code> ), 855
<code>jupyter_check()</code> (in module <code>evalml.utils</code> ), 1261	LinearDiscriminantAnalysis (class in <code>evalml.pipelines.components.transformers.dimensionality_reducers</code> ), 683
<code>jupyter_check()</code> (in module <code>evalml.utils.gen_utils</code> ), 1255	LinearDiscriminantAnalysis (class in <code>evalml.pipelines.components.transformers.dimensionality_reducers</code> ), 678
<b>K</b>	LinearRegressor (class in <code>evalml.pipelines</code> ), 1149
KNeighborsClassifier (class in <code>evalml.pipelines</code> ), 1141	LinearRegressor (class in <code>evalml.pipelines.components</code> ), 979
KNeighborsClassifier (class in <code>evalml.pipelines.components</code> ), 968	LinearRegressor (class in <code>evalml.pipelines.components.estimators</code> ), 653
KNeighborsClassifier (class in <code>evalml.pipelines.components.estimators</code> ), 644	



[LinearRegressor](#) (class in [static method](#)), 955  
[evalml.pipelines.components.estimators.regressors.load\(\)](#) ([evalml.pipelines.components.ensemble.sklearn\\_stacked\\_ensemble](#)  
[593](#) [static method](#)), 456  
[LinearRegressor](#) (class in [load\(\)](#) ([evalml.pipelines.components.ensemble.sklearn\\_stacked\\_ensemble](#)  
[evalml.pipelines.components.estimators.regressors.linear\\_regression](#) [static method](#)), 459  
[557](#) [load\(\)](#) ([evalml.pipelines.components.ensemble.sklearn\\_stacked\\_ensemble](#)  
[load\(\)](#) ([evalml.automl.automl\\_search.AutoMLSearch](#) [static method](#)), 462  
[static method](#)), 184 [load\(\)](#) ([evalml.pipelines.components.ensemble.SklearnStackedEnsembleL](#)  
[load\(\)](#) ([evalml.automl.AutoMLSearch](#) [static method](#)), [static method](#)), 465  
[194](#) [load\(\)](#) ([evalml.pipelines.components.ensemble.SklearnStackedEnsembleC](#)  
[load\(\)](#) ([evalml.AutoMLSearch](#) [static method](#)), 1265 [static method](#)), 467  
[load\(\)](#) ([evalml.pipelines.ARIMAREgressor](#) [static](#) [load\(\)](#) ([evalml.pipelines.components.ensemble.SklearnStackedEnsembleL](#)  
[method](#)), 1107 [static method](#)), 470  
[load\(\)](#) ([evalml.pipelines.binary\\_classification\\_pipeline.BinaryClassifier](#) ([evalml.pipelines.components.Estimator](#) [static](#)  
[static method](#)), 1059 [method](#)), 957  
[load\(\)](#) ([evalml.pipelines.CatBoostClassifier](#) [static](#) [load\(\)](#) ([evalml.pipelines.components.estimators.ARIMAREgressor](#)  
[method](#)), 1109 [static method](#)), 614  
[load\(\)](#) ([evalml.pipelines.CatBoostRegressor](#) [static](#) [load\(\)](#) ([evalml.pipelines.components.estimators.BaselineClassifier](#)  
[method](#)), 1112 [static method](#)), 616  
[load\(\)](#) ([evalml.pipelines.classification\\_pipeline.ClassificationPipeline](#) ([evalml.pipelines.components.estimators.BaselineRegressor](#)  
[static method](#)), 1065 [static method](#)), 618  
[load\(\)](#) ([evalml.pipelines.components.ARIMAREgressor](#) [load\(\)](#) ([evalml.pipelines.components.estimators.CatBoostClassifier](#)  
[static method](#)), 918 [static method](#)), 621  
[load\(\)](#) ([evalml.pipelines.components.BaselineClassifier](#) [load\(\)](#) ([evalml.pipelines.components.estimators.CatBoostRegressor](#)  
[static method](#)), 921 [static method](#)), 624  
[load\(\)](#) ([evalml.pipelines.components.BaselineRegressor](#) [load\(\)](#) ([evalml.pipelines.components.estimators.classifiers.baseline\\_class](#)  
[static method](#)), 923 [static method](#)), 473  
[load\(\)](#) ([evalml.pipelines.components.CatBoostClassifier](#) [load\(\)](#) ([evalml.pipelines.components.estimators.classifiers.BaselineClass](#)  
[static method](#)), 925 [static method](#)), 506  
[load\(\)](#) ([evalml.pipelines.components.CatBoostRegressor](#) [load\(\)](#) ([evalml.pipelines.components.estimators.classifiers.catboost\\_class](#)  
[static method](#)), 928 [static method](#)), 476  
[load\(\)](#) ([evalml.pipelines.components.component\\_base.ComponentBase](#) ([evalml.pipelines.components.estimators.classifiers.CatBoostClass](#)  
[static method](#)), 908 [static method](#)), 509  
[load\(\)](#) ([evalml.pipelines.components.ComponentBase](#) [load\(\)](#) ([evalml.pipelines.components.estimators.classifiers.decision\\_tree](#)  
[static method](#)), 930 [static method](#)), 479  
[load\(\)](#) ([evalml.pipelines.components.DateTimeFeaturizer](#) [load\(\)](#) ([evalml.pipelines.components.estimators.classifiers.DecisionTreeC](#)  
[static method](#)), 933 [static method](#)), 512  
[load\(\)](#) ([evalml.pipelines.components.DecisionTreeClassifier](#) [load\(\)](#) ([evalml.pipelines.components.estimators.classifiers.elasticnet\\_class](#)  
[static method](#)), 936 [static method](#)), 482  
[load\(\)](#) ([evalml.pipelines.components.DecisionTreeRegressor](#) [load\(\)](#) ([evalml.pipelines.components.estimators.classifiers.ElasticNetClass](#)  
[static method](#)), 939 [static method](#)), 515  
[load\(\)](#) ([evalml.pipelines.components.DelayedFeatureTransformer](#) ([evalml.pipelines.components.estimators.classifiers.et\\_classifier.E](#)  
[static method](#)), 941 [static method](#)), 485  
[load\(\)](#) ([evalml.pipelines.components.DFSTransformer](#) [load\(\)](#) ([evalml.pipelines.components.estimators.classifiers.ExtraTreesClass](#)  
[static method](#)), 943 [static method](#)), 518  
[load\(\)](#) ([evalml.pipelines.components.DropColumns](#) [load\(\)](#) ([evalml.pipelines.components.estimators.classifiers.kneighbors\\_cl](#)  
[static method](#)), 946 [static method](#)), 488  
[load\(\)](#) ([evalml.pipelines.components.DropNullColumns](#) [load\(\)](#) ([evalml.pipelines.components.estimators.classifiers.KNeighborsCl](#)  
[static method](#)), 948 [static method](#)), 521  
[load\(\)](#) ([evalml.pipelines.components.ElasticNetClassifier](#) [load\(\)](#) ([evalml.pipelines.components.estimators.classifiers.lightgbm\\_class](#)  
[static method](#)), 950 [static method](#)), 492  
[load\(\)](#) ([evalml.pipelines.components.ElasticNetRegressor](#) [load\(\)](#) ([evalml.pipelines.components.estimators.classifiers.LightGBMClass](#)  
[static method](#)), 953 [static method](#)), 524  
[load\(\)](#) ([evalml.pipelines.components.EmailFeaturizer](#) [load\(\)](#) ([evalml.pipelines.components.estimators.classifiers.logistic\\_regres](#)

static method), 495	static method), 578
load () (evalml.pipelines.components.estimators.classifiers.LogisticRegressionClassifier), 527	load () (evalml.pipelines.components.estimators.regressors.catboost_regressor), 543
load () (evalml.pipelines.components.estimators.classifiers.LogisticRegressionClassifier), 529	load () (evalml.pipelines.components.estimators.regressors.CatBoostRegressor), 581
load () (evalml.pipelines.components.estimators.classifiers.LogisticRegressionClassifier), 497	load () (evalml.pipelines.components.estimators.regressors.decision_tree_classifier), 547
load () (evalml.pipelines.components.estimators.classifiers.LogisticRegressionClassifier), 500	load () (evalml.pipelines.components.estimators.regressors.DecisionTreeRegressor), 584
load () (evalml.pipelines.components.estimators.classifiers.LogisticRegressionClassifier), 532	load () (evalml.pipelines.components.estimators.regressors.elasticnet_regressor), 549
load () (evalml.pipelines.components.estimators.classifiers.LogisticRegressionClassifier), 503	load () (evalml.pipelines.components.estimators.regressors.ElasticNetRegressor), 586
load () (evalml.pipelines.components.estimators.classifiers.LogisticRegressionClassifier), 534	load () (evalml.pipelines.components.estimators.regressors.et_regressor), 553
load () (evalml.pipelines.components.estimators.DecisionTreeClassifier), 627	load () (evalml.pipelines.components.estimators.regressors.ExtraTreesRegressor), 589
load () (evalml.pipelines.components.estimators.DecisionTreeRegressor), 630	load () (evalml.pipelines.components.estimators.regressors.lightgbm_regressor), 556
load () (evalml.pipelines.components.estimators.ElasticNetClassifier), 633	load () (evalml.pipelines.components.estimators.regressors.LightGBMRegressor), 592
load () (evalml.pipelines.components.estimators.ElasticNetRegressor), 635	load () (evalml.pipelines.components.estimators.regressors.linear_regressor), 559
load () (evalml.pipelines.components.estimators.Estimator), 638	load () (evalml.pipelines.components.estimators.regressors.LinearRegressor), 595
load () (evalml.pipelines.components.estimators.estimator.Estimator), 610	load () (evalml.pipelines.components.estimators.regressors.prophet_regressor), 562
load () (evalml.pipelines.components.estimators.ExtraTreesClassifier), 640	load () (evalml.pipelines.components.estimators.regressors.ProphetRegressor), 597
load () (evalml.pipelines.components.estimators.ExtraTreesRegressor), 643	load () (evalml.pipelines.components.estimators.regressors.RandomForestRegressor), 600
load () (evalml.pipelines.components.estimators.KNeighborsClassifier), 646	load () (evalml.pipelines.components.estimators.regressors.rf_regressor), 564
load () (evalml.pipelines.components.estimators.LightGBMClassifier), 649	load () (evalml.pipelines.components.estimators.regressors.svm_regressor), 567
load () (evalml.pipelines.components.estimators.LightGBMRegressor), 652	load () (evalml.pipelines.components.estimators.regressors.SVMRegressor), 602
load () (evalml.pipelines.components.estimators.LinearRegressor), 654	load () (evalml.pipelines.components.estimators.regressors.time_series_baseline_regressor), 570
load () (evalml.pipelines.components.estimators.LogisticRegressionClassifier), 657	load () (evalml.pipelines.components.estimators.regressors.TimeSeriesBaselineRegressor), 605
load () (evalml.pipelines.components.estimators.ProphetRegressor), 659	load () (evalml.pipelines.components.estimators.regressors.xgboost_regressor), 573
load () (evalml.pipelines.components.estimators.RandomForestClassifier), 662	load () (evalml.pipelines.components.estimators.regressors.XGBoostRegressor), 607
load () (evalml.pipelines.components.estimators.RandomForestRegressor), 664	load () (evalml.pipelines.components.estimators.SVMClassifier), 666
load () (evalml.pipelines.components.estimators.regressors.ArimaRegressor), 538	load () (evalml.pipelines.components.estimators.regressors.SVMRegressor), 669
load () (evalml.pipelines.components.estimators.regressors.ArimaRegressor), 576	load () (evalml.pipelines.components.estimators.TimeSeriesBaselineEstimator), 671
load () (evalml.pipelines.components.estimators.regressors.baseline_regressor), 540	load () (evalml.pipelines.components.estimators.XGBoostClassifier), 674
load () (evalml.pipelines.components.estimators.regressors.baseline_regressor), 540	load () (evalml.pipelines.components.estimators.XGBoostRegressor), 674

static method), 676

`load()` (`evalml.pipelines.components.ExtraTreesClassifier` static method), 960

`load()` (`evalml.pipelines.components.ExtraTreesRegressor` static method), 963

`load()` (`evalml.pipelines.components.FeatureSelector` static method), 965

`load()` (`evalml.pipelines.components.Imputer` static method), 968

`load()` (`evalml.pipelines.components.KNeighborsClassifier` static method), 970

`load()` (`evalml.pipelines.components.LightGBMClassifier` static method), 973

`load()` (`evalml.pipelines.components.LightGBMRegressor` static method), 976

`load()` (`evalml.pipelines.components.LinearDiscriminantAnalysis` static method), 978

`load()` (`evalml.pipelines.components.LinearRegressor` static method), 980

`load()` (`evalml.pipelines.components.LogisticRegressionClassifier` static method), 983

`load()` (`evalml.pipelines.components.LogTransformer` static method), 985

`load()` (`evalml.pipelines.components.LSA` static method), 987

`load()` (`evalml.pipelines.components.OneHotEncoder` static method), 990

`load()` (`evalml.pipelines.components.PCA` static method), 993

`load()` (`evalml.pipelines.components.PerColumnImputer` static method), 995

`load()` (`evalml.pipelines.components.PolynomialDetrender` static method), 997

`load()` (`evalml.pipelines.components.ProphetRegressor` static method), 999

`load()` (`evalml.pipelines.components.RandomForestClassifier` static method), 1002

`load()` (`evalml.pipelines.components.RandomForestRegressor` static method), 1004

`load()` (`evalml.pipelines.components.RFClassifierSelectFromModel` static method), 1007

`load()` (`evalml.pipelines.components.RFRegressorSelectFromModel` static method), 1010

`load()` (`evalml.pipelines.components.SelectByType` static method), 1012

`load()` (`evalml.pipelines.components.SelectColumns` static method), 1014

`load()` (`evalml.pipelines.components.SimpleImputer` static method), 1016

`load()` (`evalml.pipelines.components.SklearnStackedEnsembleClassifier` static method), 1018

`load()` (`evalml.pipelines.components.SklearnStackedEnsembleRegressor` static method), 1021

`load()` (`evalml.pipelines.components.SMOTENCOversampler` static method), 1023

`load()` (`evalml.pipelines.components.SMOTEOversampler` static method), 1026

`load()` (`evalml.pipelines.components.SMOTEOverSampler` static method), 1028

`load()` (`evalml.pipelines.components.StandardScaler` static method), 1030

`load()` (`evalml.pipelines.components.SVMClassifier` static method), 1032

`load()` (`evalml.pipelines.components.SVMRegressor` static method), 1034

`load()` (`evalml.pipelines.components.TargetEncoder` static method), 1037

`load()` (`evalml.pipelines.components.TargetImputer` static method), 1039

`load()` (`evalml.pipelines.components.TextFeaturizer` static method), 1041

`load()` (`evalml.pipelines.components.TimeSeriesBaselineEstimator` static method), 1044

`load()` (`evalml.pipelines.components.Transformer` static method), 1046

`load()` (`evalml.pipelines.components.transformers.column_selectors.ColumnSelector` static method), 824

`load()` (`evalml.pipelines.components.transformers.column_selectors.DropColumnSelector` static method), 827

`load()` (`evalml.pipelines.components.transformers.column_selectors.SelectColumnSelector` static method), 829

`load()` (`evalml.pipelines.components.transformers.column_selectors.SelectColumnsSelector` static method), 831

`load()` (`evalml.pipelines.components.transformers.DateTimeFeaturizer` static method), 839

`load()` (`evalml.pipelines.components.transformers.DelayedFeatureTransformer` static method), 842

`load()` (`evalml.pipelines.components.transformers.DFSTransformer` static method), 844

`load()` (`evalml.pipelines.components.transformers.dimensionality_reduction.ReducedDimPCA` static method), 679

`load()` (`evalml.pipelines.components.transformers.dimensionality_reduction.ReducedDimTSPCA` static method), 684

`load()` (`evalml.pipelines.components.transformers.dimensionality_reduction.ReducedDimTSPCA` static method), 686

`load()` (`evalml.pipelines.components.transformers.dimensionality_reduction.ReducedDimTSRCA` static method), 682

`load()` (`evalml.pipelines.components.transformers.DropColumns` static method), 846

`load()` (`evalml.pipelines.components.transformers.DropNullColumns` static method), 848

`load()` (`evalml.pipelines.components.transformers.EmailFeaturizer` static method), 850

`load()` (`evalml.pipelines.components.transformers.encoders.onehot_encoder.OneHotEncoder` static method), 690

`load()` (`evalml.pipelines.components.transformers.encoders.OneHotEncoder` static method), 697

`load()` (`evalml.pipelines.components.transformers.encoders.target_encoder.TargetEncoder` static method), 697





static method), 810

load() (evalml.pipelines.components.transformers.sampler.SMOTEOverSampler pipeline static method), 812

load() (evalml.pipelines.components.transformers.sampler.SMOTEOverSampler pipeline static method), 815

load() (evalml.pipelines.components.transformers.sampler.Undersampler pipeline static method), 817

load() (evalml.pipelines.components.transformers.sampler.Undersampler pipeline static method), 807

load() (evalml.pipelines.components.transformers.scalers.StandardScaler pipeline static method), 820

load() (evalml.pipelines.components.transformers.scalers.StandardScaler pipeline static method), 822

load() (evalml.pipelines.components.transformers.SelectByType() static method), 879

load() (evalml.pipelines.components.transformers.SelectColumns() static method), 881

load() (evalml.pipelines.components.transformers.SimpleImputer pipeline static method), 883

load() (evalml.pipelines.components.transformers.SMOTEOverSampler pipeline static method), 885

load() (evalml.pipelines.components.transformers.SMOTEOverSampler pipeline static method), 888

load() (evalml.pipelines.components.transformers.SMOTEOverSampler pipeline static method), 890

load() (evalml.pipelines.components.transformers.StandardScaler pipeline static method), 892

load() (evalml.pipelines.components.transformers.TargetEncoder pipeline static method), 895

load() (evalml.pipelines.components.transformers.TargetEncoder pipeline static method), 897

load() (evalml.pipelines.components.transformers.TextFeaturizer pipeline static method), 899

load() (evalml.pipelines.components.transformers.Transformer pipeline static method), 901

load() (evalml.pipelines.components.transformers.transformer.TargetTransformer pipeline static method), 833

load() (evalml.pipelines.components.transformers.transformer.TargetTransformer pipeline static method), 835

load() (evalml.pipelines.components.transformers.Undersampler pipeline static method), 904

load() (evalml.pipelines.components.transformers.URLFeaturizer pipeline static method), 906

load() (evalml.pipelines.components.Undersampler pipeline static method), 1048

load() (evalml.pipelines.components.URLFeaturizer pipeline static method), 1050

load() (evalml.pipelines.components.XGBoostClassifier pipeline static method), 1053

load() (evalml.pipelines.components.XGBoostRegressor pipeline static method), 1055

load() (evalml.pipelines.DecisionTreeClassifier static method), 1117

load() (evalml.pipelines.DecisionTreeRegressor static method), 1120

load() (evalml.pipelines.DelayedFeatureTransformer pipeline static method), 1123

load() (evalml.pipelines.DFSTransformer pipeline static method), 1125

load() (evalml.pipelines.ElasticNetClassifier pipeline static method), 1127

load() (evalml.pipelines.ElasticNetRegressor pipeline static method), 1130

load() (evalml.pipelines.ExtraTreesClassifier pipeline static method), 1132

load() (evalml.pipelines.ExtraTreesClassifier pipeline static method), 1135

load() (evalml.pipelines.ExtraTreesRegressor pipeline static method), 1138

load() (evalml.pipelines.FeatureSelector pipeline static method), 1140

load() (evalml.pipelines.KNeighborsClassifier pipeline static method), 1143

load() (evalml.pipelines.LightGBMClassifier pipeline static method), 1146

load() (evalml.pipelines.LightGBMRegressor pipeline static method), 1148

load() (evalml.pipelines.LinearRegressor pipeline static method), 1150

load() (evalml.pipelines.LogisticRegressionClassifier pipeline static method), 1153

load() (evalml.pipelines.multiclass\_classification\_pipeline.MulticlassClassifier pipeline static method), 1073

load() (evalml.pipelines.MulticlassClassificationPipeline pipeline static method), 1157

load() (evalml.pipelines.OneHotEncoder pipeline static method), 1160

load() (evalml.pipelines.PerColumnImputer pipeline static method), 1163

load() (evalml.pipelines.pipeline\_base.PipelineBase pipeline static method), 1077

load() (evalml.pipelines.PipelineBase pipeline static method), 1166

load() (evalml.pipelines.ProphetRegressor pipeline static method), 1169

load() (evalml.pipelines.RandomForestClassifier pipeline static method), 1171

load() (evalml.pipelines.RandomForestRegressor pipeline static method), 1173

load() (evalml.pipelines.regression\_pipeline.RegressionPipeline pipeline static method), 1083

load() (evalml.pipelines.RegressionPipeline pipeline static method), 1176

load() (evalml.pipelines.RFClassifierSelectFromModel pipeline static method), 1180

load() (evalml.pipelines.RFRegressorSelectFromModel pipeline static method), 1182

load() (evalml.pipelines.SimpleImputer pipeline static method), 1185

1184

`load()` (`evalml.pipelines.SklearnStackedEnsembleClassifier` static method), 1187

`load()` (`evalml.pipelines.SklearnStackedEnsembleRegressor` static method), 1189

`load()` (`evalml.pipelines.StandardScaler` static method), 1192

`load()` (`evalml.pipelines.SVMClassifier` static method), 1194

`load()` (`evalml.pipelines.SVMRegressor` static method), 1196

`load()` (`evalml.pipelines.TargetEncoder` static method), 1199

`load()` (`evalml.pipelines.time_series_classification_pipelines.TimeSeriesBinaryClassificationPipeline` static method), 1088

`load()` (`evalml.pipelines.time_series_classification_pipelines.TimeSeriesMulticlassClassificationPipeline` static method), 1092

`load()` (`evalml.pipelines.time_series_classification_pipelines.TimeSeriesRegressionPipeline` static method), 1096

`load()` (`evalml.pipelines.time_series_regression_pipeline.TimeSeriesRegressionPipeline` static method), 1101

`load()` (`evalml.pipelines.TimeSeriesBinaryClassificationPipeline` static method), 1202

`load()` (`evalml.pipelines.TimeSeriesClassificationPipeline` static method), 1207

`load()` (`evalml.pipelines.TimeSeriesMulticlassClassificationPipeline` static method), 1211

`load()` (`evalml.pipelines.TimeSeriesRegressionPipeline` static method), 1215

`load()` (`evalml.pipelines.Transformer` static method), 1218

`load()` (`evalml.pipelines.XGBoostClassifier` static method), 1220

`load()` (`evalml.pipelines.XGBoostRegressor` static method), 1222

`load_breast_cancer()` (in module `evalml.demos`), 248

`load_breast_cancer()` (in module `evalml.demos.breast_cancer`), 246

`load_churn()` (in module `evalml.demos`), 248

`load_churn()` (in module `evalml.demos.churn`), 246

`load_data()` (in module `evalml.preprocessing`), 1233

`load_data()` (in module `evalml.preprocessing.utils`), 1231

`load_diabetes()` (in module `evalml.demos`), 248

`load_diabetes()` (in module `evalml.demos.diabetes`), 247

`load_fraud()` (in module `evalml.demos`), 249

`load_fraud()` (in module `evalml.demos.fraud`), 247

`load_wine()` (in module `evalml.demos`), 249

`load_wine()` (in module `evalml.demos.wine`), 248

`log_error_callback()` (in module `evalml.automl.callbacks`), 187

`log_subtitle()` (in module `evalml.utils`), 1261

`log_subtitle()` (in module `evalml.utils.logger`), 1256

`log_title()` (in module `evalml.utils`), 1261

`log_title()` (in module `evalml.utils.logger`), 1256

`logger` (in module `evalml.automl.automl_search`), 186

`logger` (in module `evalml.automl.callbacks`), 187

`logger` (in module `evalml.data_checks.no_variance_data_check`), 217

`logger` (in module `evalml.objectives.sensitivity_low_alert`), 305

`logger` (in module `evalml.pipelines.component_graph`), 1069

`logger` (in module `evalml.pipelines.components.component_base`), 989

`LogLossBinary` (class in `evalml.objectives`), 410

`LogLossBinary` (class in `evalml.objectives.standard_metrics`), 333

`LogLossMulticlass` (class in `evalml.objectives`), 412

`LogLossMulticlass` (class in `evalml.objectives.standard_metrics`), 336

`LogTransformer` (class in `evalml.pipelines.components`), 984

`LogTransformer` (class in `evalml.pipelines.components.transformers`), 857

`LogTransformer` (class in `evalml.pipelines.components.transformers.preprocessing`), 779

`LogTransformer` (class in `evalml.pipelines.components.transformers.preprocessing.log_transformer`), 751

`LogisticRegressionClassifier` (class in `evalml.pipelines`), 1151

`LogisticRegressionClassifier` (class in `evalml.pipelines.components`), 981

`LogisticRegressionClassifier` (class in `evalml.pipelines.components.estimators`), 655

`LogisticRegressionClassifier` (class in `evalml.pipelines.components.estimators.classifiers`), 525

`LogisticRegressionClassifier` (class in `evalml.pipelines.components.estimators.classifiers.logistic_regression`), 493

LSA (class in `evalml.pipelines.components`), 986

LSA (class in `evalml.pipelines.components.transformers`), 860

LSA (class in `evalml.pipelines.components.transformers.preprocessing`), 782

LSA (class in `evalml.pipelines.components.transformers.preprocessing`), 753

## M

MAE (class in `evalml.objectives`), 413

MAE (class in `evalml.objectives.standard_metrics`), 337

`make_balancing_dictionary()` (in module `evalml.pipelines.components.utils`), 911

`make_data_splitter()` (in module `evalml.automl`), 196

`make_data_splitter()` (in module `evalml.automl.utils`), 190

`make_pipeline()` (in module `evalml.pipelines.utils`), 1103

MAPE (class in `evalml.objectives`), 415

MAPE (class in `evalml.objectives.standard_metrics`), 339

MaxError (class in `evalml.objectives`), 417

MaxError (class in `evalml.objectives.standard_metrics`), 340

MCCBinary (class in `evalml.objectives`), 418

MCCBinary (class in `evalml.objectives.standard_metrics`), 342

MCCMulticlass (class in `evalml.objectives`), 420

MCCMulticlass (class in `evalml.objectives.standard_metrics`), 344

MeanSquaredLogError (class in `evalml.objectives`), 422

MeanSquaredLogError (class in `evalml.objectives.standard_metrics`), 346

MedianAE (class in `evalml.objectives`), 424

MedianAE (class in `evalml.objectives.standard_metrics`), 347

`method` (in module `evalml.utils.update_checker`), 1257

`MethodPropertyNotFoundError`, 250, 252

`MissingComponentError`, 250, 252

`model_family()` (`evalml.pipelines.binary_classification_pipeline.BinaryClassificationPipeline` property), 1059

`model_family()` (`evalml.pipelines.classification_pipeline.ClassificationPipeline` property), 1065

`model_family()` (`evalml.pipelines.components.component_base.ComponentBase` property), 908

`model_family()` (`evalml.pipelines.components.ComponentBase` property), 930

`model_family()` (`evalml.pipelines.multiclass_classification_pipeline.MulticlassClassificationPipeline` property), 1073

`model_family()` (`evalml.pipelines.MulticlassClassificationPipeline` property), 1157

`model_family()` (`evalml.pipelines.pipeline_base.PipelineBase` property), 1077

`model_family()` (`evalml.pipelines.PipelineBase` property), 1166

`model_family()` (`evalml.pipelines.regression_pipeline.RegressionPipeline` property), 1083

`model_family()` (`evalml.pipelines.RegressionPipeline` property), 1177

`model_family()` (`evalml.pipelines.time_series_classification_pipelines` property), 1088

`model_family()` (`evalml.pipelines.time_series_classification_pipelines` property), 1092

`model_family()` (`evalml.pipelines.time_series_classification_pipelines` property), 1097

