

---

# **EvalML Documentation**

***Release 0.23.0***

**Alteryx, Inc.**

**Apr 21, 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>11</b>
<b>4</b>	<b>User Guide</b>	<b>41</b>
<b>5</b>	<b>API Reference</b>	<b>121</b>
<b>6</b>	<b>Release Notes</b>	<b>547</b>
	<b>Index</b>	<b>583</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
```

### **1.3 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) \* `graphviz` if you're using EvalML's plotting utilities

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 a patient has breast cancer. 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=1000, return_pandas=True)
```

```

      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
```

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 [Woodwork](#) data structure, providing us with flexibility to cast the data as necessary.

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

      X
```

```
[3]:
```

	Physical Type	Logical Type	Semantic Tag(s)
Data Column			
card_id	Int64	Integer	['numeric']
store_id	category	Categorical	['category']
datetime	datetime64[ns]	Datetime	[]
amount	Int64	Integer	['numeric']
currency	category	Categorical	['category']
customer_present	boolean	Boolean	[]
expiration_date	category	Datetime	[]
provider	category	Categorical	['category']
lat	category	Categorical	['category']
lng	category	Categorical	['category']
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 create a `DataTable` object using the [Woodwork project](#). Here, `split_data()` returns Woodwork data structures.

EvalML also accepts and works well with pandas `DataFrames`. But using the `DataTable` makes it easy to control how EvalML will treat each feature, as a numeric feature, a categorical feature, a text feature or other type of feature. Woodwork's `DataTable` includes features like inferring when a categorical feature should be treated as a text feature.

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...
```

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: xgboost, decision\_tree, catboost, linear\_model, lightgbm,  
↳extra\_trees, 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 F1: 0.000
```

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

Decision Tree Classifier w/ Imputer + DateTime Featurization Component + One Hot  
↳Encoder:

```
    Starting cross validation
    Finished cross validation - mean F1: 0.393
    High coefficient of variation (cv >= 0.2) within cross validation scores.
    Decision Tree Classifier w/ Imputer + DateTime Featurization Component + One
    ↳Hot Encoder may not perform as estimated on unseen data.
```

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

```
↳Encoder:
    Starting cross validation
    Finished cross validation - mean F1: 0.455
    High coefficient of variation (cv >= 0.2) within cross validation scores.
    Extra Trees Classifier w/ Imputer + DateTime Featurization Component + One
    ↳Hot Encoder may not perform as estimated on unseen data.
```

CatBoost Classifier w/ Imputer + DateTime Featurization Component:

```
    Starting cross validation
    Finished cross validation - mean F1: 0.738
```

Random Forest Classifier w/ Imputer + DateTime Featurization Component + One Hot

```
↳Encoder:
    Starting cross validation
    Finished cross validation - mean F1: 0.731
LightGBM Classifier w/ Imputer + DateTime Featurization Component + One Hot Encoder:
    Starting cross validation
    Finished cross validation - mean F1: 0.701
```

XGBoost Classifier w/ Imputer + DateTime Featurization Component + One Hot Encoder:

```
    Starting cross validation
    Finished cross validation - mean F1: 0.730
```

Elastic Net Classifier w/ Imputer + DateTime Featurization Component + One Hot

```
↳Encoder + Standard Scaler:
    Starting cross validation
```

(continues on next page)

(continued from previous page)

```

Finished cross validation - mean F1: 0.248
Logistic Regression Classifier w/ Imputer + DateTime Featurization Component + One_
↳ Hot Encoder + Standard Scaler:
  Starting cross validation
  Finished cross validation - mean F1: 0.319
  High coefficient of variation (cv >= 0.2) within cross validation scores.
  Logistic Regression Classifier w/ Imputer + DateTime Featurization Component_
↳ + One Hot Encoder + Standard Scaler may not perform as estimated on unseen data.