`model_family()` (`evalml.pipelines.time_series_regression_pipeline.TimeSeriesRegressionPipeline` property), 1101

`model_family()` (`evalml.pipelines.TimeSeriesBinaryClassificationPipeline` property), 1202

`model_family()` (`evalml.pipelines.TimeSeriesClassificationPipeline` property), 1207

`model_family()` (`evalml.pipelines.TimeSeriesMulticlassClassificationPipeline` property), 1211

`model_family()` (`evalml.pipelines.TimeSeriesRegressionPipeline` property), 1215

`ModelFamily` (class in `evalml.model_family`), 255

`ModelFamily` (class in `evalml.model_family.model_family`), 254

`modifies_features()` (`evalml.pipelines.components.component_base.ComponentBase` property), 908

`modifies_features()` (`evalml.pipelines.components.ComponentBase` property), 930

`modifies_target()` (`evalml.pipelines.components.component_base.ComponentBase` property), 908

`modifies_target()` (`evalml.pipelines.components.ComponentBase` property), 930

module

`evalml`, 156

`evalml.automl`, 156

`evalml.pipelines.binary_classification_pipeline.BinaryClassificationPipeline`, 156

`evalml.automl.automl_algorithm.automl_algorithm`, 156

`evalml.automl.automl_algorithm.evalml_algorithm`, 156

`evalml.automl.automl_algorithm.iterative_algorithm`, 156

`evalml.automl.automl_search`, 180

`evalml.automl.engine`, 166

`evalml.automl.engine.cf_engine`, 166

`evalml.automl.engine.dask_engine`, 166

`evalml.automl.engine.engine_base`, 166

[171](#)  
[evalml.automl.engine.sequential\\_engine,](#)  
[174](#)  
[evalml.automl.pipeline\\_search\\_plots,](#)  
[187](#)  
[evalml.automl.utils,](#) [188](#)  
[evalml.data\\_checks,](#) [198](#)  
[evalml.data\\_checks.class\\_imbalance\\_data\\_check,](#)  
[198](#)  
[evalml.data\\_checks.data\\_check,](#) [200](#)  
[evalml.data\\_checks.data\\_check\\_action,](#)  
[201](#)  
[evalml.data\\_checks.data\\_check\\_action\\_code,](#) [202](#)  
[202](#)  
[evalml.data\\_checks.data\\_check\\_message,](#)  
[202](#)  
[evalml.data\\_checks.data\\_check\\_message\\_code,](#) [204](#)  
[evalml.data\\_checks.data\\_check\\_message\\_type,](#) [206](#)  
[evalml.data\\_checks.data\\_checks,](#) [206](#)  
[evalml.data\\_checks.datetime\\_format\\_data\\_check,](#)  
[207](#)  
[evalml.data\\_checks.datetime\\_nan\\_data\\_check,](#) [208](#)  
[208](#)  
[evalml.data\\_checks.default\\_data\\_checks,](#) [210](#)  
[210](#)  
[evalml.data\\_checks.highly\\_null\\_data\\_check,](#) [211](#)  
[211](#)  
[evalml.data\\_checks.id\\_columns\\_data\\_check,](#) [213](#)  
[213](#)  
[evalml.data\\_checks.invalid\\_targets\\_data\\_check,](#)  
[214](#)  
[evalml.data\\_checks.multicollinearity\\_data\\_check,](#)  
[215](#)  
[evalml.data\\_checks.natural\\_language\\_nan\\_data\\_check,](#)  
[216](#)  
[evalml.data\\_checks.no\\_variance\\_data\\_check,](#) [217](#)  
[217](#)  
[evalml.data\\_checks.outliers\\_data\\_check,](#) [218](#)  
[218](#)  
[evalml.data\\_checks.sparsity\\_data\\_check,](#) [219](#)  
[219](#)  
[evalml.data\\_checks.target\\_distribution\\_data\\_check,](#)  
[221](#)  
[evalml.data\\_checks.target\\_leakage\\_data\\_check,](#) [222](#)  
[222](#)  
[evalml.data\\_checks.uniqueness\\_data\\_check,](#) [223](#)  
[223](#)  
[evalml.data\\_checks.utils,](#) [225](#)  
[evalml.demos,](#) [246](#)  
[evalml.demos.breast\\_cancer,](#) [246](#)  
[evalml.demos.churn,](#) [246](#)  
[evalml.demos.diabetes,](#) [247](#)  
[evalml.demos.fraud,](#) [247](#)  
[evalml.demos.wine,](#) [248](#)  
[evalml.exceptions,](#) [249](#)  
[evalml.exceptions.exceptions,](#) [249](#)  
[evalml.model\\_family,](#) [254](#)  
[evalml.model\\_family.model\\_family,](#)  
[254](#)  
[evalml.model\\_family.utils,](#) [255](#)  
[evalml.model\\_understanding,](#) [256](#)  
[evalml.model\\_understanding.force\\_plots,](#)  
[263](#)  
[evalml.model\\_understanding.graphs,](#)  
[264](#)  
[evalml.model\\_understanding.permutation\\_importance,](#)  
[274](#)  
[evalml.model\\_understanding.prediction\\_explanations,](#)  
[274](#)  
[evalml.model\\_understanding.prediction\\_explanations,](#)  
[274](#)  
[evalml.objectives,](#) [287](#)  
[evalml.objectives.binary\\_classification\\_objective,](#)  
[287](#)  
[evalml.objectives.cost\\_benefit\\_matrix,](#)  
[290](#)  
[evalml.objectives.fraud\\_cost,](#) [292](#)  
[evalml.objectives.lead\\_scoring,](#) [295](#)  
[evalml.objectives.multiclass\\_classification\\_objective,](#)  
[295](#)  
[evalml.objectives.objective\\_base,](#)  
[300](#)  
[evalml.objectives.regression\\_objective,](#)  
[302](#)  
[evalml.objectives.sensitivity\\_low\\_alert,](#)  
[305](#)  
[evalml.objectives.standard\\_metrics,](#)  
[307](#)  
[evalml.objectives.time\\_series\\_regression\\_objective,](#)  
[307](#)  
[evalml.objectives.utils,](#) [371](#)  
[evalml.pipelines,](#) [453](#)  
[evalml.pipelines.binary\\_classification\\_pipeline,](#)  
[1056](#)  
[evalml.pipelines.binary\\_classification\\_pipeline,](#)  
[1056](#)  
[evalml.pipelines.classification\\_pipeline,](#)  
[1062](#)  
[evalml.pipelines.component\\_graph,](#)  
[1066](#)  
[evalml.pipelines.components,](#) [453](#)  
[evalml.pipelines.components.component\\_base,](#)  
[907](#)  
[evalml.pipelines.components.component\\_base\\_meta,](#)  
[909](#)  
[evalml.pipelines.components.ensemble,](#)





[evalml.tuners, 1240](#)  
[evalml.pipelines.components.transformerensembles.grid\\_search\\_tuner, 753](#)  
[evalml.pipelines.components.transformerensembles.grid\\_search\\_tuner, 755](#)  
[evalml.pipelines.components.transformerensembles.grid\\_search\\_tuner, 758](#)  
[evalml.pipelines.components.transformerensembles.grid\\_search\\_tuner, 761](#)  
[evalml.pipelines.components.transformerensembles.grid\\_search\\_tuner, 763](#)  
[evalml.pipelines.components.transformerensembles.grid\\_search\\_tuner, 793](#)  
[evalml.pipelines.components.transformerensembles.grid\\_search\\_tuner, 793](#)  
[evalml.pipelines.components.transformerensembles.grid\\_search\\_tuner, 798](#)  
[evalml.pipelines.components.transformerensembles.grid\\_search\\_tuner, 805](#)  
[evalml.pipelines.components.transformerensembles.grid\\_search\\_tuner, 818](#)  
[evalml.pipelines.components.transformers.scalers.standard\\_scaler, 818](#)  
[evalml.pipelines.components.transformers.multiclass\\_classification\\_pipeline, 831](#)  
[evalml.pipelines.components.utils, 910](#)  
[evalml.pipelines.multiclass\\_classification\\_pipeline, 1070](#)  
[evalml.pipelines.pipeline\\_base, 1074](#)  
[evalml.pipelines.pipeline\\_meta, 1079](#)  
[evalml.pipelines.regression\\_pipeline, 1080](#)  
[evalml.pipelines.time\\_series\\_classification\\_pipelines, 1084](#)  
[evalml.pipelines.time\\_series\\_regression\\_pipeline, 1098](#)  
[evalml.pipelines.utils, 1102](#)  
[evalml.preprocessing, 1223](#)  
[evalml.preprocessing.data\\_splitters, 1223](#)  
[evalml.preprocessing.data\\_splitters.balanced\\_classification\\_sampler, 1223](#)  
[evalml.preprocessing.data\\_splitters.sampler, 1225](#)  
[evalml.preprocessing.data\\_splitters.time\\_series\\_sampler, 1225](#)  
[evalml.preprocessing.data\\_splitters.train\\_test\\_split, 1227](#)  
[evalml.preprocessing.utils, 1231](#)  
[evalml.problem\\_types, 1235](#)  
[evalml.problem\\_types.problem\\_types, 1235](#)  
[evalml.problem\\_types.utils, 1236](#)

**N**  
[name \(\) \(evalml.data\\_checks.class\\_imbalance\\_data\\_check.ClassImbalanceDataCheck method\), 199](#)  
[name \(\) \(evalml.data\\_checks.ClassImbalanceDataCheck method\), 227](#)  
[name \(\) \(evalml.data\\_checks.data\\_check.DataCheck method\), 201](#)  
[name \(\) \(evalml.data\\_checks.data\\_check\\_action\\_code.DataCheckActionCode method\), 202](#)  
[name \(\) \(evalml.data\\_checks.data\\_check\\_message\\_code.DataCheckMessageCode method\), 205](#)  
[name \(\) \(evalml.data\\_checks.data\\_check\\_message\\_type.DataCheckMessageType method\), 206](#)  
[name \(\) \(evalml.data\\_checks.DataCheck method\), 228](#)  
[name \(\) \(evalml.data\\_checks.DataCheckActionCode method\), 229](#)  
[name \(\) \(evalml.data\\_checks.DataCheckMessageCode method\), 231](#)  
[name \(\) \(evalml.data\\_checks.DataCheckMessageType method\), 232](#)



property), 852

name () (evalml.pipelines.components.transformers.preprocessing\_text\_transformer.TextTransformer (property), 763

name () (evalml.pipelines.components.transformers.preprocessing\_text\_transformer.TextTransformer (property), 790

name () (evalml.pipelines.components.transformers.sampler\_base\_sampler.BaseSampler (property), 795

name () (evalml.pipelines.components.transformers.sampler\_base\_sampler.BaseSampler (property), 797

name () (evalml.pipelines.components.transformers.Transformers\_fitting () (evalml.pipelines.components.BaselineClassifier (property), 901

name () (evalml.pipelines.components.transformers.transformer\_base\_transformer.BaseTransformer (property), 833

name () (evalml.pipelines.components.transformers.transformer\_base\_transformer.BaseTransformer (property), 835

name () (evalml.pipelines.Estimator property), 1132

name () (evalml.pipelines.FeatureSelector property), 1140

name () (evalml.pipelines.multiclass\_classification\_pipeline.MulticlassClassificationPipeline (property), 1073

name () (evalml.pipelines.MulticlassClassificationPipeline (property), 1157

name () (evalml.pipelines.pipeline\_base.PipelineBase (property), 1077

name () (evalml.pipelines.PipelineBase property), 1166

name () (evalml.pipelines.regression\_pipeline.RegressionPipeline (property), 1083

name () (evalml.pipelines.RegressionPipeline property), 1177

name () (evalml.pipelines.time\_series\_classification\_pipeline.TimeSeriesBinaryClassificationPipeline (property), 1088

name () (evalml.pipelines.time\_series\_classification\_pipeline.TimeSeriesClassificationPipeline (property), 1092

name () (evalml.pipelines.time\_series\_classification\_pipeline.TimeSeriesMulticlassClassificationPipeline (property), 1097

name () (evalml.pipelines.time\_series\_regression\_pipeline.TimeSeriesRegressionPipeline (property), 1101

name () (evalml.pipelines.TimeSeriesBinaryClassificationPipeline (property), 1202

name () (evalml.pipelines.TimeSeriesClassificationPipeline (property), 1207

name () (evalml.pipelines.TimeSeriesMulticlassClassificationPipeline (property), 1211

name () (evalml.pipelines.TimeSeriesRegressionPipeline (property), 1215

name () (evalml.pipelines.Transformer property), 1218

name () (evalml.problem\_types.problem\_types.ProblemTypes (method), 1236

name () (evalml.problem\_types.ProblemTypes (method), 1240

NaturalLanguageNaNDataCheck (class in evalml.data\_checks), 239

NaturalLanguageNaNDataCheck (class in evalml.data\_checks.natural\_language\_nan\_data\_checks), 239

216

method), 1107

method), 1109

method), 1112

method), 918

method), 921

method), 923

method), 926

needs\_fitting () (evalml.pipelines.components.CatBoostRegressor (method), 928

needs\_fitting () (evalml.pipelines.components.component\_base.ComponentBase (method), 930

needs\_fitting () (evalml.pipelines.components.ComponentBase (method), 933

needs\_fitting () (evalml.pipelines.components.DateTimeFeaturizer (method), 936

needs\_fitting () (evalml.pipelines.components.DecisionTreeClassifier (method), 939

needs\_fitting () (evalml.pipelines.components.DFSTransformer (method), 944

needs\_fitting () (evalml.pipelines.components.DropNullColumns (method), 948

needs\_fitting () (evalml.pipelines.components.ElasticNetClassifier (method), 950

needs\_fitting () (evalml.pipelines.components.ElasticNetRegressor (method), 953

needs\_fitting () (evalml.pipelines.components.EmailFeaturizer (method), 955

needs\_fitting () (evalml.pipelines.components.ensemble.sklearn\_stackensemble.Stackensemble (method), 456

needs\_fitting () (evalml.pipelines.components.ensemble.sklearn\_stackensemble.Stackensemble (method), 459

needs\_fitting () (evalml.pipelines.components.ensemble.sklearn\_stackensemble.Stackensemble (method), 462

needs\_fitting () (evalml.pipelines.components.ensemble.SklearnStackensemble (method), 465

needs\_fitting () (evalml.pipelines.components.ensemble.SklearnStackensemble (method), 468

needs\_fitting () (evalml.pipelines.components.ensemble.SklearnStackensemble (method), 470

needs\_fitting () (evalml.pipelines.components.Estimator (method), 957

needs\_fitting () (evalml.pipelines.components.estimators.ARIMARegressor (method), 614

needs\_fitting () (evalml.pipelines.components.estimators.BaselineClassifier (method), 614

method), 616  
needs\_fitting() (evalml.pipelines.components.estimators.BaseEstimator, 616  
method), 619  
needs\_fitting() (evalml.pipelines.components.estimators.CatBoostClassifier, 621  
method), 621  
needs\_fitting() (evalml.pipelines.components.estimators.CatBoostRegressor, 624  
method), 624  
needs\_fitting() (evalml.pipelines.components.estimators.classifiers.base, 473  
method), 473  
needs\_fitting() (evalml.pipelines.components.estimators.classifiers.BaseClassifier, 506  
method), 506  
needs\_fitting() (evalml.pipelines.components.estimators.classifiers.catboost, 476  
method), 476  
needs\_fitting() (evalml.pipelines.components.estimators.classifiers.CatBoostClassifier, 509  
method), 509  
needs\_fitting() (evalml.pipelines.components.estimators.classifiers.decision, 479  
method), 479  
needs\_fitting() (evalml.pipelines.components.estimators.classifiers.DecisionTreeClassifier, 512  
method), 512  
needs\_fitting() (evalml.pipelines.components.estimators.classifiers.elastic, 482  
method), 482  
needs\_fitting() (evalml.pipelines.components.estimators.classifiers.ElasticNetClassifier, 515  
method), 515  
needs\_fitting() (evalml.pipelines.components.estimators.classifiers.ensemble, 485  
method), 485  
needs\_fitting() (evalml.pipelines.components.estimators.classifiers.ExtraTreesClassifier, 518  
method), 518  
needs\_fitting() (evalml.pipelines.components.estimators.classifiers.kneighbors, 489  
method), 489  
needs\_fitting() (evalml.pipelines.components.estimators.classifiers.KNeighborsClassifier, 521  
method), 521  
needs\_fitting() (evalml.pipelines.components.estimators.classifiers.lightgbm, 492  
method), 492  
needs\_fitting() (evalml.pipelines.components.estimators.classifiers.LightGBMClassifier, 524  
method), 524  
needs\_fitting() (evalml.pipelines.components.estimators.classifiers.logistic, 495  
method), 495  
needs\_fitting() (evalml.pipelines.components.estimators.classifiers.LogisticRegressionClassifier, 527  
method), 527  
needs\_fitting() (evalml.pipelines.components.estimators.classifiers.RandomForestClassifier, 529  
method), 529  
needs\_fitting() (evalml.pipelines.components.estimators.classifiers.rf\_classifier, 498  
method), 498  
needs\_fitting() (evalml.pipelines.components.estimators.classifiers.svm\_classifier, 500  
method), 500  
needs\_fitting() (evalml.pipelines.components.estimators.classifiers.SVMClassifier, 532  
method), 532  
needs\_fitting() (evalml.pipelines.components.estimators.classifiers.xgboost, 504  
method), 504  
needs\_fitting() (evalml.pipelines.components.estimators.classifiers.XGBoostClassifier, 534  
method), 534  
needs\_fitting() (evalml.pipelines.components.estimators.DecisionTreeRegressor, 627  
method), 627  
needs\_fitting() (evalml.pipelines.components.estimators.DecisionTreeRegressor, 630  
method), 630  
needs\_fitting() (evalml.pipelines.components.estimators.ElasticNetRegressor, 633  
method), 633  
needs\_fitting() (evalml.pipelines.components.estimators.ElasticNetClassifier, 635  
method), 635  
needs\_fitting() (evalml.pipelines.components.estimators.Estimator, 638  
method), 638  
needs\_fitting() (evalml.pipelines.components.estimators.classifiers.base, 610  
method), 610  
needs\_fitting() (evalml.pipelines.components.estimators.classifiers.BaseClassifier, 641  
method), 641  
needs\_fitting() (evalml.pipelines.components.estimators.classifiers.catboost, 643  
method), 643  
needs\_fitting() (evalml.pipelines.components.estimators.classifiers.CatBoostClassifier, 646  
method), 646  
needs\_fitting() (evalml.pipelines.components.estimators.classifiers.decision, 649  
method), 649  
needs\_fitting() (evalml.pipelines.components.estimators.classifiers.DecisionTreeClassifier, 652  
method), 652  
needs\_fitting() (evalml.pipelines.components.estimators.classifiers.elastic, 654  
method), 654  
needs\_fitting() (evalml.pipelines.components.estimators.classifiers.ElasticNetClassifier, 657  
method), 657  
needs\_fitting() (evalml.pipelines.components.estimators.classifiers.ensemble, 660  
method), 660  
needs\_fitting() (evalml.pipelines.components.estimators.classifiers.ExtraTreesClassifier, 662  
method), 662  
needs\_fitting() (evalml.pipelines.components.estimators.classifiers.kneighbors, 664  
method), 664  
needs\_fitting() (evalml.pipelines.components.estimators.classifiers.KNeighborsClassifier, 538  
method), 538  
needs\_fitting() (evalml.pipelines.components.estimators.classifiers.lightgbm, 576  
method), 576  
needs\_fitting() (evalml.pipelines.components.estimators.classifiers.LightGBMClassifier, 541  
method), 541  
needs\_fitting() (evalml.pipelines.components.estimators.classifiers.logistic, 579  
method), 579  
needs\_fitting() (evalml.pipelines.components.estimators.classifiers.LogisticRegressionClassifier, 544  
method), 544  
needs\_fitting() (evalml.pipelines.components.estimators.classifiers.RandomForestClassifier, 581  
method), 581  
needs\_fitting() (evalml.pipelines.components.estimators.classifiers.rf\_classifier, 547  
method), 547  
needs\_fitting() (evalml.pipelines.components.estimators.classifiers.svm\_classifier, 584  
method), 584  
needs\_fitting() (evalml.pipelines.components.estimators.classifiers.SVMClassifier, 550  
method), 550  
needs\_fitting() (evalml.pipelines.components.estimators.classifiers.xgboost, 587  
method), 587  
needs\_fitting() (evalml.pipelines.components.estimators.classifiers.XGBoostClassifier, 553  
method), 553  
needs\_fitting() (evalml.pipelines.components.estimators.DecisionTreeRegressor, 589  
method), 589  
needs\_fitting() (evalml.pipelines.components.estimators.DecisionTreeRegressor, 630  
method), 630



method), 556

needs\_fitting() (evalml.pipelines.components.estimators.regressors.LightGBMRegressor method), 592

needs\_fitting() (evalml.pipelines.components.estimators.regressors.linear\_model.LogisticRegressionClassifier method), 559

needs\_fitting() (evalml.pipelines.components.estimators.regressors.linear\_model.LogTransformer method), 595

needs\_fitting() (evalml.pipelines.components.estimators.regressors.linear\_model.LSA method), 595

needs\_fitting() (evalml.pipelines.components.estimators.regressors.prophet.ProphetRegressor method), 562

needs\_fitting() (evalml.pipelines.components.estimators.regressors.ProphetRegressor method), 598

needs\_fitting() (evalml.pipelines.components.estimators.regressors.RandomForestRegressor method), 600

needs\_fitting() (evalml.pipelines.components.estimators.regressors.iforest.IforestRegressor method), 565

needs\_fitting() (evalml.pipelines.components.estimators.regressors.linear\_model.SVMRegressor method), 567

needs\_fitting() (evalml.pipelines.components.estimators.regressors.SVMRegressor method), 602

needs\_fitting() (evalml.pipelines.components.estimators.regressors.linear\_model.XGBoostRegressor method), 570

needs\_fitting() (evalml.pipelines.components.estimators.regressors.TimeSeriesBaselineEstimator method), 605

needs\_fitting() (evalml.pipelines.components.estimators.regressors.iglobst.IGlobstRegressor method), 573

needs\_fitting() (evalml.pipelines.components.estimators.regressors.XGBoostRegressor method), 607

needs\_fitting() (evalml.pipelines.components.estimators.SVMClassifier method), 667

needs\_fitting() (evalml.pipelines.components.estimators.SVMRegressor method), 669

needs\_fitting() (evalml.pipelines.components.estimators.TimeSeriesBaseLineEstimator method), 671

needs\_fitting() (evalml.pipelines.components.estimators.XGBoostClassifier method), 674

needs\_fitting() (evalml.pipelines.components.estimators.XGBoostRegressor method), 676

needs\_fitting() (evalml.pipelines.components.ExtraTreesClassifier method), 960

needs\_fitting() (evalml.pipelines.components.ExtraTreesRegressor method), 963

needs\_fitting() (evalml.pipelines.components.FeatureSelector method), 965

needs\_fitting() (evalml.pipelines.components.Imputer method), 968

needs\_fitting() (evalml.pipelines.components.KNeighborsClassifier method), 970

needs\_fitting() (evalml.pipelines.components.LightGBMClassifier method), 973

needs\_fitting() (evalml.pipelines.components.LightGBMRegressor method), 976

needs\_fitting() (evalml.pipelines.components.LinearDiscriminantAnalysis method), 978

needs\_fitting() (evalml.pipelines.components.LinearRegressor method), 980

needs\_fitting() (evalml.pipelines.components.LogisticRegressionClassifier method), 983

needs\_fitting() (evalml.pipelines.components.LogTransformer method), 985

needs\_fitting() (evalml.pipelines.components.LSA method), 987

needs\_fitting() (evalml.pipelines.components.OneHotEncoder method), 990

needs\_fitting() (evalml.pipelines.components.PCA method), 993

needs\_fitting() (evalml.pipelines.components.PerColumnImputer method), 995

needs\_fitting() (evalml.pipelines.components.PolynomialDetrender method), 997

needs\_fitting() (evalml.pipelines.components.ProphetRegressor method), 1000

needs\_fitting() (evalml.pipelines.components.RandomForestClassifier method), 1002

needs\_fitting() (evalml.pipelines.components.linear\_model.SVMClassifier method), 1004

needs\_fitting() (evalml.pipelines.components.RFClassifierSelectFromModel method), 1007

needs\_fitting() (evalml.pipelines.components.RFRegressorSelectFromModel method), 1010

needs\_fitting() (evalml.pipelines.components.SimpleImputer method), 1016

needs\_fitting() (evalml.pipelines.components.SklearnStackedEnsembleClassifier method), 1019

needs\_fitting() (evalml.pipelines.components.SklearnStackedEnsembleRegressor method), 1021

needs\_fitting() (evalml.pipelines.components.SMOTEOverSampler method), 1024

needs\_fitting() (evalml.pipelines.components.SMOTEOverSampler method), 1026

needs\_fitting() (evalml.pipelines.components.SMOTEOverSampler method), 1028

needs\_fitting() (evalml.pipelines.components.StandardScaler method), 1030

needs\_fitting() (evalml.pipelines.components.SVMClassifier method), 1032

needs\_fitting() (evalml.pipelines.components.SVMRegressor method), 1035

needs\_fitting() (evalml.pipelines.components.TargetEncoder method), 1037

needs\_fitting() (evalml.pipelines.components.TargetImputer method), 1039

needs\_fitting() (evalml.pipelines.components.TextFeaturizer method), 1041

needs\_fitting() (evalml.pipelines.components.TimeSeriesBaselineEstimator method), 1044

needs\_fitting() (evalml.pipelines.components.Transformer method), 1046

needs\_fitting() (evalml.pipelines.components.transformers.column\_

1387

[illegible]



[method](#)), 1185  
[needs\\_fitting\(\)](#) ([evalml.pipelines.SklearnStackedEnsembleClassifier](#) method), 159  
[method](#)), 1187  
[needs\\_fitting\(\)](#) ([evalml.pipelines.SklearnStackedEnsembleRegressor](#) method), 164  
[method](#)), 1190  
[needs\\_fitting\(\)](#) ([evalml.pipelines.StandardScaler](#) method), 1192  
[needs\\_fitting\(\)](#) ([evalml.pipelines.SVMClassifier](#) method), 1194  
[needs\\_fitting\(\)](#) ([evalml.pipelines.SVMRegressor](#) method), 1196  
[needs\\_fitting\(\)](#) ([evalml.pipelines.TargetEncoder](#) method), 1199  
[needs\\_fitting\(\)](#) ([evalml.pipelines.Transformer](#) method), 1218  
[needs\\_fitting\(\)](#) ([evalml.pipelines.XGBoostClassifier](#) method), 1220  
[needs\\_fitting\(\)](#) ([evalml.pipelines.XGBoostRegressor](#) method), 1223  
[new\(\)](#) ([evalml.pipelines.binary\\_classification\\_pipeline.BinaryClassificationPipeline](#) method), 1059  
[new\(\)](#) ([evalml.pipelines.classification\\_pipeline.ClassificationPipeline](#) method), 1065  
[new\(\)](#) ([evalml.pipelines.multiclass\\_classification\\_pipeline.MulticlassClassificationPipeline](#) method), 1073  
[new\(\)](#) ([evalml.pipelines.MulticlassClassificationPipeline](#) method), 1157  
[new\(\)](#) ([evalml.pipelines.pipeline\\_base.PipelineBase](#) method), 1077  
[new\(\)](#) ([evalml.pipelines.PipelineBase](#) method), 1166  
[new\(\)](#) ([evalml.pipelines.regression\\_pipeline.RegressionPipeline](#) method), 1083  
[new\(\)](#) ([evalml.pipelines.RegressionPipeline](#) method), 1177  
[new\(\)](#) ([evalml.pipelines.time\\_series\\_classification\\_pipelines.TimeSeriesBinaryClassificationPipeline](#) method), 1088  
[new\(\)](#) ([evalml.pipelines.time\\_series\\_classification\\_pipelines.TimeSeriesClassificationPipeline](#) method), 1092  
[new\(\)](#) ([evalml.pipelines.time\\_series\\_classification\\_pipelines.TimeSeriesMulticlassClassificationPipeline](#) method), 1097  
[new\(\)](#) ([evalml.pipelines.time\\_series\\_regression\\_pipeline.TimeSeriesRegressionPipeline](#) method), 1101  
[new\(\)](#) ([evalml.pipelines.TimeSeriesBinaryClassificationPipeline](#) method), 1202  
[new\(\)](#) ([evalml.pipelines.TimeSeriesClassificationPipeline](#) method), 1207  
[new\(\)](#) ([evalml.pipelines.TimeSeriesMulticlassClassificationPipeline](#) method), 1211  
[new\(\)](#) ([evalml.pipelines.TimeSeriesRegressionPipeline](#) method), 1215  
[next\\_batch\(\)](#) ([evalml.automl.automl\\_algorithm.automl\\_algorithm.AutoMLAlgorithm](#) method), 157  
[next\\_batch\(\)](#) ([evalml.automl.automl\\_algorithm.AutoMLAlgorithm](#) method), 163  
[next\\_batch\(\)](#) ([evalml.automl.automl\\_algorithm.evalml\\_algorithm.EvalMLAlgorithm](#) method), 159  
[next\\_batch\(\)](#) ([evalml.automl.automl\\_algorithm.EvalMLAlgorithm](#) method), 164  
[next\\_batch\(\)](#) ([evalml.automl.automl\\_algorithm.iterative\\_algorithm.IterativeAlgorithm](#) method), 161  
[next\\_batch\(\)](#) ([evalml.automl.automl\\_algorithm.IterativeAlgorithm](#) method), 165  
[NoParamsException](#), 1246, 1247  
[NoPositiveLabelException](#), 250, 252  
[normalize\\_confusion\\_matrix\(\)](#) (in module [evalml.model\\_understanding](#)), 284  
[normalize\\_confusion\\_matrix\(\)](#) (in module [evalml.model\\_understanding.graphs](#)), 270  
[NoVarianceDataCheck](#) (class in [evalml.data\\_checks](#)), 240  
[NoVarianceDataCheck](#) (class in [evalml.data\\_checks.no\\_variance\\_data\\_check](#)), 217  
[NumberFeaturesException](#), 250, 252  
[number\\_of\\_features\(\)](#) (in module [evalml.preprocessing](#)), 1233  
[number\\_of\\_features\(\)](#) (in module [evalml.preprocessing.feature\\_utils](#)), 1231  
[numeric\\_and\\_boolean\\_wv](#) (in module [evalml.utils.woodwork\\_utils](#)), 1257

## O

[objective\\_function\(\)](#) ([evalml.objectives.AccuracyBinary](#) method), 375  
[objective\\_function\(\)](#) ([evalml.objectives.AccuracyMulticlass](#) method), 377  
[objective\\_function\(\)](#) ([evalml.objectives.AUC](#) method), 379  
[objective\\_function\(\)](#) ([evalml.objectives.AUCMacro](#) method), 381  
[objective\\_function\(\)](#) ([evalml.objectives.AUCMicro](#) method), 382  
[objective\\_function\(\)](#) ([evalml.objectives.AUCWeighted](#) method), 384  
[objective\\_function\(\)](#) ([evalml.objectives.BalancedAccuracyBinary](#) method), 386  
[objective\\_function\(\)](#) ([evalml.objectives.BalancedAccuracyMulticlass](#) method), 388  
[objective\\_function\(\)](#) ([evalml.objectives.BinaryClassificationObjective](#) method), 288

[class method](#)), 390  
[objective\\_function\(\)](#) ([evalml.objectives.cost\\_benefit\\_matrix.CostBenefitMatrix](#) [method](#)), 291  
[objective\\_function\(\)](#) ([evalml.objectives.CostBenefitMatrix](#) [method](#)), 392  
[objective\\_function\(\)](#) ([evalml.objectives.ExpVariance](#) [method](#)), 394  
[objective\\_function\(\)](#) ([evalml.objectives.F1](#) [method](#)), 396  
[objective\\_function\(\)](#) ([evalml.objectives.F1Macro](#) [method](#)), 398  
[objective\\_function\(\)](#) ([evalml.objectives.F1Micro](#) [method](#)), 400  
[objective\\_function\(\)](#) ([evalml.objectives.F1Weighted](#) [method](#)), 401  
[objective\\_function\(\)](#) ([evalml.objectives.fraud\\_cost.FraudCost](#) [method](#)), 294  
[objective\\_function\(\)](#) ([evalml.objectives.FraudCost](#) [method](#)), 403  
[objective\\_function\(\)](#) ([evalml.objectives.Gini](#) [method](#)), 406  
[objective\\_function\(\)](#) ([evalml.objectives.lead\\_scoring.LeadScoring](#) [method](#)), 297  
[objective\\_function\(\)](#) ([evalml.objectives.LeadScoring](#) [method](#)), 409  
[objective\\_function\(\)](#) ([evalml.objectives.LogLossBinary](#) [method](#)), 411  
[objective\\_function\(\)](#) ([evalml.objectives.LogLossMulticlass](#) [method](#)), 413  
[objective\\_function\(\)](#) ([evalml.objectives.MAE](#) [method](#)), 414  
[objective\\_function\(\)](#) ([evalml.objectives.MAPE](#) [method](#)), 416  
[objective\\_function\(\)](#) ([evalml.objectives.MaxError](#) [method](#)), 417  
[objective\\_function\(\)](#) ([evalml.objectives.MCCBinary](#) [method](#)), 419  
[objective\\_function\(\)](#) ([evalml.objectives.MCCMulticlass](#) [method](#)), 421  
[objective\\_function\(\)](#) ([evalml.objectives.MeanSquaredLogError](#) [method](#)), 423  
[objective\\_function\(\)](#) ([evalml.objectives.MedianAE](#) [method](#)), 424  
[objective\\_function\(\)](#) ([evalml.objectives.MSE](#) [method](#)), 426  
[objective\\_function\(\)](#) ([evalml.objectives.multiclass\\_classification\\_objective.MulticlassC](#) [class method](#)), 299  
[objective\\_function\(\)](#) ([evalml.objectives.MulticlassClassificationObjective](#) [class method](#)), 428  
[objective\\_function\(\)](#) ([evalml.objectives.objective\\_base.ObjectiveBase](#) [class method](#)), 301  
[objective\\_function\(\)](#) ([evalml.objectives.ObjectiveBase](#) [class method](#)), 430  
[objective\\_function\(\)](#) ([evalml.objectives.Precision](#) [method](#)), 432  
[objective\\_function\(\)](#) ([evalml.objectives.PrecisionMacro](#) [method](#)), 434  
[objective\\_function\(\)](#) ([evalml.objectives.PrecisionMicro](#) [method](#)), 435  
[objective\\_function\(\)](#) ([evalml.objectives.PrecisionWeighted](#) [method](#)), 437  
[objective\\_function\(\)](#) ([evalml.objectives.R2](#) [method](#)), 438  
[objective\\_function\(\)](#) ([evalml.objectives.Recall](#) [method](#)), 440  
[objective\\_function\(\)](#) ([evalml.objectives.RecallMacro](#) [method](#)), 442  
[objective\\_function\(\)](#) ([evalml.objectives.RecallMicro](#) [method](#)), 444  
[objective\\_function\(\)](#) ([evalml.objectives.RecallWeighted](#) [method](#)), 445  
[objective\\_function\(\)](#) ([evalml.objectives.regression\\_objective.RegressionObjective](#) [class method](#)), 304  
[objective\\_function\(\)](#) ([evalml.objectives.RegressionObjective](#) [class method](#)), 447  
[objective\\_function\(\)](#) ([evalml.objectives.RootMeanSquaredError](#) [method](#)), 449  
[objective\\_function\(\)](#) ([evalml.objectives.RootMeanSquaredLogError](#) [method](#)), 451  
[objective\\_function\(\)](#) ([evalml.objectives.sensitivity\\_low\\_alert.SensitivityLowAlert](#) [method](#)), 306

<code>objective_function()</code> ( <code>evalml.objectives.SensitivityLowAlert</code> <code>method</code> ), 452	<code>objective_function()</code> ( <code>evalml.objectives.standard_metrics.MAPE</code> <code>method</code> ), 340
<code>objective_function()</code> ( <code>evalml.objectives.standard_metrics.AccuracyBinary</code> <code>method</code> ), 309	<code>objective_function()</code> ( <code>evalml.objectives.standard_metrics.MaxError</code> <code>method</code> ), 341
<code>objective_function()</code> ( <code>evalml.objectives.standard_metrics.AccuracyMulticlass</code> <code>method</code> ), 311	<code>objective_function()</code> ( <code>evalml.objectives.standard_metrics.MCCBinary</code> <code>method</code> ), 343
<code>objective_function()</code> ( <code>evalml.objectives.standard_metrics.AUC</code> <code>method</code> ), 313	<code>objective_function()</code> ( <code>evalml.objectives.standard_metrics.MCCMulticlass</code> <code>method</code> ), 345
<code>objective_function()</code> ( <code>evalml.objectives.standard_metrics.AUCMacro</code> <code>method</code> ), 315	<code>objective_function()</code> ( <code>evalml.objectives.standard_metrics.MeanSquaredLogError</code> <code>method</code> ), 346
<code>objective_function()</code> ( <code>evalml.objectives.standard_metrics.AUCMicro</code> <code>method</code> ), 317	<code>objective_function()</code> ( <code>evalml.objectives.standard_metrics.MedianAE</code> <code>method</code> ), 348
<code>objective_function()</code> ( <code>evalml.objectives.standard_metrics.AUCWeighted</code> <code>method</code> ), 318	<code>objective_function()</code> ( <code>evalml.objectives.standard_metrics.MSE</code> <code>method</code> ), 350
<code>objective_function()</code> ( <code>evalml.objectives.standard_metrics.BalancedAccuracyBinary</code> <code>method</code> ), 320	<code>objective_function()</code> ( <code>evalml.objectives.standard_metrics.Precision</code> <code>method</code> ), 352
<code>objective_function()</code> ( <code>evalml.objectives.standard_metrics.BalancedAccuracyMulticlass</code> <code>method</code> ), 322	<code>objective_function()</code> ( <code>evalml.objectives.standard_metrics.PrecisionMacro</code> <code>method</code> ), 353
<code>objective_function()</code> ( <code>evalml.objectives.standard_metrics.ExpVariance</code> <code>method</code> ), 324	<code>objective_function()</code> ( <code>evalml.objectives.standard_metrics.PrecisionMicro</code> <code>method</code> ), 355
<code>objective_function()</code> ( <code>evalml.objectives.standard_metrics.F1</code> <code>method</code> ), 326	<code>objective_function()</code> ( <code>evalml.objectives.standard_metrics.PrecisionWeighted</code> <code>method</code> ), 357
<code>objective_function()</code> ( <code>evalml.objectives.standard_metrics.F1Macro</code> <code>method</code> ), 327	<code>objective_function()</code> ( <code>evalml.objectives.standard_metrics.R2</code> <code>method</code> ), 358
<code>objective_function()</code> ( <code>evalml.objectives.standard_metrics.F1Micro</code> <code>method</code> ), 329	<code>objective_function()</code> ( <code>evalml.objectives.standard_metrics.Recall</code> <code>method</code> ), 360
<code>objective_function()</code> ( <code>evalml.objectives.standard_metrics.F1Weighted</code> <code>method</code> ), 331	<code>objective_function()</code> ( <code>evalml.objectives.standard_metrics.RecallMacro</code> <code>method</code> ), 362
<code>objective_function()</code> ( <code>evalml.objectives.standard_metrics.Gini</code> <code>method</code> ), 332	<code>objective_function()</code> ( <code>evalml.objectives.standard_metrics.RecallMicro</code> <code>method</code> ), 363
<code>objective_function()</code> ( <code>evalml.objectives.standard_metrics.LogLossBinary</code> <code>method</code> ), 335	<code>objective_function()</code> ( <code>evalml.objectives.standard_metrics.RecallWeighted</code> <code>method</code> ), 365
<code>objective_function()</code> ( <code>evalml.objectives.standard_metrics.LogLossMulticlass</code> <code>method</code> ), 336	<code>objective_function()</code> ( <code>evalml.objectives.standard_metrics.RootMeanSquaredError</code> <code>method</code> ), 366
<code>objective_function()</code> ( <code>evalml.objectives.standard_metrics.MAE</code> <code>method</code> ), 338	<code>objective_function()</code> ( <code>evalml.objectives.standard_metrics.RootMeanSquaredLogError</code> <code>method</code> ), 368

`objective_function()` *method*, 297  
`(evalml.objectives.time_series_regression_objective.TimeSeriesRegressionObjective class method)`, 370  
`ObjectiveBase` (class in `evalml.objectives`), 429  
`ObjectiveBase` (class in `evalml.objectives.objective_base`), 300  
`ObjectiveCreationError`, 250, 252  
`ObjectiveNotFoundError`, 250, 252  
`OneHotEncoder` (class in `evalml.pipelines`), 1158  
`OneHotEncoder` (class in `evalml.pipelines.components`), 988  
`OneHotEncoder` (class in `evalml.pipelines.components.transformers`), 862  
`OneHotEncoder` (class in `evalml.pipelines.components.transformers.encoders`), 695  
`OneHotEncoder` (class in `evalml.pipelines.components.transformers.encoders.onehot`), 688  
`OneHotEncoderMeta` (class in `evalml.pipelines.components.transformers.encoders.onehot`), 691  
`optimize_threshold()` *method*, 375  
`optimize_threshold()` *method*, 379  
`optimize_threshold()` *method*, 386  
`optimize_threshold()` *method*, 289  
`optimize_threshold()` *method*, 390  
`optimize_threshold()` *method*, 292  
`optimize_threshold()` *method*, 393  
`optimize_threshold()` *method*, 396  
`optimize_threshold()` *method*, 294  
`optimize_threshold()` *method*, 404  
`optimize_threshold()` *method*, 407  
`optimize_threshold()` *method*, 441  
`optimize_threshold()` *method*, 420  
`optimize_threshold()` *method*, 432  
`optimize_threshold()` *method*, 441  
`optimize_threshold()` *method*, 306  
`optimize_threshold()` *method*, 453  
`optimize_threshold()` *method*, 310  
`optimize_threshold()` *method*, 313  
`optimize_threshold()` *method*, 320  
`optimize_threshold()` *method*, 326  
`optimize_threshold()` *method*, 333  
`optimize_threshold()` *method*, 335  
`optimize_threshold()` *method*, 343  
`optimize_threshold()` *method*, 352  
`optimize_threshold()` *method*, 360  
`optimize_threshold()` *method*, 1060  
`optimize_threshold()` *method*, 1061  
`optimize_threshold()` *method*, 407

Index 1393



`parameters()` (`evalml.pipelines.components.estimators.classifiers.KNeighborsClassifier`), 521  
`parameters()` (`evalml.pipelines.components.estimators.classifiers.KNeighborsClassifier`), 538  
`parameters()` (`evalml.pipelines.components.estimators.classifiers.LightGBMClassifier`), 492  
`parameters()` (`evalml.pipelines.components.estimators.classifiers.LightGBMClassifier`), 576  
`parameters()` (`evalml.pipelines.components.estimators.classifiers.LightGBMClassifier`), 524  
`parameters()` (`evalml.pipelines.components.estimators.classifiers.LightGBMClassifier`), 541  
`parameters()` (`evalml.pipelines.components.estimators.classifiers.LogisticRegressionClassifier`), 495  
`parameters()` (`evalml.pipelines.components.estimators.classifiers.LogisticRegressionClassifier`), 579  
`parameters()` (`evalml.pipelines.components.estimators.classifiers.LogisticRegressionClassifier`), 527  
`parameters()` (`evalml.pipelines.components.estimators.classifiers.LogisticRegressionClassifier`), 544  
`parameters()` (`evalml.pipelines.components.estimators.classifiers.RandomForestClassifier`), 529  
`parameters()` (`evalml.pipelines.components.estimators.classifiers.RandomForestClassifier`), 581  
`parameters()` (`evalml.pipelines.components.estimators.classifiers.RandomForestClassifier`), 498  
`parameters()` (`evalml.pipelines.components.estimators.classifiers.RandomForestClassifier`), 547  
`parameters()` (`evalml.pipelines.components.estimators.classifiers.SVCClassifier`), 501  
`parameters()` (`evalml.pipelines.components.estimators.classifiers.SVCClassifier`), 584  
`parameters()` (`evalml.pipelines.components.estimators.classifiers.SVMClassifier`), 532  
`parameters()` (`evalml.pipelines.components.estimators.classifiers.SVMClassifier`), 550  
`parameters()` (`evalml.pipelines.components.estimators.classifiers.XGBoostClassifier`), 504  
`parameters()` (`evalml.pipelines.components.estimators.classifiers.XGBoostClassifier`), 587  
`parameters()` (`evalml.pipelines.components.estimators.classifiers.XGBoostClassifier`), 535  
`parameters()` (`evalml.pipelines.components.estimators.classifiers.XGBoostClassifier`), 553  
`parameters()` (`evalml.pipelines.components.estimators.DecisionTreeClassifier`), 627  
`parameters()` (`evalml.pipelines.components.estimators.DecisionTreeClassifier`), 590  
`parameters()` (`evalml.pipelines.components.estimators.DecisionTreeRegressor`), 630  
`parameters()` (`evalml.pipelines.components.estimators.DecisionTreeRegressor`), 556  
`parameters()` (`evalml.pipelines.components.estimators.ElasticNetClassifier`), 633  
`parameters()` (`evalml.pipelines.components.estimators.ElasticNetClassifier`), 593  
`parameters()` (`evalml.pipelines.components.estimators.ElasticNetRegressor`), 635  
`parameters()` (`evalml.pipelines.components.estimators.ElasticNetRegressor`), 559  
`parameters()` (`evalml.pipelines.components.estimators.Estimator`), 638  
`parameters()` (`evalml.pipelines.components.estimators.Estimator`), 595  
`parameters()` (`evalml.pipelines.components.estimators.estimator.Estimator`), 610  
`parameters()` (`evalml.pipelines.components.estimators.estimator.Estimator`), 562  
`parameters()` (`evalml.pipelines.components.estimators.ExtraTreesClassifier`), 641  
`parameters()` (`evalml.pipelines.components.estimators.ExtraTreesClassifier`), 598  
`parameters()` (`evalml.pipelines.components.estimators.ExtraTreesRegressor`), 644  
`parameters()` (`evalml.pipelines.components.estimators.ExtraTreesRegressor`), 600  
`parameters()` (`evalml.pipelines.components.estimators.KNeighborsClassifier`), 646  
`parameters()` (`evalml.pipelines.components.estimators.KNeighborsClassifier`), 565  
`parameters()` (`evalml.pipelines.components.estimators.LightGBMClassifier`), 649  
`parameters()` (`evalml.pipelines.components.estimators.LightGBMClassifier`), 567  
`parameters()` (`evalml.pipelines.components.estimators.LightGBMRegressor`), 652  
`parameters()` (`evalml.pipelines.components.estimators.LightGBMRegressor`), 602  
`parameters()` (`evalml.pipelines.components.estimators.LinearRegressor`), 654  
`parameters()` (`evalml.pipelines.components.estimators.LinearRegressor`), 570  
`parameters()` (`evalml.pipelines.components.estimators.LogisticRegressionClassifier`), 657  
`parameters()` (`evalml.pipelines.components.estimators.LogisticRegressionClassifier`), 605  
`parameters()` (`evalml.pipelines.components.estimators.ProphetRegressor`), 660  
`parameters()` (`evalml.pipelines.components.estimators.ProphetRegressor`), 573  
`parameters()` (`evalml.pipelines.components.estimators.RandomForestClassifier`), 662  
`parameters()` (`evalml.pipelines.components.estimators.RandomForestClassifier`), 607  
`parameters()` (`evalml.pipelines.components.estimators.RandomForestRegressor`), 664  
`parameters()` (`evalml.pipelines.components.estimators.RandomForestRegressor`), 667

`parameters()` (`evalml.pipelines.components.estimators.SVMRegressor`), 669  
`parameters()` (`evalml.pipelines.components.estimators.SVMRegressor`), 669  
`parameters()` (`evalml.pipelines.components.estimators.TimeSeriesBaselineEstimator`), 671  
`parameters()` (`evalml.pipelines.components.estimators.XGBoostClassifier`), 674  
`parameters()` (`evalml.pipelines.components.estimators.XGBoostRegressor`), 677  
`parameters()` (`evalml.pipelines.components.ExtraTreesClassifier`), 960  
`parameters()` (`evalml.pipelines.components.ExtraTreesRegressor`), 963  
`parameters()` (`evalml.pipelines.components.FeatureSelector`), 965  
`parameters()` (`evalml.pipelines.components.Imputer`), 968  
`parameters()` (`evalml.pipelines.components.KNeighborsClassifier`), 971  
`parameters()` (`evalml.pipelines.components.LightGBMClassifier`), 973  
`parameters()` (`evalml.pipelines.components.LightGBMRegressor`), 976  
`parameters()` (`evalml.pipelines.components.LinearDiscriminantAnalysis`), 978  
`parameters()` (`evalml.pipelines.components.LinearRegression`), 981  
`parameters()` (`evalml.pipelines.components.LogisticRegressionClassifier`), 983  
`parameters()` (`evalml.pipelines.components.LogTransform`), 985  
`parameters()` (`evalml.pipelines.components.LSA`), 988  
`parameters()` (`evalml.pipelines.components.OneHotEncoder`), 991  
`parameters()` (`evalml.pipelines.components.PCA`), 993  
`parameters()` (`evalml.pipelines.components.PerColumnTransformer`), 995  
`parameters()` (`evalml.pipelines.components.PolynomialDegreeReducer`), 997  
`parameters()` (`evalml.pipelines.components.ProphetRegressor`), 1000  
`parameters()` (`evalml.pipelines.components.RandomForestClassifier`), 1002  
`parameters()` (`evalml.pipelines.components.RandomForestRegressor`), 1004  
`parameters()` (`evalml.pipelines.components.RFClassifierSelectFromModel`), 1007  
`parameters()` (`evalml.pipelines.components.RFRegressorSelectFromModel`), 1010  
`parameters()` (`evalml.pipelines.components.SelectByType`), 1012  
`parameters()` (`evalml.pipelines.components.SelectColumns`), 1014  
`parameters()` (`evalml.pipelines.components.SimpleImputer`), 1016  
`parameters()` (`evalml.pipelines.components.SklearnStackedEnsembleClassifier`), 1019  
`parameters()` (`evalml.pipelines.components.SklearnStackedEnsembleRegressor`), 1021  
`parameters()` (`evalml.pipelines.components.SMOTEOverSampler`), 1024  
`parameters()` (`evalml.pipelines.components.SMOTENOverSampler`), 1026  
`parameters()` (`evalml.pipelines.components.SMOTEOverSampler`), 1028  
`parameters()` (`evalml.pipelines.components.StandardScaler`), 1030  
`parameters()` (`evalml.pipelines.components.SVMClassifier`), 1032  
`parameters()` (`evalml.pipelines.components.SVMRegressor`), 1035  
`parameters()` (`evalml.pipelines.components.TargetEncoder`), 1037  
`parameters()` (`evalml.pipelines.components.TargetImputer`), 1039  
`parameters()` (`evalml.pipelines.components.TextFeaturizer`), 1042  
`parameters()` (`evalml.pipelines.components.TimeSeriesBaselineEstimator`), 1044  
`parameters()` (`evalml.pipelines.components.Transformer`), 1046  
`parameters()` (`evalml.pipelines.components.transformers.column_selector`), 825  
`parameters()` (`evalml.pipelines.components.transformers.column_selector`), 827  
`parameters()` (`evalml.pipelines.components.transformers.column_selector`), 829  
`parameters()` (`evalml.pipelines.components.transformers.column_selector`), 831  
`parameters()` (`evalml.pipelines.components.transformers.DateTimeFeaturizer`), 839  
`parameters()` (`evalml.pipelines.components.transformers.DelayedFeaturizer`), 842  
`parameters()` (`evalml.pipelines.components.transformers.DFSTransformer`), 844  
`parameters()` (`evalml.pipelines.components.transformers.dimensionality_reduction`), 679  
`parameters()` (`evalml.pipelines.components.transformers.dimensionality_reduction`), 684  
`parameters()` (`evalml.pipelines.components.transformers.dimensionality_reduction`), 686  
`parameters()` (`evalml.pipelines.components.transformers.dimensionality_reduction`), 682  
`parameters()` (`evalml.pipelines.components.transformers.DropColumnTransformer`), 846  
`parameters()` (`evalml.pipelines.components.transformers.DropNullColumns`), 848

`parameters()` (`evalml.pipelines.components.transformers.EmailFeaturizer`), 850  
`parameters()` (`evalml.pipelines.components.transformers.EmailFeaturizer`), 871  
`parameters()` (`evalml.pipelines.components.transformers.OneHotEncoder`), 690  
`parameters()` (`evalml.pipelines.components.transformers.OneHotEncoder`), 742  
`parameters()` (`evalml.pipelines.components.transformers.OneHotEncoder`), 770  
`parameters()` (`evalml.pipelines.components.transformers.OneHotEncoder`), 770  
`parameters()` (`evalml.pipelines.components.transformers.TargetEncoder`), 694  
`parameters()` (`evalml.pipelines.components.transformers.TargetEncoder`), 745  
`parameters()` (`evalml.pipelines.components.transformers.TargetEncoder`), 773  
`parameters()` (`evalml.pipelines.components.transformers.FeatureSelector`), 702  
`parameters()` (`evalml.pipelines.components.transformers.FeatureSelector`), 775  
`parameters()` (`evalml.pipelines.components.transformers.FeatureSelector`), 775  
`parameters()` (`evalml.pipelines.components.transformers.FeatureSelector`), 747  
`parameters()` (`evalml.pipelines.components.transformers.FeatureSelector`), 777  
`parameters()` (`evalml.pipelines.components.transformers.FeatureSelector`), 777  
`parameters()` (`evalml.pipelines.components.transformers.FeatureSelector`), 779  
`parameters()` (`evalml.pipelines.components.transformers.FeatureSelector`), 750  
`parameters()` (`evalml.pipelines.components.transformers.FeatureSelector`), 750  
`parameters()` (`evalml.pipelines.components.transformers.FeatureSelector`), 752  
`parameters()` (`evalml.pipelines.components.transformers.FeatureSelector`), 781  
`parameters()` (`evalml.pipelines.components.transformers.FeatureSelector`), 783  
`parameters()` (`evalml.pipelines.components.transformers.FeatureSelector`), 755  
`parameters()` (`evalml.pipelines.components.transformers.FeatureSelector`), 757  
`parameters()` (`evalml.pipelines.components.transformers.FeatureSelector`), 786  
`parameters()` (`evalml.pipelines.components.transformers.FeatureSelector`), 760  
`parameters()` (`evalml.pipelines.components.transformers.FeatureSelector`), 763  
`parameters()` (`evalml.pipelines.components.transformers.FeatureSelector`), 788  
`parameters()` (`evalml.pipelines.components.transformers.FeatureSelector`), 790  
`parameters()` (`evalml.pipelines.components.transformers.FeatureSelector`), 765  
`parameters()` (`evalml.pipelines.components.transformers.FeatureSelector`), 767  
`parameters()` (`evalml.pipelines.components.transformers.FeatureSelector`), 792  
`parameters()` (`evalml.pipelines.components.transformers.FeatureSelector`), 861  
`parameters()` (`evalml.pipelines.components.transformers.FeatureSelector`), 874  
`parameters()` (`evalml.pipelines.components.transformers.FeatureSelector`), 877  
`parameters()` (`evalml.pipelines.components.transformers.FeatureSelector`), 795  
`parameters()` (`evalml.pipelines.components.transformers.FeatureSelector`), 797



`parameters()` (`evalml.pipelines.components.transformers.samplers.SMOTEENNComplexXGBoostClassifier` property), 800  
`parameters()` (`evalml.pipelines.components.transformers.samplers.SMOTEENNOvercomplexXGBoostRegressor` property), 802  
`parameters()` (`evalml.pipelines.components.transformers.samplers.SMOTEENNDecisionTreeClassifier` property), 805  
`parameters()` (`evalml.pipelines.components.transformers.samplers.SMOTEENNDecisionTreeRegressor` property), 810  
`parameters()` (`evalml.pipelines.components.transformers.samplers.SMOTEENNDelayedFeatureTransformer` property), 812  
`parameters()` (`evalml.pipelines.components.transformers.samplers.SMOTEENNDFSTransformer` property), 815  
`parameters()` (`evalml.pipelines.components.transformers.samplers.undersampler.ElasticNetClassifier` property), 817  
`parameters()` (`evalml.pipelines.components.transformers.samplers.undersampler.ElasticNetRegressor` property), 807  
`parameters()` (`evalml.pipelines.components.transformers.scalers.StandardScaler` property), 820  
`parameters()` (`evalml.pipelines.components.transformers.scalers.StandardScaler` property), 822  
`parameters()` (`evalml.pipelines.components.transformers.selectors.SelectByType` property), 879  
`parameters()` (`evalml.pipelines.components.transformers.selectors.SelectColumns` property), 881  
`parameters()` (`evalml.pipelines.components.transformers.simpletransformer.KNeighborsClassifier` property), 883  
`parameters()` (`evalml.pipelines.components.transformers.smoteenn.SMOTEENNComplexXGBoostClassifier` property), 886  
`parameters()` (`evalml.pipelines.components.transformers.smoteenn.SMOTEENNOvercomplexXGBoostRegressor` property), 888  
`parameters()` (`evalml.pipelines.components.transformers.smoteenn.SMOTEENNDecisionTreeClassifier` property), 890  
`parameters()` (`evalml.pipelines.components.transformers.smoteenn.SMOTEENNDecisionTreeRegressor` property), 892  
`parameters()` (`evalml.pipelines.components.transformers.smoteenn.SMOTEENNDelayedFeatureTransformer` property), 895  
`parameters()` (`evalml.pipelines.components.transformers.smoteenn.SMOTEENNDFSTransformer` property), 897  
`parameters()` (`evalml.pipelines.components.transformers.smoteenn.SMOTEENNComplexXGBoostClassifier` property), 899  
`parameters()` (`evalml.pipelines.components.transformers.transformers.PerColumnImputer` property), 901  
`parameters()` (`evalml.pipelines.components.transformers.transformers.pipeline_base.PipelineBase` property), 833  
`parameters()` (`evalml.pipelines.components.transformers.transformers.pipeline_base.PipelineBase` property), 836  
`parameters()` (`evalml.pipelines.components.transformers.transformers.ProphetRegressor` property), 904  
`parameters()` (`evalml.pipelines.components.transformers.transformers.RandomForestClassifier` property), 906  
`parameters()` (`evalml.pipelines.components.undersampler.undersampler.RandomForestRegressor` property), 1049  
`parameters()` (`evalml.pipelines.components.URLFeaturizer` property), 1051  
`parameters()` (`evalml.pipelines.components.transformers.samplers.SMOTEENNComplexXGBoostClassifier` property), 1053  
`parameters()` (`evalml.pipelines.components.transformers.samplers.SMOTEENNOvercomplexXGBoostRegressor` property), 1055  
`parameters()` (`evalml.pipelines.components.transformers.samplers.SMOTEENNDecisionTreeClassifier` property), 1117  
`parameters()` (`evalml.pipelines.components.transformers.samplers.SMOTEENNDecisionTreeRegressor` property), 1120  
`parameters()` (`evalml.pipelines.components.transformers.samplers.SMOTEENNDelayedFeatureTransformer` property), 1123  
`parameters()` (`evalml.pipelines.components.transformers.samplers.SMOTEENNDFSTransformer` property), 1125  
`parameters()` (`evalml.pipelines.components.transformers.undersampler.ElasticNetClassifier` property), 1128  
`parameters()` (`evalml.pipelines.components.transformers.undersampler.ElasticNetRegressor` property), 1130  
`parameters()` (`evalml.pipelines.components.transformers.scalers.StandardScaler` property), 1132  
`parameters()` (`evalml.pipelines.components.transformers.scalers.StandardScaler` property), 1135  
`parameters()` (`evalml.pipelines.components.transformers.selectors.SelectByType` property), 1138  
`parameters()` (`evalml.pipelines.components.transformers.selectors.SelectColumns` property), 1140  
`parameters()` (`evalml.pipelines.components.transformers.simpletransformer.KNeighborsClassifier` property), 1143  
`parameters()` (`evalml.pipelines.components.transformers.smoteenn.SMOTEENNComplexXGBoostClassifier` property), 1146  
`parameters()` (`evalml.pipelines.components.transformers.smoteenn.SMOTEENNOvercomplexXGBoostRegressor` property), 1148  
`parameters()` (`evalml.pipelines.components.transformers.smoteenn.SMOTEENNDecisionTreeClassifier` property), 1151  
`parameters()` (`evalml.pipelines.components.transformers.smoteenn.SMOTEENNDecisionTreeRegressor` property), 1153  
`parameters()` (`evalml.pipelines.components.transformers.smoteenn.SMOTEENNDelayedFeatureTransformer` property), 1073  
`parameters()` (`evalml.pipelines.components.transformers.smoteenn.SMOTEENNDFSTransformer` property), 1157  
`parameters()` (`evalml.pipelines.components.transformers.smoteenn.SMOTEENNComplexXGBoostClassifier` property), 1160  
`parameters()` (`evalml.pipelines.components.transformers.transformers.PerColumnImputer` property), 1163  
`parameters()` (`evalml.pipelines.components.transformers.transformers.pipeline_base.PipelineBase` property), 1078  
`parameters()` (`evalml.pipelines.components.transformers.transformers.pipeline_base.PipelineBase` property), 1166  
`parameters()` (`evalml.pipelines.components.transformers.transformers.ProphetRegressor` property), 1169  
`parameters()` (`evalml.pipelines.components.transformers.transformers.RandomForestClassifier` property), 1171  
`parameters()` (`evalml.pipelines.components.undersampler.undersampler.RandomForestRegressor` property), 1173  
`parameters()` (`evalml.pipelines.components.transformers.transformers.pipeline_base.PipelineBase` property), 1083

`parameters()` (`evalml.pipelines.RegressionPipeline` `PCA` (class in `evalml.pipelines.components.transformers.dimensionality_reducers`), 1177, 685  
`parameters()` (`evalml.pipelines.RFClassifierSelectFromModel` (class in `evalml.pipelines.components.transformers.dimensionality_reducers`), 1180, 680  
`parameters()` (`evalml.pipelines.RFRegressorSelectFromModel` (class in `evalml.pipelines.components.transformers.imputers`), 1182, 1161  
`parameters()` (`evalml.pipelines.SimpleImputer` property), 1185, 993  
`parameters()` (`evalml.pipelines.SklearnStackedEnsembleClassifier` (class in `evalml.pipelines.components.transformers`), 1187, 867  
`parameters()` (`evalml.pipelines.SklearnStackedEnsembleRegressor` (class in `evalml.pipelines.components.transformers.imputers`), 1190, 732  
`parameters()` (`evalml.pipelines.StandardScaler` property), 1192, 721  
`parameters()` (`evalml.pipelines.SVMClassifier` property), 1194, 721  
`parameters()` (`evalml.pipelines.SVMRegressor` property), 1196, 289  
`parameters()` (`evalml.pipelines.TargetEncoder` property), 1199, 390  
`parameters()` (`evalml.pipelines.time_series_classification_pipeline` (class in `evalml.pipelines.time_series_classification_pipeline`), 1088, 299  
`parameters()` (`evalml.pipelines.time_series_classification_pipeline` (class in `evalml.pipelines.time_series_classification_pipeline`), 1093, 428  
`parameters()` (`evalml.pipelines.time_series_classification_pipeline` (class in `evalml.pipelines.time_series_classification_pipeline`), 1097, 302  
`parameters()` (`evalml.pipelines.time_series_regression_pipeline` (class in `evalml.pipelines.time_series_regression_pipeline`), 1101, 430  
`parameters()` (`evalml.pipelines.TimeSeriesBinaryClassificationPipeline` (class in `evalml.pipelines.time_series_classification_pipeline`), 1203, 304  
`parameters()` (`evalml.pipelines.TimeSeriesClassificationPipeline` (class in `evalml.pipelines.time_series_classification_pipeline`), 1207, 447  
`parameters()` (`evalml.pipelines.TimeSeriesMulticlassClassificationPipeline` (class in `evalml.pipelines.time_series_classification_pipeline`), 1211, 370  
`parameters()` (`evalml.pipelines.TimeSeriesRegressionPipeline` (class in `evalml.pipelines.time_series_regression_pipeline`), 1215, 370  
`parameters()` (`evalml.pipelines.Transformer` property), 1218, 157  
`parameters()` (`evalml.pipelines.XGBoostClassifier` property), 1220, 163  
`parameters()` (`evalml.pipelines.XGBoostRegressor` property), 1223, 159  
`partial_dependence()` (in module `evalml.model_understanding`), 284, 164  
`partial_dependence()` (in module `evalml.model_understanding.graphs`), 271, 164  
`PartialDependenceError`, 250, 252  
`PartialDependenceErrorCode` (class in `evalml.exceptions`), 252, 161  
`PartialDependenceErrorCode` (class in `evalml.exceptions.exceptions`), 250, 166  
`PCA` (class in `evalml.pipelines.components`), 991  
`PCA` (class in `evalml.pipelines.components.transformers`), 865  
`PipelineBase` (class in `evalml.pipelines`), 1163  
`PipelineBase` (class in `evalml.pipelines`), 1163

[evalml.pipelines.pipeline\\_base](#)), 1075  
 PipelineBaseMeta (class [evalml.pipelines.pipeline\\_meta](#)), 1079  
 PipelineNotFoundError, 251, 253  
 PipelineNotYetFittedError, 251, 253  
 PipelineScoreError, 251, 253  
 PipelineSearchPlots (class [evalml.automl.pipeline\\_search\\_plots](#)), 188  
 plot() ([evalml.automl.automl\\_search.AutoMLSearch](#) property), 185  
 plot() ([evalml.automl.AutoMLSearch](#) property), 195  
 plot() ([evalml.AutoMLSearch](#) property), 1266  
 PolynomialDetrender (class [evalml.pipelines.components](#)), 995  
 PolynomialDetrender (class [evalml.pipelines.components.transformers](#)), 869  
 PolynomialDetrender (class [evalml.pipelines.components.transformers.preprocessing](#)), 784  
 PolynomialDetrender (class [evalml.pipelines.components.transformers.preprocessing.polynomial\\_detrender](#)), 756  
 positive\_only() ([evalml.objectives.AccuracyBinary](#) method), 376  
 positive\_only() ([evalml.objectives.AccuracyMulticlass](#) method), 377  
 positive\_only() ([evalml.objectives.AUC](#) method), 379  
 positive\_only() ([evalml.objectives.AUCMacro](#) method), 381  
 positive\_only() ([evalml.objectives.AUCMicro](#) method), 383  
 positive\_only() ([evalml.objectives.AUCWeighted](#) method), 384  
 positive\_only() ([evalml.objectives.BalancedAccuracyBinary](#) method), 386  
 positive\_only() ([evalml.objectives.BalancedAccuracyMulticlass](#) method), 388  
 positive\_only() ([evalml.objectives.binary\\_classification\\_objective.BinaryClassificationObjective](#) method), 289  
 positive\_only() ([evalml.objectives.BinaryClassificationObjective](#) method), 390  
 positive\_only() ([evalml.objectives.cost\\_benefit\\_matrix.CostBenefitMatrix](#) method), 292  
 positive\_only() ([evalml.objectives.CostBenefitMatrix](#) method), 393  
 positive\_only() ([evalml.objectives.ExpVariance](#) method), 394  
 positive\_only() ([evalml.objectives.F1](#) method), 397  
 positive\_only() ([evalml.objectives.F1Macro](#) method), 398  
 positive\_only() ([evalml.objectives.F1Micro](#) method), 400  
 positive\_only() ([evalml.objectives.F1Weighted](#) method), 401  
 positive\_only() ([evalml.objectives.fraud\\_cost.FraudCost](#) method), 295  
 positive\_only() ([evalml.objectives.FraudCost](#) method), 404  
 positive\_only() ([evalml.objectives.Gini](#) method), 407  
 positive\_only() ([evalml.objectives.lead\\_scoring.LeadScoring](#) method), 297  
 positive\_only() ([evalml.objectives.LeadScoring](#) method), 409  
 positive\_only() ([evalml.objectives.LogLossBinary](#) method), 411  
 positive\_only() ([evalml.objectives.LogLossMulticlass](#) method), 413  
 positive\_only() ([evalml.objectives.MAE](#) method), 414  
 positive\_only() ([evalml.objectives.MAPE](#) method), 416  
 positive\_only() ([evalml.objectives.MaxError](#) method), 418  
 positive\_only() ([evalml.objectives.MCCBinary](#) method), 420  
 positive\_only() ([evalml.objectives.MCCMulticlass](#) method), 421  
 positive\_only() ([evalml.objectives.MeanSquaredLogError](#) method), 423  
 positive\_only() ([evalml.objectives.MedianAE](#) method), 425  
 positive\_only() ([evalml.objectives.MSE](#) method), 426  
 positive\_only() ([evalml.objectives.multiclass\\_classification\\_objective.MulticlassClassificationObjective](#) method), 299  
 positive\_only() ([evalml.objectives.MulticlassClassificationObjective](#) method), 428  
 positive\_only() ([evalml.objectives.objective\\_base.ObjectiveBase](#) method), 302  
 positive\_only() ([evalml.objectives.ObjectiveBase](#) method), 430  
 positive\_only() ([evalml.objectives.Precision](#) method), 432  
 positive\_only() ([evalml.objectives.PrecisionMacro](#) method), 434  
 positive\_only() ([evalml.objectives.PrecisionMicro](#) method), 435  
 positive\_only() ([evalml.objectives.PrecisionWeighted](#) method), 437  
 positive\_only() ([evalml.objectives.R2](#) method), 439  
 positive\_only() ([evalml.objectives.Recall](#) method), 441  
 positive\_only() ([evalml.objectives.RecallMacro](#) method), 442

method), 442

positive\_only() (evalml.objectives.RecallMicro method), 444

positive\_only() (evalml.objectives.RecallWeighted method), 446

positive\_only() (evalml.objectives.regression\_objective.RegressionObjective method), 304

positive\_only() (evalml.objectives.RegressionObjective method), 448

positive\_only() (evalml.objectives.RootMeanSquaredError method), 449

positive\_only() (evalml.objectives.RootMeanSquaredLogError method), 451

positive\_only() (evalml.objectives.sensitivity\_low\_alert.SensitivityLowAlert method), 306

positive\_only() (evalml.objectives.SensitivityLowAlert method), 453

positive\_only() (evalml.objectives.standard\_metrics.AccuracyBinary method), 310

positive\_only() (evalml.objectives.standard\_metrics.AccuracyMulticlass method), 311

positive\_only() (evalml.objectives.standard\_metrics.AUC method), 314

positive\_only() (evalml.objectives.standard\_metrics.AUCMacro method), 315

positive\_only() (evalml.objectives.standard\_metrics.AUCMicro method), 317

positive\_only() (evalml.objectives.standard\_metrics.AUCWeighted method), 318

positive\_only() (evalml.objectives.standard\_metrics.BalancedAccuracyBinary method), 321

positive\_only() (evalml.objectives.standard\_metrics.BalancedAccuracyMulticlass method), 322

positive\_only() (evalml.objectives.standard\_metrics.ExplainedVariance method), 324

positive\_only() (evalml.objectives.standard\_metrics.F1 method), 326

positive\_only() (evalml.objectives.standard\_metrics.F1Macro method), 328

positive\_only() (evalml.objectives.standard\_metrics.F1Micro method), 329

positive\_only() (evalml.objectives.standard\_metrics.F1Weighted method), 331

positive\_only() (evalml.objectives.standard\_metrics.Gini method), 333

positive\_only() (evalml.objectives.standard\_metrics.LogLossBinary method), 335

positive\_only() (evalml.objectives.standard\_metrics.LogLossMulticlass method), 337

positive\_only() (evalml.objectives.standard\_metrics.MAE method), 338

positive\_only() (evalml.objectives.standard\_metrics.MAPE method), 340

positive\_only() (evalml.objectives.standard\_metrics.MaxError method), 341

positive\_only() (evalml.objectives.standard\_metrics.MCCBinary method), 343

positive\_only() (evalml.objectives.standard\_metrics.MCCMulticlass method), 345

positive\_only() (evalml.objectives.standard\_metrics.MeanSquaredLogError method), 347

positive\_only() (evalml.objectives.standard\_metrics.MedianAE method), 348

positive\_only() (evalml.objectives.standard\_metrics.MSE method), 350

positive\_only() (evalml.objectives.standard\_metrics.Precision method), 352

positive\_only() (evalml.objectives.standard\_metrics.PrecisionMacro method), 354

positive\_only() (evalml.objectives.standard\_metrics.PrecisionMicro method), 355

positive\_only() (evalml.objectives.standard\_metrics.PrecisionWeighted method), 357

positive\_only() (evalml.objectives.standard\_metrics.R2 method), 358

positive\_only() (evalml.objectives.standard\_metrics.Recall method), 360

positive\_only() (evalml.objectives.standard\_metrics.RecallMacro method), 362

positive\_only() (evalml.objectives.standard\_metrics.RecallMicro method), 364

positive\_only() (evalml.objectives.standard\_metrics.RecallWeighted method), 365

positive\_only() (evalml.objectives.standard\_metrics.RootMeanSquaredError method), 367

positive\_only() (evalml.objectives.standard\_metrics.RootMeanSquaredLogError method), 368

positive\_only() (evalml.objectives.time\_series\_regression\_objective.ExplainedVariance method), 370

Precision (class in evalml.objectives), 431

Precision (class in evalml.objectives.standard\_metrics), 350

precision\_recall\_curve() (in module evalml.model\_understanding), 285

precision\_recall\_curve() (in module evalml.model\_understanding.graphs), 272

PrecisionMacro (class in evalml.objectives), 433

PrecisionMacro (class in evalml.objectives.standard\_metrics), 353

PrecisionMicro (class in evalml.objectives), 434

PrecisionMicro (class in evalml.objectives.standard\_metrics), 354

PrecisionWeighted (class in evalml.objectives), 436

PrecisionWeighted (class in evalml.objectives.standard\_metrics), 356

predict() (evalml.pipelines.ARIMAREgressor method), 1107



<b>Index</b>	<b>1401</b>
--------------	-------------

`predict()` (`evalml.pipelines.components.estimators.estimators.Estimator` or `evalml.pipelines.components.estimators.regressors.ProphetRegressor`), 610

`predict()` (`evalml.pipelines.components.estimators.ExtraTreesClassifier` or `evalml.pipelines.components.estimators.regressors.ProphetRegressor`), 641

`predict()` (`evalml.pipelines.components.estimators.ExtraTreesRegressor` or `evalml.pipelines.components.estimators.regressors.RandomForestClassifier`), 644

`predict()` (`evalml.pipelines.components.estimators.KNeighborsClassifier` or `evalml.pipelines.components.estimators.regressors.rf_regressor`), 646

`predict()` (`evalml.pipelines.components.estimators.LightGBMClassifier` or `evalml.pipelines.components.estimators.regressors.svm_regressor`), 649

`predict()` (`evalml.pipelines.components.estimators.LightGBMRegressor` or `evalml.pipelines.components.estimators.regressors.SVMRegressor`), 652

`predict()` (`evalml.pipelines.components.estimators.LinearRegressor` or `evalml.pipelines.components.estimators.regressors.time_series_regressor`), 655

`predict()` (`evalml.pipelines.components.estimators.LogisticRegressionClassifier` or `evalml.pipelines.components.estimators.regressors.TimeSeriesRegressor`), 657

`predict()` (`evalml.pipelines.components.estimators.ProphetRegressor` or `evalml.pipelines.components.estimators.regressors.xgboost_classifier`), 660

`predict()` (`evalml.pipelines.components.estimators.RandomForestClassifier` or `evalml.pipelines.components.estimators.regressors.XGBoostClassifier`), 662

`predict()` (`evalml.pipelines.components.estimators.RandomForestRegressor` or `evalml.pipelines.components.estimators.SVMClassifier`), 664

`predict()` (`evalml.pipelines.components.estimators.regressors.arima_regressor` or `evalml.pipelines.components.estimators.SVMRegressor`), 538

`predict()` (`evalml.pipelines.components.estimators.regressors.ARIMARegressor` or `evalml.pipelines.components.estimators.TimeSeriesBaselineRegressor`), 576

`predict()` (`evalml.pipelines.components.estimators.regressors.baseline_regressor` or `evalml.pipelines.components.estimators.XGBoostClassifier`), 541

`predict()` (`evalml.pipelines.components.estimators.regressors.BaselineRegressor` or `evalml.pipelines.components.estimators.XGBoostRegressor`), 579

`predict()` (`evalml.pipelines.components.estimators.regressors.catboost_regressor` or `evalml.pipelines.components.estimators.XGBoostRegressor`), 544

`predict()` (`evalml.pipelines.components.estimators.regressors.CatBoostRegressor` or `evalml.pipelines.components.ExtraTreesRegressor`), 581

`predict()` (`evalml.pipelines.components.estimators.regressors.decision_tree_regressor` or `evalml.pipelines.DavissonTreeRegressor`), 547

`predict()` (`evalml.pipelines.components.estimators.regressors.DecisionTreeRegressor` or `evalml.pipelines.components.LightGBMClassifier`), 584

`predict()` (`evalml.pipelines.components.estimators.regressors.elasticnet_regressor` or `evalml.pipelines.ElasticNetRegressor`), 550

`predict()` (`evalml.pipelines.components.estimators.regressors.ElasticNetRegressor` or `evalml.pipelines.components.LinearRegressor`), 587

`predict()` (`evalml.pipelines.components.estimators.regressors.extra_trees_regressor` or `evalml.pipelines.components.LogisticRegressionClassifier`), 553

`predict()` (`evalml.pipelines.components.estimators.regressors.ExtraTreesRegressor` or `evalml.pipelines.components.ProphetRegressor`), 590

`predict()` (`evalml.pipelines.components.estimators.regressors.lightgbm_regressor` or `evalml.pipelines.LightGBMRegressor`), 556

`predict()` (`evalml.pipelines.components.estimators.regressors.LightGBMRegressor` or `evalml.pipelines.components.RandomForestRegressor`), 593

`predict()` (`evalml.pipelines.components.estimators.regressors.linear_regressor` or `evalml.pipelines.LinearRegressor`), 559

`predict()` (`evalml.pipelines.components.estimators.regressors.LinearRegressor` or `evalml.pipelines.components.SklearnStackedEnsembleRegressor`), 595

[predict \(\) \(evalml.pipelines.components.SVMClassifier method\), 1177](#)  
[predict \(\) \(evalml.pipelines.components.SVMRegressor method\), 1032](#)  
[predict \(\) \(evalml.pipelines.components.SVMRegressor method\), 1187](#)  
[predict \(\) \(evalml.pipelines.components.TimeSeriesBaselineEstimator method\), 1190](#)  
[predict \(\) \(evalml.pipelines.components.time\\_series\\_classification\\_pipelines.TimeSeriesClassifier method\), 1044](#)  
[predict \(\) \(evalml.pipelines.components.time\\_series\\_classification\\_pipelines.TimeSeriesRegressor method\), 1194](#)  
[predict \(\) \(evalml.pipelines.components.time\\_series\\_regression\\_pipeline.TimeSeriesRegressor method\), 912](#)  
[predict \(\) \(evalml.pipelines.components.time\\_series\\_regression\\_pipeline.TimeSeriesRegressor method\), 1196](#)  
[predict \(\) \(evalml.pipelines.components.XGBoostClassifier method\), 1088](#)  
[predict \(\) \(evalml.pipelines.components.XGBoostClassifier method\), 1053](#)  
[predict \(\) \(evalml.pipelines.components.XGBoostRegressor method\), 1093](#)  
[predict \(\) \(evalml.pipelines.components.XGBoostRegressor method\), 1056](#)  
[predict \(\) \(evalml.pipelines.DecisionTreeClassifier method\), 1097](#)  
[predict \(\) \(evalml.pipelines.DecisionTreeClassifier method\), 1118](#)  
[predict \(\) \(evalml.pipelines.DecisionTreeRegressor method\), 1101](#)  
[predict \(\) \(evalml.pipelines.DecisionTreeRegressor method\), 1120](#)  
[predict \(\) \(evalml.pipelines.ElasticNetClassifier method\), 1203](#)  
[predict \(\) \(evalml.pipelines.ElasticNetClassifier method\), 1128](#)  
[predict \(\) \(evalml.pipelines.ElasticNetRegressor method\), 1207](#)  
[predict \(\) \(evalml.pipelines.ElasticNetRegressor method\), 1130](#)  
[predict \(\) \(evalml.pipelines.Estimator method\), 1132](#)  
[predict \(\) \(evalml.pipelines.ExtraTreesClassifier method\), 1211](#)  
[predict \(\) \(evalml.pipelines.ExtraTreesClassifier method\), 1135](#)  
[predict \(\) \(evalml.pipelines.ExtraTreesRegressor method\), 1215](#)  
[predict \(\) \(evalml.pipelines.ExtraTreesRegressor method\), 1138](#)  
[predict \(\) \(evalml.pipelines.KNeighborsClassifier method\), 1220](#)  
[predict \(\) \(evalml.pipelines.KNeighborsClassifier method\), 1143](#)  
[predict \(\) \(evalml.pipelines.LightGBMClassifier method\), 1223](#)  
[predict \(\) \(evalml.pipelines.LightGBMClassifier method\), 1146](#)  
[predict \(\) \(evalml.pipelines.LightGBMRegressor method\), 1107](#)  
[predict \(\) \(evalml.pipelines.LightGBMRegressor method\), 1148](#)  
[predict \(\) \(evalml.pipelines.LinearRegressor method\), 1110](#)  
[predict \(\) \(evalml.pipelines.LinearRegressor method\), 1151](#)  
[predict \(\) \(evalml.pipelines.LogisticRegressionClassifier method\), 1112](#)  
[predict \(\) \(evalml.pipelines.LogisticRegressionClassifier method\), 1153](#)  
[predict \(\) \(evalml.pipelines.MulticlassClassificationPipeline method\), 1066](#)  
[predict \(\) \(evalml.pipelines.MulticlassClassificationPipeline method\), 1073](#)  
[predict \(\) \(evalml.pipelines.MulticlassClassificationPipeline method\), 1157](#)  
[predict \(\) \(evalml.pipelines.pipeline\\_base.PipelineBase method\), 1066](#)  
[predict \(\) \(evalml.pipelines.pipeline\\_base.PipelineBase method\), 1078](#)  
[predict \(\) \(evalml.pipelines.PipelineBase method\), 1166](#)  
[predict \(\) \(evalml.pipelines.PipelineBase method\), 1169](#)  
[predict \(\) \(evalml.pipelines.ProphetRegressor method\), 926](#)  
[predict \(\) \(evalml.pipelines.RandomForestClassifier method\), 928](#)  
[predict \(\) \(evalml.pipelines.RandomForestClassifier method\), 1171](#)  
[predict \(\) \(evalml.pipelines.RandomForestRegressor method\), 936](#)  
[predict \(\) \(evalml.pipelines.RandomForestRegressor method\), 1173](#)  
[predict \(\) \(evalml.pipelines.regression\\_pipeline.RegressionPipeline method\), 939](#)  
[predict \(\) \(evalml.pipelines.regression\\_pipeline.RegressionPipeline method\), 1083](#)  
[predict \(\) \(evalml.pipelines.RegressionPipeline method\), 939](#)  
[predict \(\) \(evalml.pipelines.RegressionPipeline method\), 1083](#)  
[predict \(\) \(evalml.pipelines.SklearnStackedEnsembleClassifier method\), 1177](#)  
[predict \(\) \(evalml.pipelines.SklearnStackedEnsembleClassifier method\), 1187](#)  
[predict \(\) \(evalml.pipelines.SklearnStackedEnsembleRegressor method\), 1190](#)  
[predict \(\) \(evalml.pipelines.SklearnStackedEnsembleRegressor method\), 1194](#)  
[predict \(\) \(evalml.pipelines.SklearnStackedEnsembleRegressor method\), 1196](#)  
[predict \(\) \(evalml.pipelines.time\\_series\\_classification\\_pipelines.TimeSeriesClassifier method\), 1044](#)  
[predict \(\) \(evalml.pipelines.time\\_series\\_classification\\_pipelines.TimeSeriesRegressor method\), 1194](#)  
[predict \(\) \(evalml.pipelines.time\\_series\\_classification\\_pipelines.TimeSeriesRegressor method\), 1196](#)  
[predict \(\) \(evalml.pipelines.time\\_series\\_regression\\_pipeline.TimeSeriesRegressor method\), 1101](#)  
[predict \(\) \(evalml.pipelines.time\\_series\\_regression\\_pipeline.TimeSeriesRegressor method\), 1120](#)  
[predict \(\) \(evalml.pipelines.TimeSeriesBinaryClassificationPipeline method\), 1203](#)  
[predict \(\) \(evalml.pipelines.TimeSeriesBinaryClassificationPipeline method\), 1207](#)  
[predict \(\) \(evalml.pipelines.TimeSeriesClassificationPipeline method\), 1211](#)  
[predict \(\) \(evalml.pipelines.TimeSeriesClassificationPipeline method\), 1215](#)  
[predict \(\) \(evalml.pipelines.TimeSeriesMulticlassClassificationPipeline method\), 1211](#)  
[predict \(\) \(evalml.pipelines.TimeSeriesMulticlassClassificationPipeline method\), 1215](#)  
[predict \(\) \(evalml.pipelines.TimeSeriesRegressionPipeline method\), 1215](#)  
[predict \(\) \(evalml.pipelines.TimeSeriesRegressionPipeline method\), 1215](#)  
[predict \(\) \(evalml.pipelines.XGBoostClassifier method\), 1220](#)  
[predict \(\) \(evalml.pipelines.XGBoostClassifier method\), 1223](#)  
[predict \(\) \(evalml.pipelines.XGBoostRegressor method\), 1223](#)  
[predict \(\) \(evalml.pipelines.XGBoostRegressor method\), 1223](#)  
[predict\\_proba \(\) \(evalml.pipelines.ARIMARegressor method\), 1107](#)  
[predict\\_proba \(\) \(evalml.pipelines.ARIMARegressor method\), 1107](#)  
[predict\\_proba \(\) \(evalml.pipelines.binary\\_classification\\_pipeline.BinaryClassifier method\), 1060](#)  
[predict\\_proba \(\) \(evalml.pipelines.binary\\_classification\\_pipeline.BinaryClassifier method\), 1060](#)  
[predict\\_proba \(\) \(evalml.pipelines.CatBoostClassifier method\), 1110](#)  
[predict\\_proba \(\) \(evalml.pipelines.CatBoostClassifier method\), 1110](#)  
[predict\\_proba \(\) \(evalml.pipelines.CatBoostRegressor method\), 1112](#)  
[predict\\_proba \(\) \(evalml.pipelines.CatBoostRegressor method\), 1112](#)  
[predict\\_proba \(\) \(evalml.pipelines.classification\\_pipeline.ClassificationPipeline method\), 1066](#)  
[predict\\_proba \(\) \(evalml.pipelines.classification\\_pipeline.ClassificationPipeline method\), 1066](#)  
[predict\\_proba \(\) \(evalml.pipelines.components.ARIMARegressor method\), 919](#)  
[predict\\_proba \(\) \(evalml.pipelines.components.ARIMARegressor method\), 919](#)  
[predict\\_proba \(\) \(evalml.pipelines.components.BaselineClassifier method\), 921](#)  
[predict\\_proba \(\) \(evalml.pipelines.components.BaselineClassifier method\), 921](#)  
[predict\\_proba \(\) \(evalml.pipelines.components.BaselineRegressor method\), 923](#)  
[predict\\_proba \(\) \(evalml.pipelines.components.BaselineRegressor method\), 923](#)  
[predict\\_proba \(\) \(evalml.pipelines.components.CatBoostClassifier method\), 926](#)  
[predict\\_proba \(\) \(evalml.pipelines.components.CatBoostClassifier method\), 926](#)  
[predict\\_proba \(\) \(evalml.pipelines.components.CatBoostRegressor method\), 928](#)  
[predict\\_proba \(\) \(evalml.pipelines.components.CatBoostRegressor method\), 928](#)  
[predict\\_proba \(\) \(evalml.pipelines.components.DecisionTreeClassifier method\), 936](#)  
[predict\\_proba \(\) \(evalml.pipelines.components.DecisionTreeClassifier method\), 936](#)  
[predict\\_proba \(\) \(evalml.pipelines.components.DecisionTreeRegressor method\), 939](#)  
[predict\\_proba \(\) \(evalml.pipelines.components.DecisionTreeRegressor method\), 939](#)  
[predict\\_proba \(\) \(evalml.pipelines.components.ElasticNetClassifier method\), 1203](#)  
[predict\\_proba \(\) \(evalml.pipelines.components.ElasticNetClassifier method\), 1203](#)

method), 951

predict\_proba() (evalml.pipelines.components.ElasticNetRegressor), 953

predict\_proba() (evalml.pipelines.components.ensemble\_stacked\_ensemble\_classifier), 456

predict\_proba() (evalml.pipelines.components.ensemble\_stacked\_ensemble\_classifier), 459

predict\_proba() (evalml.pipelines.components.ensemble\_stacked\_ensemble\_classifier), 462

predict\_proba() (evalml.pipelines.components.ensemble\_stacked\_ensemble\_classifier), 465

predict\_proba() (evalml.pipelines.components.ensemble\_stacked\_ensemble\_classifier), 468

predict\_proba() (evalml.pipelines.components.ensemble\_stacked\_ensemble\_classifier), 470

predict\_proba() (evalml.pipelines.components.Estimator), 957

predict\_proba() (evalml.pipelines.components.estimators.ARMARegressor), 614

predict\_proba() (evalml.pipelines.components.estimators.BaseClassifier), 616

predict\_proba() (evalml.pipelines.components.estimators.BaseRegressor), 619

predict\_proba() (evalml.pipelines.components.estimators.CatBoostClassifier), 621

predict\_proba() (evalml.pipelines.components.estimators.CatBoostRegressor), 624

predict\_proba() (evalml.pipelines.components.estimators.classifiers.base\_line\_classifier), 473

predict\_proba() (evalml.pipelines.components.estimators.classifiers.BaseLineClassifier), 507

predict\_proba() (evalml.pipelines.components.estimators.classifiers.catboost\_classifier), 476

predict\_proba() (evalml.pipelines.components.estimators.classifiers.CatBoostClassifier), 509

predict\_proba() (evalml.pipelines.components.estimators.classifiers.decision\_tree\_classifier), 479

predict\_proba() (evalml.pipelines.components.estimators.classifiers.DecisionTreeClassifier), 512

predict\_proba() (evalml.pipelines.components.estimators.classifiers.elastic\_net\_classifier), 482

predict\_proba() (evalml.pipelines.components.estimators.classifiers.ElasticNetClassifier), 515

predict\_proba() (evalml.pipelines.components.estimators.classifiers.elo\_classifier), 486

predict\_proba() (evalml.pipelines.components.estimators.classifiers.ExtraTreesClassifier), 518

predict\_proba() (evalml.pipelines.components.estimators.classifiers.kneighbors\_classifier), 489

predict\_proba() (evalml.pipelines.components.estimators.classifiers.KNeighborsClassifier), 521

predict\_proba() (evalml.pipelines.components.estimators.classifiers.lightgbm\_classifier), 492

predict\_proba() (evalml.pipelines.components.estimators.classifiers.highGBMClassifier), 524

predict\_proba() (evalml.pipelines.components.estimators.classifiers.NetPiperRegressor), 495

predict\_proba() (evalml.pipelines.components.estimators.classifiers.StackedEnsembleClassifier), 527

predict\_proba() (evalml.pipelines.components.estimators.classifiers.StackedEnsembleClassifier), 530

predict\_proba() (evalml.pipelines.components.estimators.classifiers.StackedEnsembleRegressor), 498

predict\_proba() (evalml.pipelines.components.estimators.classifiers.StackedEnsembleRegressor), 501

predict\_proba() (evalml.pipelines.components.estimators.classifiers.StackedEnsembleRegressor), 532

predict\_proba() (evalml.pipelines.components.estimators.classifiers.StackedEnsembleRegressor), 504

predict\_proba() (evalml.pipelines.components.estimators.classifiers.StackedEnsembleRegressor), 535

predict\_proba() (evalml.pipelines.components.estimators.DecisionTreeRegressor), 627

predict\_proba() (evalml.pipelines.components.estimators.DecisionTreeRegressor), 630

predict\_proba() (evalml.pipelines.components.estimators.ElasticNetClassifier), 633

predict\_proba() (evalml.pipelines.components.estimators.ElasticNetClassifier), 636

predict\_proba() (evalml.pipelines.components.estimators.Estimator), 638

predict\_proba() (evalml.pipelines.components.estimators.estimator.BaseClassifier), 610

predict\_proba() (evalml.pipelines.components.estimators.ExtraTreesClassifier), 641

predict\_proba() (evalml.pipelines.components.estimators.ExtraTreesClassifier), 644

predict\_proba() (evalml.pipelines.components.estimators.KNeighborsClassifier), 646

predict\_proba() (evalml.pipelines.components.estimators.LightGBMClassifier), 649

predict\_proba() (evalml.pipelines.components.estimators.LightGBMClassifier), 652

predict\_proba() (evalml.pipelines.components.estimators.LinearRegressor), 655

predict\_proba() (evalml.pipelines.components.estimators.LogisticRegressor), 657

predict\_proba() (evalml.pipelines.components.estimators.ProphetRegressor), 660

predict\_proba() (evalml.pipelines.components.estimators.RandomForestClassifier), 662

predict\_proba() (evalml.pipelines.components.estimators.RandomForestClassifier), 664

predict\_proba() (evalml.pipelines.components.estimators.regressors.classifiers.highGBMClassifier), 538

predict\_proba() (evalml.pipelines.components.estimators.regressors.classifiers.StackedEnsembleRegressor), 576

predict\_proba() (evalml.pipelines.components.estimators.regressors.classifiers.StackedEnsembleRegressor), 576



method), 541

predict\_proba() (evalml.pipelines.components.estimators.XGBoostClassifier method), 579

predict\_proba() (evalml.pipelines.components.estimators.XGBoostRegressor method), 544

predict\_proba() (evalml.pipelines.components.estimators.ExtraTreesClassifier method), 581

predict\_proba() (evalml.pipelines.components.estimators.KNeighborsClassifier method), 547

predict\_proba() (evalml.pipelines.components.estimators.LightGBMClassifier method), 584

predict\_proba() (evalml.pipelines.components.estimators.LightGBMRegressor method), 550

predict\_proba() (evalml.pipelines.components.estimators.LinearRegressor method), 587

predict\_proba() (evalml.pipelines.components.estimators.LogisticRegressionClassifier method), 553

predict\_proba() (evalml.pipelines.components.estimators.ProphetRegressor method), 590

predict\_proba() (evalml.pipelines.components.estimators.RandomForestClassifier method), 556

predict\_proba() (evalml.pipelines.components.estimators.RandomForestRegressor method), 593

predict\_proba() (evalml.pipelines.components.estimators.SklearnStackedEnsembleClassifier method), 559

predict\_proba() (evalml.pipelines.components.estimators.SklearnStackedEnsembleRegressor method), 595

predict\_proba() (evalml.pipelines.components.estimators.SVMClassifier method), 562

predict\_proba() (evalml.pipelines.components.estimators.SVMRegressor method), 598

predict\_proba() (evalml.pipelines.components.estimators.TimeSeriesBaselineEstimator method), 600

predict\_proba() (evalml.pipelines.components.estimators.WrappedSKClassifier method), 565

predict\_proba() (evalml.pipelines.components.estimators.XGBoostClassifier method), 568

predict\_proba() (evalml.pipelines.components.estimators.XGBoostRegressor method), 603

predict\_proba() (evalml.pipelines.components.estimators.XGBoostClassifier method), 570

predict\_proba() (evalml.pipelines.components.estimators.XGBoostRegressor method), 605

predict\_proba() (evalml.pipelines.components.estimators.XGBoostClassifier method), 573

predict\_proba() (evalml.pipelines.components.estimators.XGBoostRegressor method), 607

predict\_proba() (evalml.pipelines.components.estimators.XGBoostClassifier method), 667

predict\_proba() (evalml.pipelines.components.estimators.XGBoostRegressor method), 669

predict\_proba() (evalml.pipelines.components.estimators.XGBoostClassifier method), 671

predict\_proba() (evalml.pipelines.components.estimators.XGBoostClassifier method), 674

predict\_proba() (evalml.pipelines.components.estimators.XGBoostRegressor method), 677

predict\_proba() (evalml.pipelines.components.estimators.ExtraTreesClassifier method), 960

predict\_proba() (evalml.pipelines.components.ExtraTreesRegressor method), 963

predict\_proba() (evalml.pipelines.components.LightGBMClassifier method), 971

predict\_proba() (evalml.pipelines.components.LightGBMRegressor method), 973

predict\_proba() (evalml.pipelines.components.ElasticNetRegressor method), 976

predict\_proba() (evalml.pipelines.components.LinearRegressor method), 981

predict\_proba() (evalml.pipelines.components.LogisticRegressionClassifier method), 983

predict\_proba() (evalml.pipelines.components.ProphetRegressor method), 1000

predict\_proba() (evalml.pipelines.components.RandomForestClassifier method), 1002

predict\_proba() (evalml.pipelines.components.RandomForestRegressor method), 1004

predict\_proba() (evalml.pipelines.components.SklearnStackedEnsembleClassifier method), 1019

predict\_proba() (evalml.pipelines.components.SklearnStackedEnsembleRegressor method), 1021

predict\_proba() (evalml.pipelines.components.SVMClassifier method), 1033

predict\_proba() (evalml.pipelines.components.SVMRegressor method), 1035

predict\_proba() (evalml.pipelines.components.TimeSeriesBaselineEstimator method), 1044

predict\_proba() (evalml.pipelines.components.WrappedSKClassifier method), 912

predict\_proba() (evalml.pipelines.components.XGBoostClassifier method), 1053

predict\_proba() (evalml.pipelines.components.XGBoostRegressor method), 1056

predict\_proba() (evalml.pipelines.components.XGBoostClassifier method), 1118

predict\_proba() (evalml.pipelines.components.XGBoostRegressor method), 1120

predict\_proba() (evalml.pipelines.components.XGBoostClassifier method), 1128

predict\_proba() (evalml.pipelines.ElasticNetRegressor method), 1130

predict\_proba() (evalml.pipelines.Estimator method), 1132

predict\_proba() (evalml.pipelines.ExtraTreesClassifier method), 1135

predict\_proba() (evalml.pipelines.ExtraTreesRegressor method), 1138

predict\_proba() (evalml.pipelines.KNeighborsClassifier method), 1138

[method](#)), 1143  
[predict\\_proba\(\)](#) ([evalml.pipelines.LightGBMClassifier](#) [method](#)), 1146  
[predict\\_proba\(\)](#) ([evalml.pipelines.LightGBMRegressor](#) [method](#)), 1149  
[predict\\_proba\(\)](#) ([evalml.pipelines.LinearRegressor](#) [method](#)), 1151  
[predict\\_proba\(\)](#) ([evalml.pipelines.LogisticRegressionClassifier](#) [method](#)), 1153  
[predict\\_proba\(\)](#) ([evalml.pipelines.multiclass\\_classification\\_pipeline.MulticlassClassificationPipeline](#) [method](#)), 1073  
[predict\\_proba\(\)](#) ([evalml.pipelines.MulticlassClassificationPipeline](#) [method](#)), 1157  
[predict\\_proba\(\)](#) ([evalml.pipelines.ProphetRegressor](#) [method](#)), 1169  
[predict\\_proba\(\)](#) ([evalml.pipelines.RandomForestClassifier](#) [method](#)), 1171  
[predict\\_proba\(\)](#) ([evalml.pipelines.RandomForestRegressor](#) [method](#)), 1173  
[predict\\_proba\(\)](#) ([evalml.pipelines.SklearnStackedEnsembleClassifier](#) [method](#)), 1187  
[predict\\_proba\(\)](#) ([evalml.pipelines.SklearnStackedEnsembleRegressor](#) [method](#)), 1190  
[predict\\_proba\(\)](#) ([evalml.pipelines.SVMClassifier](#) [method](#)), 1194  
[predict\\_proba\(\)](#) ([evalml.pipelines.SVMRegressor](#) [method](#)), 1196  
[predict\\_proba\(\)](#) ([evalml.pipelines.time\\_series\\_classification\\_pipeline.TimeSeriesBinaryClassificationPipeline](#) [method](#)), 1089  
[predict\\_proba\(\)](#) ([evalml.pipelines.time\\_series\\_classification\\_pipeline.TimeSeriesClassificationPipeline](#) [method](#)), 1093  
[predict\\_proba\(\)](#) ([evalml.pipelines.time\\_series\\_classification\\_pipeline.TimeSeriesMulticlassClassificationPipeline](#) [method](#)), 1097  
[predict\\_proba\(\)](#) ([evalml.pipelines.TimeSeriesBinaryClassificationPipeline](#) [method](#)), 1203  
[predict\\_proba\(\)](#) ([evalml.pipelines.TimeSeriesClassificationPipeline](#) [method](#)), 1207  
[predict\\_proba\(\)](#) ([evalml.pipelines.TimeSeriesMulticlassClassificationPipeline](#) [method](#)), 1212  
[predict\\_proba\(\)](#) ([evalml.pipelines.XGBoostClassifier](#) [method](#)), 1220  
[predict\\_proba\(\)](#) ([evalml.pipelines.XGBoostRegressor](#) [method](#)), 1223  
[print\\_deps\(\)](#) (in module [evalml.utils.cli\\_utils](#)), 1252  
[print\\_info\(\)](#) (in module [evalml.utils.cli\\_utils](#)), 1252  
[print\\_sys\\_info\(\)](#) (in module [evalml.utils.cli\\_utils](#)), 1252  
[ProblemTypes](#) (class in [evalml.problem\\_types](#)), 1240  
[ProblemTypes](#) (class in [evalml.problem\\_types.problem\\_types](#)), 1236  
[ProphetRegressor](#) (class in [evalml.pipelines](#)), 1167  
[ProphetRegressor](#) (class in [evalml.pipelines.components](#)), 998  
[ProphetRegressor](#) (class in [evalml.pipelines.components.estimators](#)), 658  
[ProphetRegressor](#) (class in [evalml.pipelines.components.estimators.regressors](#)), 595  
[ProphetRegressor](#) (class in [evalml.pipelines.components.estimators.regressors.prophet\\_regressor](#)), 595  
[propose\(\)](#) ([evalml.tuners.grid\\_search\\_tuner.GridSearchTuner](#) [method](#)), 1243  
[propose\(\)](#) ([evalml.tuners.GridSearchTuner](#) [method](#)), 1247  
[propose\(\)](#) ([evalml.tuners.random\\_search\\_tuner.RandomSearchTuner](#) [method](#)), 1243  
[propose\(\)](#) ([evalml.tuners.RandomSearchTuner](#) [method](#)), 1248  
[propose\(\)](#) ([evalml.tuners.skopt\\_tuner.SKOptTuner](#) [method](#)), 1244  
[propose\(\)](#) ([evalml.tuners.SKOptTuner](#) [method](#)), 1249  
[propose\(\)](#) ([evalml.tuners.Tuner](#) [method](#)), 1250  
[propose\(\)](#) ([evalml.tuners.tuner.Tuner](#) [method](#)), 1245

## R

[R2](#) (class in [evalml.objectives](#)), 438  
[R2](#) (class in [evalml.objectives.standard\\_metrics](#)), 357  
[raise\\_error\\_callback\(\)](#) (in module [evalml.automl.callbacks](#)), 187  
[RandomForestClassifier](#) (class in [evalml.pipelines](#)), 1169  
[RandomForestClassifier](#) (class in [evalml.pipelines.components](#)), 1000  
[RandomForestClassifier](#) (class in [evalml.pipelines.components.estimators](#)), 660  
[RandomForestClassifier](#) (class in [evalml.pipelines.components.estimators.classifiers](#)), 527  
[RandomForestClassifier](#) (class in [evalml.pipelines.components.estimators.classifiers.rf\\_classifier](#)), 496  
[RandomForestRegressor](#) (class in [evalml.pipelines](#)), 1171  
[RandomForestRegressor](#) (class in [evalml.pipelines.components](#)), 1002  
[RandomForestRegressor](#) (class in [evalml.pipelines.components.estimators](#)), 662  
[RandomForestRegressor](#) (class in [evalml.pipelines.components.estimators.regressors](#)), 598  
[RandomForestRegressor](#) (class in [evalml.pipelines.components.estimators.regressors.rf\\_regressor](#)), 563  
[RandomSearchTuner](#) (class in [evalml.tuners](#)), 1247

RandomSearchTuner (class in *evalml.tuners.random\_search\_tuner*), 1242  
 rankings() (*evalml.automl.automl\_search.AutoMLSearch* property), 185  
 rankings() (*evalml.AutoMLSearch* property), 1266  
 Recall (class in *evalml.objectives*), 439  
 Recall (class in *evalml.objectives.standard\_metrics*), 359  
 RecallMacro (class in *evalml.objectives*), 441  
 RecallMacro (class in *evalml.objectives.standard\_metrics*), 361  
 RecallMicro (class in *evalml.objectives*), 443  
 RecallMicro (class in *evalml.objectives.standard\_metrics*), 362  
 RecallWeighted (class in *evalml.objectives*), 445  
 RecallWeighted (class in *evalml.objectives.standard\_metrics*), 364  
 register() (*evalml.pipelines.components.component\_base\_meta.ComponentBaseMeta* method), 910  
 register() (*evalml.pipelines.components.ComponentBaseMeta* method), 931  
 register() (*evalml.pipelines.components.transformers.encoders.one\_hot\_encoder.OneHotEncoderMeta* method), 691  
 register() (*evalml.pipelines.components.transformers.imputers.target\_imputer.TargetImputerMeta* method), 729  
 register() (*evalml.pipelines.pipeline\_meta.PipelineBaseMeta* method), 1079  
 register() (*evalml.pipelines.pipeline\_meta.TimeSeriesPipelineBaseMeta* method), 1080  
 register() (*evalml.utils.base\_meta.BaseMeta* method), 1251  
 RegressionObjective (class in *evalml.objectives*), 446  
 RegressionObjective (class in *evalml.objectives.regression\_objective*), 303  
 RegressionPipeline (class in *evalml.pipelines*), 1174  
 RegressionPipeline (class in *evalml.pipelines.regression\_pipeline*), 1080  
 results() (*evalml.automl.automl\_search.AutoMLSearch* property), 185  
 results() (*evalml.AutoMLSearch* property), 1266  
 results() (*evalml.AutoMLSearch* property), 195  
 RFClassifierSelectFromModel (class in *evalml.pipelines*), 1178  
 RFClassifierSelectFromModel (class in *evalml.pipelines.components*), 1005  
 RFClassifierSelectFromModel (class in *evalml.pipelines.components.transformers*), 872  
 RFClassifierSelectFromModel (class in *evalml.pipelines.components.transformers.feature\_selection*), 712  
 RFClassifierSelectFromModel (class in *evalml.pipelines.components.transformers.feature\_selection.rf\_classifier*), 703  
 RFRegressorSelectFromModel (class in *evalml.pipelines*), 1180  
 RFRegressorSelectFromModel (class in *evalml.pipelines.components*), 1008  
 RFRegressorSelectFromModel (class in *evalml.pipelines.components.transformers*), 875  
 RFRegressorSelectFromModel (class in *evalml.pipelines.components.transformers.feature\_selection*), 715  
 RFRegressorSelectFromModel (class in *evalml.pipelines.components.transformers.feature\_selection.rf\_regressor*), 707  
 roc\_curve() (in module *evalml.model\_understanding*), 286  
 roc\_curve() (in module *evalml.model\_understanding.graphs*), 272  
 RootMeanSquaredError (class in *evalml.objectives*), 448  
 RootMeanSquaredError (class in *evalml.objectives.regression\_metrics*), 366  
 RootMeanSquaredLogError (class in *evalml.objectives*), 450  
 RootMeanSquaredLogError (class in *evalml.objectives.standard\_metrics*), 367

## S

safe\_repr() (in module *evalml.utils*), 1261  
 safe\_repr() (in module *evalml.utils.gen\_utils*), 1255  
 SamplerBase (class in *evalml.preprocessing.data\_splitters*), 1229  
 SamplerBase (class in *evalml.preprocessing.data\_splitters.sampler\_base*), 1225  
 save() (*evalml.automl.automl\_search.AutoMLSearch* method), 185  
 save() (*evalml.AutoMLSearch* method), 1266  
 save() (*evalml.pipelines.ARIMARegressor* method), 1107  
 save() (*evalml.pipelines.binary\_classification\_pipeline.BinaryClassificationPipeline* method), 1060  
 save() (*evalml.pipelines.CatBoostClassifier* method), 1110  
 save() (*evalml.pipelines.CatBoostRegressor* method), 1112  
 save() (*evalml.pipelines.classification\_pipeline.ClassificationPipeline* method), 1066

<code>save()</code> ( <code>evalml.pipelines.components.ARIMAREgressor</code> method), 919	<code>save()</code> ( <code>evalml.pipelines.components.estimators.CatBoostClassifier</code> method), 622
<code>save()</code> ( <code>evalml.pipelines.components.BaselineClassifier</code> method), 921	<code>save()</code> ( <code>evalml.pipelines.components.estimators.CatBoostRegressor</code> method), 624
<code>save()</code> ( <code>evalml.pipelines.components.BaselineRegressor</code> method), 923	<code>save()</code> ( <code>evalml.pipelines.components.estimators.classifiers.baseline_classifier</code> method), 473
<code>save()</code> ( <code>evalml.pipelines.components.CatBoostClassifier</code> method), 926	<code>save()</code> ( <code>evalml.pipelines.components.estimators.classifiers.BaselineClassifier</code> method), 507
<code>save()</code> ( <code>evalml.pipelines.components.CatBoostRegressor</code> method), 928	<code>save()</code> ( <code>evalml.pipelines.components.estimators.classifiers.catboost_classifier</code> method), 476
<code>save()</code> ( <code>evalml.pipelines.components.component_base.ComponentBase</code> method), 909	<code>save()</code> ( <code>evalml.pipelines.components.estimators.classifiers.CatBoostClassifier</code> method), 509
<code>save()</code> ( <code>evalml.pipelines.components.ComponentBase</code> method), 930	<code>save()</code> ( <code>evalml.pipelines.components.estimators.classifiers.decision_tree_classifier</code> method), 479
<code>save()</code> ( <code>evalml.pipelines.components.DateTimeFeaturizer</code> method), 933	<code>save()</code> ( <code>evalml.pipelines.components.estimators.classifiers.DecisionTreeClassifier</code> method), 512
<code>save()</code> ( <code>evalml.pipelines.components.DecisionTreeClassifier</code> method), 936	<code>save()</code> ( <code>evalml.pipelines.components.estimators.classifiers.elasticnet_classifier</code> method), 483
<code>save()</code> ( <code>evalml.pipelines.components.DecisionTreeRegressor</code> method), 939	<code>save()</code> ( <code>evalml.pipelines.components.estimators.classifiers.ElasticNetClassifier</code> method), 515
<code>save()</code> ( <code>evalml.pipelines.components.DelayedFeatureTransformer</code> method), 941	<code>save()</code> ( <code>evalml.pipelines.components.estimators.classifiers.et_classifier.ElasticNetClassifier</code> method), 486
<code>save()</code> ( <code>evalml.pipelines.components.DFSTransformer</code> method), 944	<code>save()</code> ( <code>evalml.pipelines.components.estimators.classifiers.ExtraTreesClassifier</code> method), 518
<code>save()</code> ( <code>evalml.pipelines.components.DropColumns</code> method), 946	<code>save()</code> ( <code>evalml.pipelines.components.estimators.classifiers.kneighbors_classifier</code> method), 489
<code>save()</code> ( <code>evalml.pipelines.components.DropNullColumns</code> method), 948	<code>save()</code> ( <code>evalml.pipelines.components.estimators.classifiers.KNeighborsClassifier</code> method), 521
<code>save()</code> ( <code>evalml.pipelines.components.ElasticNetClassifier</code> method), 951	<code>save()</code> ( <code>evalml.pipelines.components.estimators.classifiers.lightgbm_classifier</code> method), 492
<code>save()</code> ( <code>evalml.pipelines.components.ElasticNetRegressor</code> method), 953	<code>save()</code> ( <code>evalml.pipelines.components.estimators.classifiers.LightGBMClassifier</code> method), 524
<code>save()</code> ( <code>evalml.pipelines.components.EmailFeaturizer</code> method), 955	<code>save()</code> ( <code>evalml.pipelines.components.estimators.classifiers.logistic_regression_classifier</code> method), 495
<code>save()</code> ( <code>evalml.pipelines.components.ensemble.sklearn_stacked_ensemble_classifier</code> method), 456	<code>save()</code> ( <code>evalml.pipelines.components.estimators.classifiers.LogisticRegressionClassifier</code> method), 527
<code>save()</code> ( <code>evalml.pipelines.components.ensemble.sklearn_stacked_ensemble_classifier</code> method), 459	<code>save()</code> ( <code>evalml.pipelines.components.estimators.classifiers.SklearnStackedEnsembleClassifier</code> method), 530
<code>save()</code> ( <code>evalml.pipelines.components.ensemble.sklearn_stacked_ensemble_classifier</code> method), 462	<code>save()</code> ( <code>evalml.pipelines.components.estimators.classifiers.SklearnStackedEnsembleRegressor</code> method), 498
<code>save()</code> ( <code>evalml.pipelines.components.ensemble.SklearnStackedEnsembleClassifier</code> method), 465	<code>save()</code> ( <code>evalml.pipelines.components.estimators.classifiers.svm_classifier</code> method), 501
<code>save()</code> ( <code>evalml.pipelines.components.ensemble.SklearnStackedEnsembleClassifier</code> method), 468	<code>save()</code> ( <code>evalml.pipelines.components.estimators.classifiers.SVMClassifier</code> method), 532
<code>save()</code> ( <code>evalml.pipelines.components.ensemble.SklearnStackedEnsembleClassifier</code> method), 471	<code>save()</code> ( <code>evalml.pipelines.components.estimators.classifiers.xgboost_classifier</code> method), 504
<code>save()</code> ( <code>evalml.pipelines.components.Estimator</code> method), 958	<code>save()</code> ( <code>evalml.pipelines.components.estimators.classifiers.XGBoostClassifier</code> method), 535
<code>save()</code> ( <code>evalml.pipelines.components.estimators.ARIMAREgressor</code> method), 614	<code>save()</code> ( <code>evalml.pipelines.components.estimators.DecisionTreeClassifier</code> method), 627
<code>save()</code> ( <code>evalml.pipelines.components.estimators.BaselineClassifier</code> method), 617	<code>save()</code> ( <code>evalml.pipelines.components.estimators.DecisionTreeRegressor</code> method), 630
<code>save()</code> ( <code>evalml.pipelines.components.estimators.BaselineRegressor</code> method), 619	<code>save()</code> ( <code>evalml.pipelines.components.estimators.ElasticNetClassifier</code> method), 633



[save \(\) \(evalml.pipelines.components.estimators.ElasticNetRegressor method\), 636](#)  
[save \(\) \(evalml.pipelines.components.estimators.Estimator method\), 638](#)  
[save \(\) \(evalml.pipelines.components.estimators.estimators.Estimator method\), 610](#)  
[save \(\) \(evalml.pipelines.components.estimators.ExtraTreesClassifier method\), 641](#)  
[save \(\) \(evalml.pipelines.components.estimators.ExtraTreesRegressor method\), 644](#)  
[save \(\) \(evalml.pipelines.components.estimators.KNeighborsClassifier method\), 647](#)  
[save \(\) \(evalml.pipelines.components.estimators.LightGBMClassifier method\), 650](#)  
[save \(\) \(evalml.pipelines.components.estimators.LightGBMRegressor method\), 652](#)  
[save \(\) \(evalml.pipelines.components.estimators.LinearRegressor method\), 655](#)  
[save \(\) \(evalml.pipelines.components.estimators.LogisticRegressionClassifier method\), 657](#)  
[save \(\) \(evalml.pipelines.components.estimators.ProphetRegressor method\), 660](#)  
[save \(\) \(evalml.pipelines.components.estimators.RandomForestClassifier method\), 662](#)  
[save \(\) \(evalml.pipelines.components.estimators.RandomForestRegressor method\), 665](#)  
[save \(\) \(evalml.pipelines.components.estimators.regressors.svm method\), 538](#)  
[save \(\) \(evalml.pipelines.components.estimators.regressors.ARIMARegressor method\), 577](#)  
[save \(\) \(evalml.pipelines.components.estimators.regressors.baseline method\), 541](#)  
[save \(\) \(evalml.pipelines.components.estimators.regressors.BaselineRegressor method\), 579](#)  
[save \(\) \(evalml.pipelines.components.estimators.regressors.xgboost method\), 544](#)  
[save \(\) \(evalml.pipelines.components.estimators.regressors.CatBoostRegressor method\), 582](#)  
[save \(\) \(evalml.pipelines.components.estimators.regressors.decision\\_tree method\), 547](#)  
[save \(\) \(evalml.pipelines.components.estimators.regressors.DecisionTreeRegressor method\), 584](#)  
[save \(\) \(evalml.pipelines.components.estimators.regressors.elasticnet method\), 550](#)  
[save \(\) \(evalml.pipelines.components.estimators.regressors.ElasticNetRegressor method\), 587](#)  
[save \(\) \(evalml.pipelines.components.estimators.regressors.linear method\), 553](#)  
[save \(\) \(evalml.pipelines.components.estimators.regressors.ExtraTreesRegressor method\), 590](#)  
[save \(\) \(evalml.pipelines.components.estimators.regressors.lightgbm method\), 556](#)  
[save \(\) \(evalml.pipelines.components.estimators.regressors.LightGBMRegressor method\), 593](#)  
[save \(\) \(evalml.pipelines.components.estimators.regressors.linear method\), 559](#)  
[save \(\) \(evalml.pipelines.components.estimators.regressors.LinearRegressor method\), 595](#)  
[save \(\) \(evalml.pipelines.components.estimators.regressors.prophet\\_regressor method\), 562](#)  
[save \(\) \(evalml.pipelines.components.estimators.regressors.ProphetRegressor method\), 598](#)  
[save \(\) \(evalml.pipelines.components.estimators.regressors.RandomForestRegressor method\), 600](#)  
[save \(\) \(evalml.pipelines.components.estimators.regressors.rf\\_regressor method\), 565](#)  
[save \(\) \(evalml.pipelines.components.estimators.regressors.svm\\_regressor method\), 568](#)  
[save \(\) \(evalml.pipelines.components.estimators.regressors.SVMRegressor method\), 603](#)  
[save \(\) \(evalml.pipelines.components.estimators.regressors.time\\_series\\_baseline method\), 570](#)  
[save \(\) \(evalml.pipelines.components.estimators.regressors.TimeSeriesBaselineRegressor method\), 605](#)  
[save \(\) \(evalml.pipelines.components.estimators.regressors.xgboost\\_regressor method\), 573](#)  
[save \(\) \(evalml.pipelines.components.estimators.regressors.XGBoostRegressor method\), 607](#)  
[save \(\) \(evalml.pipelines.components.estimators.SVMClassifier method\), 667](#)  
[save \(\) \(evalml.pipelines.components.estimators.SVMRegressor method\), 669](#)  
[save \(\) \(evalml.pipelines.components.estimators.TimeSeriesBaselineEstimator method\), 672](#)  
[save \(\) \(evalml.pipelines.components.estimators.XGBoostClassifier method\), 674](#)  
[save \(\) \(evalml.pipelines.components.estimators.XGBoostRegressor method\), 677](#)  
[save \(\) \(evalml.pipelines.components.ExtraTreesClassifier method\), 960](#)  
[save \(\) \(evalml.pipelines.components.ExtraTreesRegressor method\), 963](#)  
[save \(\) \(evalml.pipelines.components.Imputer method\), 968](#)  
[save \(\) \(evalml.pipelines.components.LightGBMClassifier method\), 974](#)  
[save \(\) \(evalml.pipelines.components.LightGBMRegressor method\), 976](#)  
[save \(\) \(evalml.pipelines.components.LinearDiscriminantAnalysis method\), 978](#)  
[save \(\) \(evalml.pipelines.components.LinearRegressor method\), 981](#)  
[save \(\) \(evalml.pipelines.components.LogisticRegressionClassifier method\), 984](#)

<code>save()</code> ( <code>evalml.pipelines.components.LogTransformer</code> method), 986	<code>save()</code> ( <code>evalml.pipelines.components.transformers.column_selectors.ColumnSelector</code> method), 825
<code>save()</code> ( <code>evalml.pipelines.components.LSA</code> method), 988	<code>save()</code> ( <code>evalml.pipelines.components.transformers.column_selectors.DropColumnSelector</code> method), 827
<code>save()</code> ( <code>evalml.pipelines.components.OneHotEncoder</code> method), 991	<code>save()</code> ( <code>evalml.pipelines.components.transformers.column_selectors.SelectColumnSelector</code> method), 829
<code>save()</code> ( <code>evalml.pipelines.components.PCA</code> method), 993	<code>save()</code> ( <code>evalml.pipelines.components.transformers.column_selectors.SelectColumnSelector</code> method), 831
<code>save()</code> ( <code>evalml.pipelines.components.PerColumnImputer</code> method), 995	<code>save()</code> ( <code>evalml.pipelines.components.transformers.DateTimeFeaturizer</code> method), 839
<code>save()</code> ( <code>evalml.pipelines.components.PolynomialDetrender</code> method), 997	<code>save()</code> ( <code>evalml.pipelines.components.transformers.DelayedFeatureTransformer</code> method), 842
<code>save()</code> ( <code>evalml.pipelines.components.ProphetRegressor</code> method), 1000	<code>save()</code> ( <code>evalml.pipelines.components.transformers.DFSTransformer</code> method), 844
<code>save()</code> ( <code>evalml.pipelines.components.RandomForestClassifier</code> method), 1002	<code>save()</code> ( <code>evalml.pipelines.components.transformers.dimensionality_reduction.ReducedDimensionalityTransformer</code> method), 679
<code>save()</code> ( <code>evalml.pipelines.components.RandomForestRegressor</code> method), 1005	<code>save()</code> ( <code>evalml.pipelines.components.transformers.dimensionality_reduction.ReducedDimensionalityTransformer</code> method), 684
<code>save()</code> ( <code>evalml.pipelines.components.RFClassifierSelectFromModel</code> method), 1007	<code>save()</code> ( <code>evalml.pipelines.components.transformers.dimensionality_reduction.ReducedDimensionalityTransformer</code> method), 687
<code>save()</code> ( <code>evalml.pipelines.components.RFRegressorSelectFromModel</code> method), 1010	<code>save()</code> ( <code>evalml.pipelines.components.transformers.dimensionality_reduction.ReducedDimensionalityTransformer</code> method), 682
<code>save()</code> ( <code>evalml.pipelines.components.SelectByType</code> method), 1012	<code>save()</code> ( <code>evalml.pipelines.components.transformers.DropColumns</code> method), 846
<code>save()</code> ( <code>evalml.pipelines.components.SelectColumns</code> method), 1014	<code>save()</code> ( <code>evalml.pipelines.components.transformers.DropNullColumns</code> method), 848
<code>save()</code> ( <code>evalml.pipelines.components.SimpleImputer</code> method), 1016	<code>save()</code> ( <code>evalml.pipelines.components.transformers.EmailFeaturizer</code> method), 850
<code>save()</code> ( <code>evalml.pipelines.components.SklearnStackedEnsembleClassifier</code> method), 1019	<code>save()</code> ( <code>evalml.pipelines.components.transformers.encoders.onehot_encoder.OneHotEncoder</code> method), 690
<code>save()</code> ( <code>evalml.pipelines.components.SklearnStackedEnsembleRegressor</code> method), 1022	<code>save()</code> ( <code>evalml.pipelines.components.transformers.encoders.OneHotEncoder</code> method), 697
<code>save()</code> ( <code>evalml.pipelines.components.SMOTENCOversampler</code> method), 1024	<code>save()</code> ( <code>evalml.pipelines.components.transformers.encoders.target_encoder.TargetEncoder</code> method), 694
<code>save()</code> ( <code>evalml.pipelines.components.SMOTENCOversampler</code> method), 1026	<code>save()</code> ( <code>evalml.pipelines.components.transformers.encoders.TargetEncoder</code> method), 700
<code>save()</code> ( <code>evalml.pipelines.components.SMOTEOversampler</code> method), 1028	<code>save()</code> ( <code>evalml.pipelines.components.transformers.feature_selection.FeatureSelector</code> method), 703
<code>save()</code> ( <code>evalml.pipelines.components.StandardScaler</code> method), 1030	<code>save()</code> ( <code>evalml.pipelines.components.transformers.feature_selection.FeatureSelector</code> method), 712
<code>save()</code> ( <code>evalml.pipelines.components.SVMClassifier</code> method), 1033	<code>save()</code> ( <code>evalml.pipelines.components.transformers.feature_selection.rf_classifier.RandomForestClassifier</code> method), 706
<code>save()</code> ( <code>evalml.pipelines.components.SVMRegressor</code> method), 1035	<code>save()</code> ( <code>evalml.pipelines.components.transformers.feature_selection.rf_regressor.RandomForestRegressor</code> method), 709
<code>save()</code> ( <code>evalml.pipelines.components.TargetEncoder</code> method), 1037	<code>save()</code> ( <code>evalml.pipelines.components.transformers.feature_selection.RFCFeatureSelector</code> method), 715
<code>save()</code> ( <code>evalml.pipelines.components.TargetImputer</code> method), 1039	<code>save()</code> ( <code>evalml.pipelines.components.transformers.feature_selection.RFRFeatureSelector</code> method), 717
<code>save()</code> ( <code>evalml.pipelines.components.TextFeaturizer</code> method), 1042	<code>save()</code> ( <code>evalml.pipelines.components.transformers.FeatureSelector</code> method), 852
<code>save()</code> ( <code>evalml.pipelines.components.TimeSeriesBaselineEstimator</code> method), 1044	<code>save()</code> ( <code>evalml.pipelines.components.transformers.Imputer</code> method), 855
<code>save()</code> ( <code>evalml.pipelines.components.Transformer</code> method), 1046	<code>save()</code> ( <code>evalml.pipelines.components.transformers.imputers.Imputer</code> method), 732

[save \(\) \(evalml.pipelines.components.transformers.imputers.simple\\_imputer \(evalml.pipelines.components.transformers.preprocessing.polynomial\\_saver method\), 720](#)  
[save \(\) \(evalml.pipelines.components.transformers.imputers.simple\\_imputer \(evalml.pipelines.components.transformers.preprocessing.polynomial\\_saver method\), 723](#)  
[save \(\) \(evalml.pipelines.components.transformers.imputers.simple\\_imputer \(evalml.pipelines.components.transformers.preprocessing.text\\_featurizer method\), 734](#)  
[save \(\) \(evalml.pipelines.components.transformers.imputers.simple\\_imputer \(evalml.pipelines.components.transformers.preprocessing.text\\_featurizer method\), 726](#)  
[save \(\) \(evalml.pipelines.components.transformers.imputers.simple\\_imputer \(evalml.pipelines.components.transformers.preprocessing.text\\_featurizer method\), 737](#)  
[save \(\) \(evalml.pipelines.components.transformers.imputers.simple\\_imputer \(evalml.pipelines.components.transformers.preprocessing.text\\_featurizer method\), 728](#)  
[save \(\) \(evalml.pipelines.components.transformers.imputers.simple\\_imputer \(evalml.pipelines.components.transformers.preprocessing.text\\_featurizer method\), 739](#)  
[save \(\) \(evalml.pipelines.components.transformers.LinearDiscriminantAnalysis \(evalml.pipelines.components.transformers.preprocessing.transformer method\), 857](#)  
[save \(\) \(evalml.pipelines.components.transformers.LogTransformer \(evalml.pipelines.components.transformers.preprocessing.URLFeaturizer method\), 859](#)  
[save \(\) \(evalml.pipelines.components.transformers.LSA save \(\) \(evalml.pipelines.components.transformers.RFClassifierSelectFromModel method\), 861](#)  
[save \(\) \(evalml.pipelines.components.transformers.OneHotEncoder \(evalml.pipelines.components.transformers.RFRegressorSelectFromModel method\), 864](#)  
[save \(\) \(evalml.pipelines.components.transformers.PCA save \(\) \(evalml.pipelines.components.transformers.samplers.base\\_sampler method\), 866](#)  
[save \(\) \(evalml.pipelines.components.transformers.PerColumnImputer \(evalml.pipelines.components.transformers.samplers.base\\_sampler method\), 869](#)  
[save \(\) \(evalml.pipelines.components.transformers.PolynomialDegree \(evalml.pipelines.components.transformers.samplers.oversampler method\), 871](#)  
[save \(\) \(evalml.pipelines.components.transformers.preprocessing.delayed\\_features\\_transformer \(evalml.pipelines.components.transformers.samplers.oversampler method\), 742](#)  
[save \(\) \(evalml.pipelines.components.transformers.preprocessing.delayed\\_features\\_transformer \(evalml.pipelines.components.transformers.samplers.oversampler method\), 770](#)  
[save \(\) \(evalml.pipelines.components.transformers.preprocessing.delayed\\_features\\_transformer \(evalml.pipelines.components.transformers.samplers.oversampler method\), 745](#)  
[save \(\) \(evalml.pipelines.components.transformers.preprocessing.delayed\\_features\\_transformer \(evalml.pipelines.components.transformers.samplers.oversampler method\), 773](#)  
[save \(\) \(evalml.pipelines.components.transformers.preprocessing.delayed\\_features\\_transformer \(evalml.pipelines.components.transformers.samplers.oversampler method\), 775](#)  
[save \(\) \(evalml.pipelines.components.transformers.preprocessing.delayed\\_features\\_transformer \(evalml.pipelines.components.transformers.samplers.oversampler method\), 747](#)  
[save \(\) \(evalml.pipelines.components.transformers.preprocessing.delayed\\_features\\_transformer \(evalml.pipelines.components.transformers.samplers.oversampler method\), 777](#)  
[save \(\) \(evalml.pipelines.components.transformers.preprocessing.delayed\\_features\\_transformer \(evalml.pipelines.components.transformers.samplers.oversampler method\), 779](#)  
[save \(\) \(evalml.pipelines.components.transformers.preprocessing.delayed\\_features\\_transformer \(evalml.pipelines.components.transformers.samplers.oversampler method\), 750](#)  
[save \(\) \(evalml.pipelines.components.transformers.preprocessing.delayed\\_features\\_transformer \(evalml.pipelines.components.transformers.samplers.oversampler method\), 752](#)  
[save \(\) \(evalml.pipelines.components.transformers.preprocessing.delayed\\_features\\_transformer \(evalml.pipelines.components.transformers.samplers.oversampler method\), 781](#)  
[save \(\) \(evalml.pipelines.components.transformers.preprocessing.delayed\\_features\\_transformer \(evalml.pipelines.components.transformers.samplers.oversampler method\), 783](#)  
[save \(\) \(evalml.pipelines.components.transformers.preprocessing.delayed\\_features\\_transformer \(evalml.pipelines.components.transformers.samplers.oversampler method\), 755](#)

`save()` (`evalml.pipelines.components.transformers.SMOTEOverSampler` method), 888  
`save()` (`evalml.pipelines.components.transformers.SMOTEOverSampler` method), 890  
`save()` (`evalml.pipelines.components.transformers.StandardScaler` method), 892  
`save()` (`evalml.pipelines.components.transformers.TargetEncoder` method), 895  
`save()` (`evalml.pipelines.components.transformers.TargetImputer` method), 897  
`save()` (`evalml.pipelines.components.transformers.TextFeaturizer` method), 899  
`save()` (`evalml.pipelines.components.transformers.Transformer` method), 902  
`save()` (`evalml.pipelines.components.transformers.transformer.TargetTransformer` method), 833  
`save()` (`evalml.pipelines.components.transformers.transformer.TargetTransformer` method), 836  
`save()` (`evalml.pipelines.components.transformers.Undersampler` method), 904  
`save()` (`evalml.pipelines.components.transformers.URLFeaturizer` method), 906  
`save()` (`evalml.pipelines.components.Undersampler` method), 1049  
`save()` (`evalml.pipelines.components.URLFeaturizer` method), 1051  
`save()` (`evalml.pipelines.components.XGBoostClassifier` method), 1053  
`save()` (`evalml.pipelines.components.XGBoostRegressor` method), 1056  
`save()` (`evalml.pipelines.DecisionTreeClassifier` method), 1118  
`save()` (`evalml.pipelines.DecisionTreeRegressor` method), 1121  
`save()` (`evalml.pipelines.DelayedFeatureTransformer` method), 1123  
`save()` (`evalml.pipelines.DFSTransformer` method), 1125  
`save()` (`evalml.pipelines.ElasticNetClassifier` method), 1128  
`save()` (`evalml.pipelines.ElasticNetRegressor` method), 1130  
`save()` (`evalml.pipelines.Estimator` method), 1133  
`save()` (`evalml.pipelines.ExtraTreesClassifier` method), 1135  
`save()` (`evalml.pipelines.ExtraTreesRegressor` method), 1138  
`save()` (`evalml.pipelines.FeatureSelector` method), 1140  
`save()` (`evalml.pipelines.KNeighborsClassifier` method), 1143  
`save()` (`evalml.pipelines.LightGBMClassifier` method), 1146  
`save()` (`evalml.pipelines.LightGBMRegressor` method), 1149  
`save()` (`evalml.pipelines.LinearRegressor` method), 1151  
`save()` (`evalml.pipelines.LogisticRegressionClassifier` method), 1154  
`save()` (`evalml.pipelines.multiclass_classification_pipeline.MulticlassClassifier` method), 1158  
`save()` (`evalml.pipelines.MulticlassClassificationPipeline` method), 1160  
`save()` (`evalml.pipelines.OneHotEncoder` method), 1163  
`save()` (`evalml.pipelines.PerColumnImputer` method), 1167  
`save()` (`evalml.pipelines.pipeline_base.PipelineBase` method), 1169  
`save()` (`evalml.pipelines.PipelineBase` method), 1171  
`save()` (`evalml.pipelines.ProphetRegressor` method), 1174  
`save()` (`evalml.pipelines.RandomForestClassifier` method), 1177  
`save()` (`evalml.pipelines.RandomForestRegressor` method), 1180  
`save()` (`evalml.pipelines.regression_pipeline.RegressionPipeline` method), 1182  
`save()` (`evalml.pipelines.RegressionPipeline` method), 1185  
`save()` (`evalml.pipelines.RFClassifierSelectFromModel` method), 1187  
`save()` (`evalml.pipelines.RFRegressorSelectFromModel` method), 1190  
`save()` (`evalml.pipelines.SimpleImputer` method), 1192  
`save()` (`evalml.pipelines.SklearnStackedEnsembleClassifier` method), 1194  
`save()` (`evalml.pipelines.SklearnStackedEnsembleRegressor` method), 1197  
`save()` (`evalml.pipelines.StandardScaler` method), 1199  
`save()` (`evalml.pipelines.SVMClassifier` method), 1203  
`save()` (`evalml.pipelines.SVMRegressor` method), 1207  
`save()` (`evalml.pipelines.TargetEncoder` method), 1212  
`save()` (`evalml.pipelines.time_series_classification_pipelines.TimeSeriesClassifier` method), 1215  
`save()` (`evalml.pipelines.time_series_classification_pipelines.TimeSeriesClassifier` method), 1218  
`save()` (`evalml.pipelines.time_series_classification_pipelines.TimeSeriesClassifier` method), 1221  
`save()` (`evalml.pipelines.time_series_classification_pipelines.TimeSeriesClassifier` method), 1224  
`save()` (`evalml.pipelines.time_series_regression_pipeline.TimeSeriesRegressor` method), 1227  
`save()` (`evalml.pipelines.TimeSeriesBinaryClassificationPipeline` method), 1230  
`save()` (`evalml.pipelines.TimeSeriesClassificationPipeline` method), 1233  
`save()` (`evalml.pipelines.TimeSeriesMulticlassClassificationPipeline` method), 1236



[save\(\) \(evalml.pipelines.TimeSeriesRegressionPipeline method\), 1216](#)  
[save\(\) \(evalml.pipelines.Transformer method\), 1218](#)  
[save\(\) \(evalml.pipelines.XGBoostClassifier method\), 1221](#)  
[save\(\) \(evalml.pipelines.XGBoostRegressor method\), 1223](#)  
[save\\_plot\(\) \(in module evalml.utils\), 1261](#)  
[save\\_plot\(\) \(in module evalml.utils.gen\\_utils\), 1255](#)  
[scikit\\_learn\\_wrapped\\_estimator\(\) \(in module evalml.pipelines.components.utils\), 911](#)  
[score\(\) \(evalml.objectives.AccuracyBinary method\), 376](#)  
[score\(\) \(evalml.objectives.AccuracyMulticlass method\), 377](#)  
[score\(\) \(evalml.objectives.AUC method\), 379](#)  
[score\(\) \(evalml.objectives.AUCMacro method\), 381](#)  
[score\(\) \(evalml.objectives.AUCMicro method\), 383](#)  
[score\(\) \(evalml.objectives.AUCWeighted method\), 384](#)  
[score\(\) \(evalml.objectives.BalancedAccuracyBinary method\), 386](#)  
[score\(\) \(evalml.objectives.BalancedAccuracyMulticlass method\), 388](#)  
[score\(\) \(evalml.objectives.binary\\_classification\\_objective.BinaryClassificationObjective method\), 289](#)  
[score\(\) \(evalml.objectives.BinaryClassificationObjective method\), 390](#)  
[score\(\) \(evalml.objectives.cost\\_benefit\\_matrix.CostBenefitMatrix method\), 292](#)  
[score\(\) \(evalml.objectives.CostBenefitMatrix method\), 393](#)  
[score\(\) \(evalml.objectives.ExpVariance method\), 395](#)  
[score\(\) \(evalml.objectives.F1 method\), 397](#)  
[score\(\) \(evalml.objectives.F1Macro method\), 398](#)  
[score\(\) \(evalml.objectives.F1Micro method\), 400](#)  
[score\(\) \(evalml.objectives.F1Weighted method\), 402](#)  
[score\(\) \(evalml.objectives.fraud\\_cost.FraudCost method\), 295](#)  
[score\(\) \(evalml.objectives.FraudCost method\), 404](#)  
[score\(\) \(evalml.objectives.Gini method\), 407](#)  
[score\(\) \(evalml.objectives.lead\\_scoring.LeadScoring method\), 297](#)  
[score\(\) \(evalml.objectives.LeadScoring method\), 409](#)  
[score\(\) \(evalml.objectives.LogLossBinary method\), 411](#)  
[score\(\) \(evalml.objectives.LogLossMulticlass method\), 413](#)  
[score\(\) \(evalml.objectives.MAE method\), 415](#)  
[score\(\) \(evalml.objectives.MAPE method\), 416](#)  
[score\(\) \(evalml.objectives.MaxError method\), 418](#)  
[score\(\) \(evalml.objectives.MCCBinary method\), 420](#)  
[score\(\) \(evalml.objectives.MCCMulticlass method\), 422](#)  
[score\(\) \(evalml.objectives.MeanSquaredLogError method\), 423](#)  
[score\(\) \(evalml.objectives.MedianAE method\), 425](#)  
[score\(\) \(evalml.objectives.MSE method\), 426](#)  
[score\(\) \(evalml.objectives.multiclass\\_classification\\_objective.MulticlassClassificationObjective method\), 299](#)  
[score\(\) \(evalml.objectives.MulticlassClassificationObjective method\), 428](#)  
[score\(\) \(evalml.objectives.objective\\_base.ObjectiveBase method\), 302](#)  
[score\(\) \(evalml.objectives.ObjectiveBase method\), 430](#)  
[score\(\) \(evalml.objectives.Precision method\), 432](#)  
[score\(\) \(evalml.objectives.PrecisionMacro method\), 434](#)  
[score\(\) \(evalml.objectives.PrecisionMicro method\), 436](#)  
[score\(\) \(evalml.objectives.PrecisionWeighted method\), 437](#)  
[score\(\) \(evalml.objectives.R2 method\), 439](#)  
[score\(\) \(evalml.objectives.Recall method\), 441](#)  
[score\(\) \(evalml.objectives.RecallMacro method\), 443](#)  
[score\(\) \(evalml.objectives.RecallMicro method\), 444](#)  
[score\(\) \(evalml.objectives.RecallWeighted method\), 445](#)  
[score\(\) \(evalml.objectives.regression\\_objective.RegressionObjective method\), 304](#)  
[score\(\) \(evalml.objectives.RegressionObjective method\), 448](#)  
[score\(\) \(evalml.objectives.RootMeanSquaredError method\), 449](#)  
[score\(\) \(evalml.objectives.RootMeanSquaredLogError method\), 451](#)  
[score\(\) \(evalml.objectives.sensitivity\\_low\\_alert.SensitivityLowAlert method\), 307](#)  
[score\(\) \(evalml.objectives.SensitivityLowAlert method\), 453](#)  
[score\(\) \(evalml.objectives.standard\\_metrics.AccuracyBinary method\), 310](#)  
[score\(\) \(evalml.objectives.standard\\_metrics.AccuracyMulticlass method\), 312](#)  
[score\(\) \(evalml.objectives.standard\\_metrics.AUC method\), 314](#)  
[score\(\) \(evalml.objectives.standard\\_metrics.AUCMacro method\), 315](#)  
[score\(\) \(evalml.objectives.standard\\_metrics.AUCMicro method\), 317](#)  
[score\(\) \(evalml.objectives.standard\\_metrics.AUCWeighted method\), 319](#)  
[score\(\) \(evalml.objectives.standard\\_metrics.BalancedAccuracyBinary method\), 321](#)  
[score\(\) \(evalml.objectives.standard\\_metrics.BalancedAccuracyMulticlass method\), 322](#)  
[score\(\) \(evalml.objectives.standard\\_metrics.ExpVariance method\), 324](#)

`score()` (`evalml.objectives.standard_metrics.F1` method), 326  
`score()` (`evalml.objectives.standard_metrics.F1Macro` method), 328  
`score()` (`evalml.objectives.standard_metrics.F1Micro` method), 329  
`score()` (`evalml.objectives.standard_metrics.F1Weighted` method), 331  
`score()` (`evalml.objectives.standard_metrics.Gini` method), 333  
`score()` (`evalml.objectives.standard_metrics.LogLossBinary` method), 335  
`score()` (`evalml.objectives.standard_metrics.LogLossMulticlass` method), 337  
`score()` (`evalml.objectives.standard_metrics.MAE` method), 338  
`score()` (`evalml.objectives.standard_metrics.MAPE` method), 340  
`score()` (`evalml.objectives.standard_metrics.MaxError` method), 341  
`score()` (`evalml.objectives.standard_metrics.MCCBinary` method), 344  
`score()` (`evalml.objectives.standard_metrics.MCCMulticlass` method), 345  
`score()` (`evalml.objectives.standard_metrics.MeanSquaredLogError` method), 347  
`score()` (`evalml.objectives.standard_metrics.MedianAE` method), 348  
`score()` (`evalml.objectives.standard_metrics.MSE` method), 350  
`score()` (`evalml.objectives.standard_metrics.Precision` method), 352  
`score()` (`evalml.objectives.standard_metrics.PrecisionMacro` method), 354  
`score()` (`evalml.objectives.standard_metrics.PrecisionMicro` method), 355  
`score()` (`evalml.objectives.standard_metrics.PrecisionWeighted` method), 357  
`score()` (`evalml.objectives.standard_metrics.R2` method), 358  
`score()` (`evalml.objectives.standard_metrics.Recall` method), 360  
`score()` (`evalml.objectives.standard_metrics.RecallMacro` method), 362  
`score()` (`evalml.objectives.standard_metrics.RecallMicro` method), 364  
`score()` (`evalml.objectives.standard_metrics.RecallWeighted` method), 365  
`score()` (`evalml.objectives.standard_metrics.RootMeanSquaredError` method), 367  
`score()` (`evalml.objectives.standard_metrics.RootMeanSquaredLogError` method), 368  
`score()` (`evalml.objectives.time_series_regression_objective.TimeSeriesRegression` method), 370  
`score()` (`evalml.pipelines.binary_classification_pipeline.BinaryClassificationPipeline` method), 1061  
`score()` (`evalml.pipelines.classification_pipeline.ClassificationPipeline` method), 1066  
`score()` (`evalml.pipelines.components.utils.WrappedSKClassifier` method), 912  
`score()` (`evalml.pipelines.components.utils.WrappedSKRegressor` method), 913  
`score()` (`evalml.pipelines.multiclass_classification_pipeline.MulticlassClassificationPipeline` method), 1074  
`score()` (`evalml.pipelines.MulticlassClassificationPipeline` method), 1158  
`score()` (`evalml.pipelines.pipeline_base.PipelineBase` method), 1078  
`score()` (`evalml.pipelines.PipelineBase` method), 1167  
`score()` (`evalml.pipelines.regression_pipeline.RegressionPipeline` method), 1084  
`score()` (`evalml.pipelines.RegressionPipeline` method), 1177  
`score()` (`evalml.pipelines.time_series_classification_pipeline.TimeSeriesClassificationPipeline` method), 1089  
`score()` (`evalml.pipelines.time_series_classification_pipeline.TimeSeriesClassificationPipeline` method), 1093  
`score()` (`evalml.pipelines.time_series_classification_pipeline.TimeSeriesClassificationPipeline` method), 1098  
`score()` (`evalml.pipelines.time_series_regression_pipeline.TimeSeriesRegressionPipeline` method), 1102  
`score()` (`evalml.pipelines.TimeSeriesBinaryClassificationPipeline` method), 1203  
`score()` (`evalml.pipelines.TimeSeriesClassificationPipeline` method), 1208  
`score()` (`evalml.pipelines.TimeSeriesMulticlassClassificationPipeline` method), 1212  
`score()` (`evalml.pipelines.TimeSeriesRegressionPipeline` method), 1216  
`score_needs_proba()` (`evalml.objectives.binary_classification_objective.BinaryClassificationObjective` property), 289  
`score_needs_proba()` (`evalml.objectives.BinaryClassificationObjective` property), 391  
`score_needs_proba()` (`evalml.objectives.multiclass_classification_objective.MulticlassClassificationObjective` property), 300  
`score_needs_proba()` (`evalml.objectives.MulticlassClassificationObjective` property), 428  
`score_needs_proba()` (`evalml.objectives.objective_base.ObjectiveBase` property), 302  
`score_needs_proba()` (`evalml.objectives.ObjectiveBase` property), 410  
`score_needs_proba()` (`evalml.objectives.ObjectiveBase` property), 410

(`evalml.objectives.regression_objective.RegressionObjective` property), 304  
 score\_needs\_proba() (`evalml.objectives.RegressionObjective` property), 448  
 score\_needs\_proba() (`evalml.objectives.time_series_regression_objective.TimeSeriesRegressionObjective` property), 371  
 score\_pipeline() (in module `evalml.automl.engine.engine_base`), 173  
 score\_pipelines() (`evalml.automl.automl_search.AutoMLSearch` method), 185  
 score\_pipelines() (`evalml.automl.AutoMLSearch` method), 195  
 score\_pipelines() (`evalml.AutoMLSearch` method), 1266  
 search() (`evalml.automl.automl_search.AutoMLSearch` method), 185  
 search() (`evalml.automl.AutoMLSearch` method), 195  
 search() (`evalml.AutoMLSearch` method), 1266  
 search() (in module `evalml`), 1267  
 search() (in module `evalml.automl`), 197  
 search() (in module `evalml.automl.automl_search`), 186  
 search\_iteration\_plot() (`evalml.automl.pipeline_search_plots.PipelineSearchPlots` method), 188  
 SearchIterationPlot (class in `evalml.automl.pipeline_search_plots`), 188  
 SEED\_BOUNDS (in module `evalml.utils`), 1261  
 SEED\_BOUNDS (in module `evalml.utils.gen_utils`), 1256  
 SelectByType (class in `evalml.pipelines.components`), 1010  
 SelectByType (class in `evalml.pipelines.components.transformers`), 877  
 SelectByType (class in `evalml.pipelines.components.transformers.columns_selector`), 827  
 SelectColumns (class in `evalml.pipelines.components`), 1012  
 SelectColumns (class in `evalml.pipelines.components.transformers`), 879  
 SelectColumns (class in `evalml.pipelines.components.transformers.columns_selector`), 829  
 send\_data\_to\_cluster() (`evalml.automl.engine.dask_engine.DaskEngine` method), 169  
 send\_data\_to\_cluster() (`evalml.automl.engine.DaskEngine` method), 177  
 SensitivityLowAlert (class in `evalml.objectives`), 451  
 SensitivityLowAlert (class in `evalml.objectives.sensitivity_low_alert`), 305  
 SequentialComputation (class in `evalml.pipelines.components`), 174  
 SequentialEngine (class in `evalml.automl`), 197  
 SequentialEngine (class in `evalml.automl.engine`), 179  
 SequentialEngine (class in `evalml.automl.engine.sequential_engine`), 174  
 set\_fit() (`evalml.pipelines.components.component_base_meta.ComponentBaseMeta` class method), 910  
 set\_fit() (`evalml.pipelines.components.ComponentBaseMeta` class method), 931  
 set\_fit() (`evalml.pipelines.components.transformers.encoders.onehot_encoder.OneHotEncoder` class method), 691  
 set\_fit() (`evalml.pipelines.components.transformers.imputers.target_imputer.TargetImputer` class method), 729  
 set\_fit() (`evalml.pipelines.pipeline_meta.PipelineBaseMeta` class method), 1079  
 set\_fit() (`evalml.pipelines.pipeline_meta.TimeSeriesPipelineBaseMeta` class method), 1080  
 set\_fit() (`evalml.utils.base_meta.BaseMeta` class method), 1251  
 set\_params() (`evalml.pipelines.components.utils.WrappedSKClassifier` method), 913  
 set\_params() (`evalml.pipelines.components.utils.WrappedSKRegressor` method), 914  
 setup\_job\_log() (`evalml.automl.engine.cf_engine.CFEngine` static method), 167  
 setup\_job\_log() (`evalml.automl.engine.CFEngine` static method), 175  
 setup\_job\_log() (`evalml.automl.engine.dask_engine.DaskEngine` static method), 170  
 setup\_job\_log() (`evalml.automl.engine.DaskEngine` static method), 177  
 setup\_job\_log() (`evalml.automl.engine.engine_base.EngineBase` static method), 171  
 setup\_job\_log() (`evalml.automl.engine.EngineBase` static method), 178  
 setup\_job\_log() (`evalml.automl.engine.sequential_engine.SequentialEngine` static method), 175  
 setup\_job\_log() (`evalml.automl.engine.SequentialEngine` static method), 179  
 setup\_job\_log() (`evalml.automl.EngineBase` static method), 196  
 setup\_job\_log() (`evalml.automl.SequentialEngine` static method), 198  
 silent\_error\_callback() (in module `evalml.automl.callbacks`), 187

SimpleImputer (class in [evalml.pipelines](#)), 1183

SimpleImputer (class in [evalml.pipelines.components](#)), 1014

SimpleImputer (class in [evalml.pipelines.components.transformers](#)), 881

SimpleImputer (class in [evalml.pipelines.components.transformers.imputers](#)), 735

SimpleImputer (class in [evalml.pipelines.components.transformers.imputers.simple\\_imputer](#)), 724

SklearnStackedEnsembleBase (class in [evalml.pipelines.components.ensemble](#)), 463

SklearnStackedEnsembleBase (class in [evalml.pipelines.components.ensemble.sklearn\\_stacked\\_ensemble\\_base](#)), 454

SklearnStackedEnsembleClassifier (class in [evalml.pipelines](#)), 1185

SklearnStackedEnsembleClassifier (class in [evalml.pipelines.components](#)), 1017

SklearnStackedEnsembleClassifier (class in [evalml.pipelines.components.ensemble](#)), 465

SklearnStackedEnsembleClassifier (class in [evalml.pipelines.components.ensemble.sklearn\\_stacked\\_ensemble\\_classifier](#)), 457

SklearnStackedEnsembleRegressor (class in [evalml.pipelines](#)), 1188

SklearnStackedEnsembleRegressor (class in [evalml.pipelines.components](#)), 1019

SklearnStackedEnsembleRegressor (class in [evalml.pipelines.components.ensemble](#)), 468

SklearnStackedEnsembleRegressor (class in [evalml.pipelines.components.ensemble.sklearn\\_stacked\\_ensemble\\_regressor](#)), 460

SKOptTuner (class in [evalml.tuners](#)), 1248

SKOptTuner (class in [evalml.tuners.skopt\\_tuner](#)), 1244

SMOTENCOverSampler (class in [evalml.pipelines.components](#)), 1022

SMOTENCOverSampler (class in [evalml.pipelines.components.transformers](#)), 884

SMOTENCOverSampler (class in [evalml.pipelines.components.transformers.samplers](#)), 808

SMOTENCOverSampler (class in [evalml.pipelines.components.transformers.samplers.oversamplers](#)), 798

SMOTENOverSampler (class in [evalml.pipelines.components](#)), 1024

SMOTENOverSampler (class in [evalml.pipelines.components.transformers](#)), 886

SMOTENOverSampler (class in [evalml.pipelines.components.transformers.samplers](#)), 811

SMOTENOverSampler (class in [evalml.pipelines.components.transformers.samplers.oversamplers](#)), 800

SMOTEOverSampler (class in [evalml.pipelines.components](#)), 1026

SMOTEOverSampler (class in [evalml.pipelines.components.transformers](#)), 888

SMOTEOverSampler (class in [evalml.pipelines.components.transformers.samplers](#)), 813

SMOTEOverSampler (class in [evalml.pipelines.components.transformers.samplers.oversamplers](#)), 806

sparsity\_score() (evalml.data\_checks.sparsity\_data\_check.SparsityDataCheck static method), 220

sparsity\_score() (evalml.data\_checks.SparsityDataCheck static method), 241

SparsityDataCheck (class in [evalml.data\\_checks](#)), 241

SparsityDataCheck (class in [evalml.data\\_checks.sparsity\\_data\\_check](#)), 241

split() (evalml.preprocessing.data\_splitters.time\_series\_split.TimeSeriesSplit method), 1226

split() (evalml.preprocessing.data\_splitters.TimeSeriesSplit method), 1229

split() (evalml.preprocessing.data\_splitters.training\_validation\_split.TrainingValidationSplit method), 1227

split() (evalml.preprocessing.data\_splitters.TrainingValidationSplit method), 1230

split() (evalml.preprocessing.data\_splitters.TrainingValidationSplit method), 1234

split() (evalml.preprocessing.TrainingValidationSplit method), 1235

split\_data() (in module [evalml.preprocessing](#)), 1233

split\_data() (in module [evalml.preprocessing.utils](#)), 1232

StandardScaler (class in [evalml.pipelines](#)), 1190

StandardScaler (class in [evalml.pipelines.components](#)), 1029

StandardScaler (class in [evalml.pipelines.components.transformers](#)), 801

StandardScaler (class in [evalml.pipelines.components.transformers.scalers](#)), 821

StandardScaler (class in [evalml.pipelines.components.transformers.scalers.standard\\_scaler](#)), 818



[submit\(\)](#) ([evalml.automl.engine.cf\\_engine.CFClient](#) [method](#)), 166  
[submit\\_evaluation\\_job\(\)](#) ([evalml.automl.engine.cf\\_engine.CFEngine](#) [method](#)), 167  
[submit\\_evaluation\\_job\(\)](#) ([evalml.automl.engine.CFEngine](#) [method](#)), 176  
[submit\\_evaluation\\_job\(\)](#) ([evalml.automl.engine.dask\\_engine.DaskEngine](#) [method](#)), 170  
[submit\\_evaluation\\_job\(\)](#) ([evalml.automl.engine.DaskEngine](#) [method](#)), 177  
[submit\\_evaluation\\_job\(\)](#) ([evalml.automl.engine.engine\\_base.EngineBase](#) [method](#)), 171  
[submit\\_evaluation\\_job\(\)](#) ([evalml.automl.engine.EngineBase](#) [method](#)), 178  
[submit\\_evaluation\\_job\(\)](#) ([evalml.automl.engine.sequential\\_engine.SequentialEngine](#) [method](#)), 175  
[submit\\_evaluation\\_job\(\)](#) ([evalml.automl.engine.SequentialEngine](#) [method](#)), 179  
[submit\\_evaluation\\_job\(\)](#) ([evalml.automl.EngineBase](#) [method](#)), 196  
[submit\\_evaluation\\_job\(\)](#) ([evalml.automl.SequentialEngine](#) [method](#)), 198  
[submit\\_scoring\\_job\(\)](#) ([evalml.automl.engine.cf\\_engine.CFEngine](#) [method](#)), 168  
[submit\\_scoring\\_job\(\)](#) ([evalml.automl.engine.CFEngine](#) [method](#)), 176  
[submit\\_scoring\\_job\(\)](#) ([evalml.automl.engine.dask\\_engine.DaskEngine](#) [method](#)), 170  
[submit\\_scoring\\_job\(\)](#) ([evalml.automl.engine.DaskEngine](#) [method](#)), 177  
[submit\\_scoring\\_job\(\)](#) ([evalml.automl.engine.engine\\_base.EngineBase](#) [method](#)), 171  
[submit\\_scoring\\_job\(\)](#) ([evalml.automl.engine.EngineBase](#) [method](#)), 178  
[submit\\_scoring\\_job\(\)](#) ([evalml.automl.engine.sequential\\_engine.SequentialEngine](#) [method](#)), 175  
[submit\\_scoring\\_job\(\)](#) ([evalml.automl.engine.SequentialEngine](#) [method](#)), 179  
[submit\\_scoring\\_job\(\)](#) ([evalml.automl.EngineBase](#) [method](#)), 196  
[submit\\_scoring\\_job\(\)](#) ([evalml.automl.SequentialEngine](#) [method](#)), 198  
[summary\(\)](#) ([evalml.pipelines.binary\\_classification\\_pipeline.BinaryClassificationPipeline](#) [property](#)), 1061  
[summary\(\)](#) ([evalml.pipelines.classification\\_pipeline.ClassificationPipeline](#) [property](#)), 1066  
[summary\(\)](#) ([evalml.pipelines.multiclass\\_classification\\_pipeline.MulticlassClassificationPipeline](#) [property](#)), 1074  
[summary\(\)](#) ([evalml.pipelines.MulticlassClassificationPipeline](#) [property](#)), 1158  
[summary\(\)](#) ([evalml.pipelines.pipeline\\_base.PipelineBase](#) [property](#)), 1078  
[summary\(\)](#) ([evalml.pipelines.PipelineBase](#) [property](#)), 1167  
[summary\(\)](#) ([evalml.pipelines.regression\\_pipeline.RegressionPipeline](#) [property](#)), 1084  
[summary\(\)](#) ([evalml.pipelines.RegressionPipeline](#) [property](#)), 1178  
[summary\(\)](#) ([evalml.pipelines.time\\_series\\_classification\\_pipelines.TimeSeriesClassificationPipeline](#) [property](#)), 1089  
[summary\(\)](#) ([evalml.pipelines.time\\_series\\_classification\\_pipelines.TimeSeriesClassificationPipeline](#) [property](#)), 1089

property), 1093

summary() (evalml.pipelines.time\_series\_classification\_pipelines.TimeSeriesMulticlassClassificationPipeline, 286  
property), 1098

summary() (evalml.pipelines.time\_series\_regression\_pipeline.TimeSeriesRegressionPipeline, 273  
property), 1102

summary() (evalml.pipelines.TimeSeriesBinaryClassificationPipeline, 1234  
property), 1204

summary() (evalml.pipelines.TimeSeriesClassificationPipeline, 1232  
property), 1208

summary() (evalml.pipelines.TimeSeriesMulticlassClassificationPipeline, 242  
property), 1212

summary() (evalml.pipelines.TimeSeriesRegressionPipeline, 221  
property), 1216

supported\_problem\_types() (evalml.pipelines.components.ensemble.sklearn\_stacked\_ensemble\_base.SklearnStackedEnsembleBase, 456  
property), 456

supported\_problem\_types() (evalml.pipelines.components.ensemble.SklearnStackedEnsembleBase, 893  
property), 465

supported\_problem\_types() (evalml.pipelines.components.Estimator, 698  
property), 958

supported\_problem\_types() (evalml.pipelines.components.estimators.Estimator, 692  
property), 638

supported\_problem\_types() (evalml.pipelines.components.estimators.estimator.Estimator, 1038  
property), 610

supported\_problem\_types() (evalml.pipelines.Estimator property), 1133

SVMClassifier (class in evalml.pipelines), 1192

SVMClassifier (class in evalml.pipelines.components), 1031

SVMClassifier (class in evalml.pipelines.components.estimators), 665

SVMClassifier (class in evalml.pipelines.components.estimators.classifiers), 530

SVMClassifier (class in evalml.pipelines.components.estimators.classifiers.svm\_classifier), 499

SVMRegressor (class in evalml.pipelines), 1194

SVMRegressor (class in evalml.pipelines.components), 1033

SVMRegressor (class in evalml.pipelines.components.estimators), 667

SVMRegressor (class in evalml.pipelines.components.estimators.regressors), 600

SVMRegressor (class in evalml.pipelines.components.estimators.regressors.svm\_regressor), 566

T

t\_sne() (in module evalml.model\_understanding), 273

target\_distribution() (in module evalml.preprocessing), 1234

target\_distribution() (in module evalml.preprocessing.utils), 1232

TargetDistributionDataCheck (class in evalml.data\_checks), 242

TargetDistributionDataCheck (class in evalml.data\_checks.target\_distribution\_data\_check), 221

TargetEncoder (class in evalml.pipelines), 1197

TargetEncoder (class in evalml.pipelines.components), 1035

TargetEncoder (class in evalml.pipelines.components.transformers), 893

TargetEncoder (class in evalml.pipelines.components.transformers.encoders), 698

TargetEncoder (class in evalml.pipelines.components.transformers.encoders.target\_encoder), 692

TargetImputer (class in evalml.pipelines.components), 1038

TargetImputer (class in evalml.pipelines.components.transformers), 895

TargetImputer (class in evalml.pipelines.components.transformers.imputers), 737

TargetImputer (class in evalml.pipelines.components.transformers.imputers.target\_imputer), 727

TargetImputerMeta (class in evalml.pipelines.components.transformers.imputers.target\_imputer), 729

TargetLeakageDataCheck (class in evalml.data\_checks), 243

TargetLeakageDataCheck (class in evalml.data\_checks.target\_leakage\_data\_check), 222

TargetTransformer (class in evalml.pipelines.components.transformers.transformer), 831

TextFeaturizer (class in evalml.pipelines.components), 1040

TextFeaturizer (class in evalml.pipelines.components.transformers), 897

TextFeaturizer (class in evalml.pipelines.components.transformers.preprocessing), 566

786  
 TextFeaturizer (class in TimeSeriesSplit (class in *evalml.preprocessing*),  
*evalml.pipelines.components.transformers.preprocessing.text\_featurizer*),  
 758  
 TextTransformer (class in *evalml.preprocessing.data\_splitters*), 1229  
*evalml.pipelines.components.transformers.preprocessing.text\_transformer*  
 788  
 TextTransformer (class in *evalml.preprocessing.data\_splitters.time\_series\_split*),  
 1226  
*evalml.pipelines.components.transformers.preprocessing.text\_transformer*  
 761  
 threshold() (*evalml.pipelines.binary\_classification\_pipeline.BinaryClassificationPipeline*.  
*property*), 1061  
 threshold() (*evalml.pipelines.binary\_classification\_pipeline.BinaryClassificationPipeline*.  
*property*), 1062  
 threshold() (*evalml.pipelines.time\_series\_classification\_pipeline.TimeSeriesClassificationPipeline*.  
*property*), 1089  
 threshold() (*evalml.pipelines.TimeSeriesBinaryClassificationPipeline* (in module  
*evalml.data\_checks.data\_check\_action.DataCheckAction*  
*method*), 201  
 threshold() (*evalml.pipelines.time\_series\_classification\_pipeline.TimeSeriesClassificationPipeline*.  
*property*), 1062  
 threshold() (*evalml.pipelines.time\_series\_classification\_pipeline.TimeSeriesClassificationPipeline*.  
*property*), 1089  
 threshold() (*evalml.pipelines.TimeSeriesBinaryClassificationPipeline* (in module  
*evalml.data\_checks.data\_check\_action.DataCheckAction*  
*method*), 229  
 time\_elapsed() (in module *evalml.utils.logger*), to\_dict() (*evalml.data\_checks.DataCheckError*  
 1256  
*method*), 230  
 TimeSeriesBaselineEstimator (class in *evalml.pipelines.components*), 1042  
 TimeSeriesBaselineEstimator (class in *evalml.pipelines.components.estimators*), 669  
 TimeSeriesBaselineEstimator (class in *train\_and\_score\_pipeline* (in module  
*evalml.pipelines.components.estimators.regressors*), *evalml.automl.engine*), 179  
 603  
 TimeSeriesBaselineEstimator (class in *evalml.automl.engine.engine\_base*), 173  
*evalml.pipelines.components.estimators.regressors.time\_series\_baseline\_estimator* (in  
 568  
*evalml.automl.engine*), 180  
 TimeSeriesBinaryClassificationPipeline *train\_pipeline* (in module  
*evalml.pipelines*), 1199  
*evalml.automl.engine.engine\_base*), 173  
 TimeSeriesBinaryClassificationPipeline *train\_pipelines* (in module  
*evalml.pipelines.time\_series\_classification\_pipeline*), 1085  
*evalml.automl.automl\_search.AutoMLSearch*  
*method*), 185  
 TimeSeriesClassificationPipeline (class in *train\_pipelines* (in module  
*evalml.pipelines*), 1204  
*evalml.automl.AutoMLSearch*  
*method*), 196  
 TimeSeriesClassificationPipeline (class in *train\_pipelines* (in module  
*evalml.pipelines.time\_series\_classification\_pipeline*), 1089  
*evalml.AutoMLSearch*  
*method*), 1266  
 TrainingValidationSplit (class in  
*evalml.preprocessing*), 1234  
 TrainingValidationSplit (class in  
*evalml.preprocessing.data\_splitters*), 1230  
 TimeSeriesMulticlassClassificationPipeline (class in *train\_pipelines* (in module  
*evalml.pipelines.time\_series\_classification\_pipeline*), 1094  
*evalml.preprocessing.data\_splitters.training\_validation\_split*),  
 1227  
 TimeSeriesPipelineBaseMeta (class in *evalml.pipelines.pipeline\_meta*), 1079  
 transform() (*evalml.pipelines.components.DateTimeFeaturizer*  
*method*), 933  
 TimeSeriesRegressionObjective (class in *evalml.objectives.time\_series\_regression\_objective*),  
 369  
 transform() (*evalml.pipelines.components.DelayedFeatureTransformer*  
*method*), 942  
 TimeSeriesRegressionPipeline (class in *transform* (in module  
*evalml.pipelines*), 1212  
*method*), 944  
 TimeSeriesRegressionPipeline (class in *transform* (in module  
*evalml.pipelines.time\_series\_regression\_pipeline*),  
*method*), 946

`transform()` (`evalml.pipelines.components.DropNullColumns` method), 948  
`transform()` (`evalml.pipelines.components.EmailFeaturizer` method), 955  
`transform()` (`evalml.pipelines.components.FeatureSelector` method), 966  
`transform()` (`evalml.pipelines.components.Imputer` method), 968  
`transform()` (`evalml.pipelines.components.LinearDiscriminantAnalysis` method), 978  
`transform()` (`evalml.pipelines.components.LogTransformer` method), 986  
`transform()` (`evalml.pipelines.components.LSA` method), 988  
`transform()` (`evalml.pipelines.components.OneHotEncoder` method), 991  
`transform()` (`evalml.pipelines.components.PCA` method), 993  
`transform()` (`evalml.pipelines.components.PerColumnImputer` method), 995  
`transform()` (`evalml.pipelines.components.PolynomialDegreeReducer` method), 998  
`transform()` (`evalml.pipelines.components.RFClassifierSelectFromModel` method), 1007  
`transform()` (`evalml.pipelines.components.RFRegressorSelectFromModel` method), 1010  
`transform()` (`evalml.pipelines.components.SelectByType` method), 1012  
`transform()` (`evalml.pipelines.components.SelectColumns` method), 1014  
`transform()` (`evalml.pipelines.components.SimpleImputer` method), 1016  
`transform()` (`evalml.pipelines.components.SMOTEOverSampler` method), 1024  
`transform()` (`evalml.pipelines.components.SMOTEOverSampler` method), 1026  
`transform()` (`evalml.pipelines.components.SMOTEOverSampler` method), 1028  
`transform()` (`evalml.pipelines.components.StandardScaler` method), 1030  
`transform()` (`evalml.pipelines.components.TargetEncoder` method), 1037  
`transform()` (`evalml.pipelines.components.TargetImputer` method), 1040  
`transform()` (`evalml.pipelines.components.TextFeaturizer` method), 1042  
`transform()` (`evalml.pipelines.components.Transformer` method), 1046  
`transform()` (`evalml.pipelines.components.transformer_column_selector` method), 825  
`transform()` (`evalml.pipelines.components.transformer_column_selector` method), 827  
`transform()` (`evalml.pipelines.components.transformer_column_selector` method), 829  
`transform()` (`evalml.pipelines.components.transformers.column_selector` method), 831  
`transform()` (`evalml.pipelines.components.transformers.DateTimeFeaturizer` method), 840  
`transform()` (`evalml.pipelines.components.transformers.DelayedFeaturizer` method), 842  
`transform()` (`evalml.pipelines.components.transformers.DFSTransformer` method), 844  
`transform()` (`evalml.pipelines.components.transformers.dimensionality_reduction` method), 679  
`transform()` (`evalml.pipelines.components.transformers.dimensionality_reduction` method), 684  
`transform()` (`evalml.pipelines.components.transformers.dimensionality_reduction` method), 687  
`transform()` (`evalml.pipelines.components.transformers.dimensionality_reduction` method), 682  
`transform()` (`evalml.pipelines.components.transformers.DropColumns` method), 846  
`transform()` (`evalml.pipelines.components.transformers.DropNullColumns` method), 848  
`transform()` (`evalml.pipelines.components.transformers.EmailFeaturizer` method), 850  
`transform()` (`evalml.pipelines.components.transformers.encoders.onehot` method), 690  
`transform()` (`evalml.pipelines.components.transformers.encoders.OneHotEncoder` method), 697  
`transform()` (`evalml.pipelines.components.transformers.encoders.target` method), 694  
`transform()` (`evalml.pipelines.components.transformers.encoders.TargetEncoder` method), 700  
`transform()` (`evalml.pipelines.components.transformers.feature_selection` method), 703  
`transform()` (`evalml.pipelines.components.transformers.feature_selection` method), 712  
`transform()` (`evalml.pipelines.components.transformers.feature_selection` method), 706  
`transform()` (`evalml.pipelines.components.transformers.feature_selection` method), 709  
`transform()` (`evalml.pipelines.components.transformers.feature_selection` method), 715  
`transform()` (`evalml.pipelines.components.transformers.feature_selection` method), 717  
`transform()` (`evalml.pipelines.components.transformers.FeatureSelector` method), 853  
`transform()` (`evalml.pipelines.components.transformers.Imputer` method), 855  
`transform()` (`evalml.pipelines.components.transformers.imputers.Imputer` method), 732  
`transform()` (`evalml.pipelines.components.transformers.imputers.imputer` method), 720  
`transform()` (`evalml.pipelines.components.transformers.imputers.per_column` method), 723  
`transform()` (`evalml.pipelines.components.transformers.imputers.PerColumnImputer` method), 735



transform() (evalml.pipelines.components.transformers.imputers.SimpleImputer method), 726

transform() (evalml.pipelines.components.transformers.imputers.SimpleImputer method), 737

transform() (evalml.pipelines.components.transformers.imputers.TargetEncoder method), 729

transform() (evalml.pipelines.components.transformers.imputers.TargetEncoder method), 739

transform() (evalml.pipelines.components.transformers.LinearDiscriminantAnalysis method), 857

transform() (evalml.pipelines.components.transformers.LogTransformer method), 859

transform() (evalml.pipelines.components.transformers.LSA method), 861

transform() (evalml.pipelines.components.transformers.OneHotEncoder method), 864

transform() (evalml.pipelines.components.transformers.PCA method), 867

transform() (evalml.pipelines.components.transformers.PolynomialPulverizer method), 869

transform() (evalml.pipelines.components.transformers.PolynomialDegree method), 871

transform() (evalml.pipelines.components.transformers.preprocessing.DelayedFeatureTransformer method), 742

transform() (evalml.pipelines.components.transformers.preprocessing.DelayedFeatureTransformer method), 770

transform() (evalml.pipelines.components.transformers.preprocessing.DelayedFeatureTransformer method), 745

transform() (evalml.pipelines.components.transformers.preprocessing.DelayedFeatureTransformer method), 773

transform() (evalml.pipelines.components.transformers.preprocessing.DelayedFeatureTransformer method), 775

transform() (evalml.pipelines.components.transformers.preprocessing.DropNullColumns method), 747

transform() (evalml.pipelines.components.transformers.preprocessing.DropNullColumns method), 777

transform() (evalml.pipelines.components.transformers.preprocessing.EvalMLPipeline method), 779

transform() (evalml.pipelines.components.transformers.preprocessing.FeatureSelector method), 750

transform() (evalml.pipelines.components.transformers.preprocessing.LogTransform method), 753

transform() (evalml.pipelines.components.transformers.preprocessing.LogTransform method), 781

transform() (evalml.pipelines.components.transformers.preprocessing.LSA method), 783

transform() (evalml.pipelines.components.transformers.preprocessing.LSA method), 755

transform() (evalml.pipelines.components.transformers.preprocessing.PolynomialDegree method), 758

transform() (evalml.pipelines.components.transformers.preprocessing.PolynomialDegree method), 786

transform() (evalml.pipelines.components.transformers.preprocessing.TargetEncoder method), 760

transform() (evalml.pipelines.components.transformers.preprocessing.TargetEncoder method), 763

transform() (evalml.pipelines.components.transformers.preprocessing.TargetEncoder method), 788

transform() (evalml.pipelines.components.transformers.preprocessing.TargetEncoder method), 790

transform() (evalml.pipelines.components.transformers.preprocessing.TargetEncoder method), 765

transform() (evalml.pipelines.components.transformers.preprocessing.TargetEncoder method), 767

transform() (evalml.pipelines.components.transformers.preprocessing.TargetEncoder method), 792

transform() (evalml.pipelines.components.transformers.RFClassifier method), 874

transform() (evalml.pipelines.components.transformers.RFRegressor method), 877

transform() (evalml.pipelines.components.transformers.samplers.base method), 795

transform() (evalml.pipelines.components.transformers.samplers.base method), 797

transform() (evalml.pipelines.components.transformers.samplers.overs method), 800

transform() (evalml.pipelines.components.transformers.samplers.overs method), 802

transform() (evalml.pipelines.components.transformers.samplers.overs method), 805

transform() (evalml.pipelines.components.transformers.samplers.SMO method), 810

transform() (evalml.pipelines.components.transformers.samplers.SMO method), 813

transform() (evalml.pipelines.components.transformers.samplers.SMO method), 815

transform() (evalml.pipelines.components.transformers.samplers.SMO method), 818

transform() (evalml.pipelines.components.transformers.samplers.under method), 808

transform() (evalml.pipelines.components.transformers.scalers.standard method), 820

transform() (evalml.pipelines.components.transformers.scalers.StandardScaler method), 822

transform() (evalml.pipelines.components.transformers.SelectByType method), 879

transform() (evalml.pipelines.components.transformers.SelectColumns method), 881

transform() (evalml.pipelines.components.transformers.SimpleImputer method), 883

transform() (evalml.pipelines.components.transformers.SMOTEOver method), 886

transform() (evalml.pipelines.components.transformers.SMOTEOver method), 888

transform() (evalml.pipelines.components.transformers.SMOTEOver method), 890

transform() (evalml.pipelines.components.transformers.StandardScaler method), 892

transform() (evalml.pipelines.components.transformers.TargetEncoder, 190	evalml.automl.utils), 190
method), 895	Tuner (class in evalml.tuners), 1249
transform() (evalml.pipelines.components.transformers.TargetHotClass, 1245	method), 897
transform() (evalml.pipelines.components.transformers.TextFeaturizer	method), 899
method), 899	Undersampler (class in
transform() (evalml.pipelines.components.transformers.Transformers, 1046	evalml.pipelines.components), 1046
method), 902	Undersampler (class in
transform() (evalml.pipelines.components.transformers.transformers.TargetTransformer, 902	evalml.pipelines.components.transformers),
method), 833	902
transform() (evalml.pipelines.components.transformers.transformers.TargetTransformer, 902	method), 836
method), 836	evalml.pipelines.components.transformers.samplers),
transform() (evalml.pipelines.components.transformers.Undersampler	method), 904
method), 904	Undersampler (class in
transform() (evalml.pipelines.components.transformers.URLFeaturizer, 805	evalml.pipelines.components.transformers.samplers.undersample
method), 906	805
transform() (evalml.pipelines.components.Undersampler, 1049	method), 1049
method), 1049	uniqueness_score() (evalml.data_checks.uniqueness_data_check.UniquenessDataCheck
transform() (evalml.pipelines.components.URLFeaturizer, 1051	static method), 224
method), 1051	uniqueness_score() (evalml.data_checks.UniquenessDataCheck
transform() (evalml.pipelines.DelayedFeatureTransformer, 1123	static method), 245
method), 1123	UniquenessDataCheck (class in
transform() (evalml.pipelines.DFSTransformer, 1125	evalml.data_checks), 244
method), 1125	UniquenessDataCheck (class in
transform() (evalml.pipelines.FeatureSelector, 1141	evalml.data_checks.uniqueness_data_check),
method), 1141	224
transform() (evalml.pipelines.OneHotEncoder, 1161	update() (evalml.automl.pipeline_search_plots.SearchIterationPlot
method), 1161	method), 188
transform() (evalml.pipelines.PerColumnImputer, 1163	URLFeaturizer (class in
method), 1163	evalml.pipelines.components), 1049
transform() (evalml.pipelines.RFClassifierSelectFromModel, 1180	URLFeaturizer (class in
method), 1180	evalml.pipelines.components.transformers),
transform() (evalml.pipelines.RFRegressorSelectFromModel, 1183	904
method), 1183	URLFeaturizer (class in
transform() (evalml.pipelines.SimpleImputer, 1185	evalml.pipelines.components.transformers.preprocessing),
method), 1185	790
transform() (evalml.pipelines.StandardScaler, 1192	URLFeaturizer (class in
method), 1192	evalml.pipelines.components.transformers.preprocessing.transfor
transform() (evalml.pipelines.TargetEncoder, 1199	method), 766
method), 1199	
transform() (evalml.pipelines.Transformer method), 1218	
1218	
Transformer (class in evalml.pipelines), 1216	
Transformer (class in evalml.pipelines.components), 1044	
1044	
Transformer (class in evalml.pipelines.components.transformers), 900	
900	
Transformer (class in evalml.pipelines.components.transformers.transformer), 834	
834	
tune_binary_threshold() (in module evalml.automl), 198	
198	
tune_binary_threshold() (in module evalml.automl), 198	
198	

`validate()` (`evalml.data_checks.datetime_format_data_check.DateTimeFormatDataCheck`), 207  
`validate()` (`evalml.data_checks.datetime_nan_data_check.DateTimeNaNDataCheck`), 209  
`validate()` (`evalml.data_checks.DateTimeFormatDataCheck`), 233  
`validate()` (`evalml.data_checks.DateTimeNaNDataCheck`), 234  
`validate()` (`evalml.data_checks.default_data_checks.DefaultDataChecks`), 211  
`validate()` (`evalml.data_checks.DefaultDataChecks`), 235  
`validate()` (`evalml.data_checks.EmptyDataChecks`), 235  
`validate()` (`evalml.data_checks.highly_null_data_check.HighlyNullDataCheck`), 211  
`validate()` (`evalml.data_checks.HighlyNullDataCheck`), 236  
`validate()` (`evalml.data_checks.id_columns_data_check.IDColumnsDataCheck`), 213  
`validate()` (`evalml.data_checks.IDColumnsDataCheck`), 237  
`validate()` (`evalml.data_checks.invalid_targets_data_check.InvalidTargetDataCheck`), 214  
`validate()` (`evalml.data_checks.InvalidTargetDataCheck`), 238  
`validate()` (`evalml.data_checks.multicollinearity_data_check.MulticollinearityDataCheck`), 215  
`validate()` (`evalml.data_checks.MulticollinearityDataCheck`), 239  
`validate()` (`evalml.data_checks.natural_language_nan_data_check.NaturalLanguageNaNDataCheck`), 216  
`validate()` (`evalml.data_checks.NaturalLanguageNaNDataCheck`), 239  
`validate()` (`evalml.data_checks.no_variance_data_check.NoVarianceDataCheck`), 218  
`validate()` (`evalml.data_checks.NoVarianceDataCheck`), 240  
`validate()` (`evalml.data_checks.outliers_data_check.OutliersDataCheck`), 218  
`validate()` (`evalml.data_checks.OutliersDataCheck`), 240  
`validate()` (`evalml.data_checks.sparsity_data_check.SparsityDataCheck`), 220  
`validate()` (`evalml.data_checks.SparsityDataCheck`), 242  
`validate()` (`evalml.data_checks.target_distribution_data_check.TargetDistributionDataCheck`), 221  
`validate()` (`evalml.data_checks.target_leakage_data_check.TargetLeakageDataCheck`), 222  
`validate()` (`evalml.data_checks.TargetDistributionDataCheck`), 242  
`validate()` (`evalml.data_checks.TargetLeakageDataCheck`), 243

`validate_inputs()` (`evalml.data_checks.datetime_format_data_check.DateTimeFormatDataCheck`), 207  
`validate_inputs()` (`evalml.data_checks.datetime_nan_data_check.DateTimeNaNDataCheck`), 209  
`validate_inputs()` (`evalml.data_checks.DateTimeFormatDataCheck`), 225  
`validate_inputs()` (`evalml.data_checks.DateTimeNaNDataCheck`), 225  
`validate_inputs()` (`evalml.data_checks.default_data_checks.DefaultDataChecks`), 211  
`validate_inputs()` (`evalml.data_checks.DefaultDataChecks`), 378  
`validate_inputs()` (`evalml.data_checks.EmptyDataChecks`), 380  
`validate_inputs()` (`evalml.data_checks.highly_null_data_check.HighlyNullDataCheck`), 381  
`validate_inputs()` (`evalml.data_checks.HighlyNullDataCheck`), 383  
`validate_inputs()` (`evalml.data_checks.id_columns_data_check.IDColumnsDataCheck`), 384  
`validate_inputs()` (`evalml.data_checks.invalid_targets_data_check.InvalidTargetDataCheck`), 387  
`validate_inputs()` (`evalml.data_checks.InvalidTargetDataCheck`), 388  
`validate_inputs()` (`evalml.data_checks.multicollinearity_data_check.MulticollinearityDataCheck`), 389  
`validate_inputs()` (`evalml.data_checks.MulticollinearityDataCheck`), 289  
`validate_inputs()` (`evalml.data_checks.natural_language_nan_data_check.NaturalLanguageNaNDataCheck`), 391  
`validate_inputs()` (`evalml.data_checks.NaturalLanguageNaNDataCheck`), 391  
`validate_inputs()` (`evalml.data_checks.no_variance_data_check.NoVarianceDataCheck`), 292  
`validate_inputs()` (`evalml.data_checks.NoVarianceDataCheck`), 395  
`validate_inputs()` (`evalml.data_checks.outliers_data_check.OutliersDataCheck`), 395  
`validate_inputs()` (`evalml.data_checks.OutliersDataCheck`), 399  
`validate_inputs()` (`evalml.data_checks.sparsity_data_check.SparsityDataCheck`), 400  
`validate_inputs()` (`evalml.data_checks.SparsityDataCheck`), 400  
`validate_inputs()` (`evalml.data_checks.target_distribution_data_check.TargetDistributionDataCheck`), 400  
`validate_inputs()` (`evalml.data_checks.target_leakage_data_check.TargetLeakageDataCheck`), 400  
`validate_inputs()` (`evalml.data_checks.TargetDistributionDataCheck`), 295  
`validate_inputs()` (`evalml.data_checks.TargetLeakageDataCheck`), 404

```

validate_inputs() (evalml.objectives.Gini
method), 407
validate_inputs() (evalml.objectives.lead_scoring.LeadScoring
method), 297
validate_inputs() (evalml.objectives.LeadScoring
method), 409
validate_inputs() (evalml.objectives.LogLossBinary
method), 412
validate_inputs() (evalml.objectives.LogLossMulticlass
method), 413
validate_inputs() (evalml.objectives.MAE
method), 415
validate_inputs() (evalml.objectives.MAPE
method), 416
validate_inputs() (evalml.objectives.MaxError
method), 418
validate_inputs() (evalml.objectives.MCCBinary
method), 420
validate_inputs() (evalml.objectives.MCCMulticlass
method), 422
validate_inputs() (evalml.objectives.MeanSquaredLogError
method), 423
validate_inputs() (evalml.objectives.MedianAE
method), 425
validate_inputs() (evalml.objectives.MSE
method), 426
validate_inputs() (evalml.objectives.multiclass_classification_objective.MulticlassClassificationObjective
method), 300
validate_inputs() (evalml.objectives.MulticlassClassificationObjective
method), 428
validate_inputs() (evalml.objectives.objective_base.ObjectiveBase
method), 302
validate_inputs() (evalml.objectives.ObjectiveBase
method), 430
validate_inputs() (evalml.objectives.Precision
method), 433
validate_inputs() (evalml.objectives.PrecisionMacro
method), 434
validate_inputs() (evalml.objectives.PrecisionMicro
method), 436
validate_inputs() (evalml.objectives.PrecisionWeighted
method), 437
validate_inputs() (evalml.objectives.R2
method), 439
validate_inputs() (evalml.objectives.Recall
method), 441
validate_inputs() (evalml.objectives.RecallMacro
method), 443
validate_inputs() (evalml.objectives.RecallMicro
method), 444
validate_inputs() (evalml.objectives.RecallWeighted
method), 446
validate_inputs() (evalml.objectives.regression_objective.RegressionObjective
method), 304
validate_inputs() (evalml.objectives.RegressionObjective
method), 448
validate_inputs() (evalml.objectives.RootMeanSquaredError
method), 449
validate_inputs() (evalml.objectives.RootMeanSquaredLogError
method), 451
validate_inputs() (evalml.objectives.sensitivity_low_alert.SensitivityLowAlert
method), 307
validate_inputs() (evalml.objectives.SensitivityLowAlert
method), 453
validate_inputs() (evalml.objectives.standard_metrics.AccuracyBinary
method), 310
validate_inputs() (evalml.objectives.standard_metrics.AccuracyMulticlass
method), 312
validate_inputs() (evalml.objectives.standard_metrics.AUC
method), 314
validate_inputs() (evalml.objectives.standard_metrics.AUCMacro
method), 316
validate_inputs() (evalml.objectives.standard_metrics.AUCMicro
method), 317
validate_inputs() (evalml.objectives.standard_metrics.AUCWeighted
method), 319
validate_inputs() (evalml.objectives.standard_metrics.BalancedAccuracyBinary
method), 321
validate_inputs() (evalml.objectives.standard_metrics.BalancedAccuracyMulticlass
method), 323

```

<code>validate_inputs()</code> ( <code>evalml.objectives.standard_metrics.ExpVariance</code> <code>method</code> ), 324	<code>validate_inputs()</code> ( <code>evalml.objectives.standard_metrics.PrecisionMicro</code> <code>method</code> ), 355
<code>validate_inputs()</code> ( <code>evalml.objectives.standard_metrics.F1</code> <code>method</code> ), 326	<code>validate_inputs()</code> ( <code>evalml.objectives.standard_metrics.PrecisionWeighted</code> <code>method</code> ), 357
<code>validate_inputs()</code> ( <code>evalml.objectives.standard_metrics.F1Macro</code> <code>method</code> ), 328	<code>validate_inputs()</code> ( <code>evalml.objectives.standard_metrics.R2</code> <code>method</code> ), 359
<code>validate_inputs()</code> ( <code>evalml.objectives.standard_metrics.F1Micro</code> <code>method</code> ), 329	<code>validate_inputs()</code> ( <code>evalml.objectives.standard_metrics.Recall</code> <code>method</code> ), 361
<code>validate_inputs()</code> ( <code>evalml.objectives.standard_metrics.F1Weighted</code> <code>method</code> ), 331	<code>validate_inputs()</code> ( <code>evalml.objectives.standard_metrics.RecallMacro</code> <code>method</code> ), 362
<code>validate_inputs()</code> ( <code>evalml.objectives.standard_metrics.Gini</code> <code>method</code> ), 333	<code>validate_inputs()</code> ( <code>evalml.objectives.standard_metrics.RecallMicro</code> <code>method</code> ), 364
<code>validate_inputs()</code> ( <code>evalml.objectives.standard_metrics.LogLossBinary</code> <code>method</code> ), 335	<code>validate_inputs()</code> ( <code>evalml.objectives.standard_metrics.RecallWeighted</code> <code>method</code> ), 365
<code>validate_inputs()</code> ( <code>evalml.objectives.standard_metrics.LogLossMulticlass</code> <code>method</code> ), 337	<code>validate_inputs()</code> ( <code>evalml.objectives.standard_metrics.RootMeanSquaredError</code> <code>method</code> ), 367
<code>validate_inputs()</code> ( <code>evalml.objectives.standard_metrics.MAE</code> <code>method</code> ), 338	<code>validate_inputs()</code> ( <code>evalml.objectives.standard_metrics.RootMeanSquaredLogError</code> <code>method</code> ), 369
<code>validate_inputs()</code> ( <code>evalml.objectives.standard_metrics.MAPE</code> <code>method</code> ), 340	<code>validate_inputs()</code> ( <code>evalml.objectives.time_series_regression_objective.TimeSeriesRegression</code> <code>method</code> ), 371
<code>validate_inputs()</code> ( <code>evalml.objectives.standard_metrics.MaxError</code> <code>method</code> ), 342	<code>value()</code> ( <code>evalml.data_checks.data_check_action_code.DataCheckActionCode</code> <code>method</code> ), 202
<code>validate_inputs()</code> ( <code>evalml.objectives.standard_metrics.MCCBinary</code> <code>method</code> ), 344	<code>value()</code> ( <code>evalml.data_checks.data_check_message_code.DataCheckMessageCode</code> <code>method</code> ), 205
<code>validate_inputs()</code> ( <code>evalml.objectives.standard_metrics.MCCMulticlass</code> <code>method</code> ), 345	<code>value()</code> ( <code>evalml.data_checks.data_check_message_type.DataCheckMessageType</code> <code>method</code> ), 206
<code>validate_inputs()</code> ( <code>evalml.objectives.standard_metrics.MeanSquaredLogError</code> <code>method</code> ), 347	<code>value()</code> ( <code>evalml.data_checks.DataCheckActionCode</code> <code>method</code> ), 229
<code>validate_inputs()</code> ( <code>evalml.objectives.standard_metrics.MedianAE</code> <code>method</code> ), 349	<code>value()</code> ( <code>evalml.data_checks.DataCheckMessageCode</code> <code>method</code> ), 231
<code>validate_inputs()</code> ( <code>evalml.objectives.standard_metrics.MSE</code> <code>method</code> ), 350	<code>value()</code> ( <code>evalml.data_checks.DataCheckMessageType</code> <code>method</code> ), 232
<code>validate_inputs()</code> ( <code>evalml.objectives.standard_metrics.Precision</code> <code>method</code> ), 352	<code>value()</code> ( <code>evalml.exceptions.exceptions.PartialDependenceErrorCode</code> <code>method</code> ), 251
<code>validate_inputs()</code> ( <code>evalml.objectives.standard_metrics.PrecisionMacro</code> <code>method</code> ), 354	<code>value()</code> ( <code>evalml.exceptions.PartialDependenceErrorCode</code> <code>method</code> ), 253
	<code>value()</code> ( <code>evalml.model_family.model_family.ModelFamily</code> <code>method</code> ), 255
	<code>value()</code> ( <code>evalml.model_family.ModelFamily</code> <code>method</code> ), 256
	<code>value()</code> ( <code>evalml.model_understanding.prediction_explanations.explainer</code> <code>method</code> ), 260
	<code>value()</code> ( <code>evalml.problem_types.problem_types.ProblemTypes</code> <code>method</code> ), 1236



`value()` (*evalml.problem\_types.ProblemTypes*  
*method*), [1240](#)  
`visualize_decision_tree()` (*in module*  
*evalml.model\_understanding.graphs*), [273](#)

## W

`warning()` (*evalml.automl.engine.engine\_base.JobLogger*  
*method*), [173](#)  
`warning_not_unique_enough` (*in module*  
*evalml.data\_checks.uniqueness\_data\_check*),  
[225](#)  
`warning_too_unique` (*in module*  
*evalml.data\_checks.sparsity\_data\_check*),  
[221](#)  
`warning_too_unique` (*in module*  
*evalml.data\_checks.uniqueness\_data\_check*),  
[225](#)  
`WrappedSKClassifier` (*class in*  
*evalml.pipelines.components.utils*), [912](#)  
`WrappedSKRegressor` (*class in*  
*evalml.pipelines.components.utils*), [913](#)  
`write_to_logger()`  
(*evalml.automl.engine.engine\_base.JobLogger*  
*method*), [173](#)

## X

`XGBoostClassifier` (*class in evalml.pipelines*),  
[1218](#)  
`XGBoostClassifier` (*class in*  
*evalml.pipelines.components*), [1051](#)  
`XGBoostClassifier` (*class in*  
*evalml.pipelines.components.estimators*),  
[672](#)  
`XGBoostClassifier` (*class in*  
*evalml.pipelines.components.estimators.classifiers*),  
[532](#)  
`XGBoostClassifier` (*class in*  
*evalml.pipelines.components.estimators.classifiers.xgboost\_classifier*),  
[502](#)  
`XGBoostRegressor` (*class in evalml.pipelines*), [1221](#)  
`XGBoostRegressor` (*class in*  
*evalml.pipelines.components*), [1053](#)  
`XGBoostRegressor` (*class in*  
*evalml.pipelines.components.estimators*),  
[674](#)  
`XGBoostRegressor` (*class in*  
*evalml.pipelines.components.estimators.regressors*),  
[605](#)  
`XGBoostRegressor` (*class in*  
*evalml.pipelines.components.estimators.regressors.xgboost\_regressor*),  
[571](#)