Search finished after 00:21
Best pipeline: CatBoost Classifier w/ Imputer + DateTime Featurization Component
Best pipeline F1: 0.738349

```

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  mean_cv_score  \
0    3  CatBoost Classifier w/ Imputer + DateTime Feat...    0.738349
1    4  Random Forest Classifier w/ Imputer + DateTime...    0.731328
2    6  XGBoost Classifier w/ Imputer + DateTime Featu...    0.730268
3    5  LightGBM Classifier w/ Imputer + DateTime Feat...    0.700767
4    2  Extra Trees Classifier w/ Imputer + DateTime F...    0.455416
5    1  Decision Tree Classifier w/ Imputer + DateTime...    0.393324
6    8  Logistic Regression Classifier w/ Imputer + Da...    0.318985
7    7  Elastic Net Classifier w/ Imputer + DateTime F...    0.247528
8    0      Mode Baseline Binary Classification Pipeline    0.000000

      standard_deviation_cv_score  validation_score  \
0                0.017274            0.757576
1                0.039761            0.774194
2                0.005309            0.733333
3                0.019222            0.711864
4                0.181772            0.253521
5                0.253968            0.686567
6                0.064165            0.385965
7                0.002861            0.249180
8                0.000000            0.000000

      percent_better_than_baseline  high_variance_cv  \
0                73.834901            False
1                73.132788            False
2                73.026820            False
3                70.076675            False
4                45.541613             True
5                39.332397             True
6                31.898487             True
7                24.752836            False
8                 0.000000            False

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

```

(continues on next page)

(continued from previous page)

```

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'}}

```

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)
```

```

*****
* CatBoost Classifier w/ Imputer + DateTime Featurization Component *
*****

Problem Type: binary
Model Family: CatBoost

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
3. CatBoost Classifier
  * n_estimators : 10
  * eta : 0.03
  * max_depth : 6
  * bootstrap_type : None
  * silent : True
  * allow_writing_files : False

Training
=====
Training for binary problems.
Objective to optimize binary classification pipeline thresholds for: <evalml.
  ↳ objectives.standard_metrics.F1 object at 0x7fa66f3e7880>
Total training time (including CV): 1.2 seconds

Cross Validation
-----

```

	F1	MCC	Binary	Log Loss	Binary	AUC	Precision	Balanced Accuracy
↳ Binary	Accuracy	Binary	Sensitivity	at Low Alert Rates	# Training	# Validation		
0	0.758	0.735	0.569	0.882	0.893			0.
↳ 822		0.940			0.000	375	267	
1	0.733	0.736	0.561	0.782	1.000			0.
↳ 789		0.940			0.000	375	267	
2	0.724	0.728	0.573	0.790	1.000			0.
↳ 784		0.940			0.000	380	266	
mean	0.738	0.733	0.568	0.818	0.964			0.
↳ 799		0.940			0.000	-	-	

(continues on next page)

(continued from previous page)

std	0.017	0.004	0.006	0.055	0.062	0.
↪021	0.000			0.000	-	-
coef of var	0.023	0.006	0.011	0.068	0.064	0.
↪026	0.000			inf	-	-

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}, 'DateTime_Featurization Component': {'features_to_extract': ['year', 'month', 'day_of_week', 'hour'], 'encode_as_categories': False}, 'CatBoost Classifier': {'n_estimators': 10, 'eta': 0.03, 'max_depth': 6, 'bootstrap_type': None, 'silent': True, 'allow_writing_files': False}}
```

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', 0.72727272727273)])
```

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=1000)
```

```

                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

```

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.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)

print(X.types)
```

Data 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	boolean	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'],
    max_batches=1,
    optimize_thresholds=True)
```

```
automl.search()
```

```
Generating pipelines to search over...
```

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

```
Optimizing for Fraud Cost.
```

(continues on next page)



(continued from previous page)

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: decision\_tree, xgboost, random\_forest, catboost, linear\_model,  
 ↪ extra\_trees, lightgbm

```
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: 0.160

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

Decision Tree Classifier w/ Imputer + One Hot Encoder:

Starting cross validation

Finished cross validation - mean Fraud Cost: 0.021

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

Decision Tree Classifier w/ Imputer + One Hot Encoder may not perform as  
 ↪estimated on unseen data.

Extra Trees Classifier w/ Imputer + One Hot Encoder:

Starting cross validation

Finished cross validation - mean Fraud Cost: 0.008

CatBoost Classifier w/ Imputer:

Starting cross validation

Finished cross validation - mean Fraud Cost: 0.008

Random Forest Classifier w/ Imputer + One Hot Encoder:

Starting cross validation

Finished cross validation - mean Fraud Cost: 0.008

LightGBM Classifier w/ Imputer + One Hot Encoder:

Starting cross validation

Finished cross validation - mean Fraud Cost: 0.009

XGBoost Classifier w/ Imputer + One Hot Encoder:

Starting cross validation

Finished cross validation - mean Fraud Cost: 0.008

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

Starting cross validation

Finished cross validation - mean Fraud Cost: 0.008

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

Starting cross validation

Finished cross validation - mean Fraud Cost: 0.008

Search finished after 00:14

Best pipeline: Extra Trees Classifier w/ Imputer + One Hot Encoder

Best pipeline Fraud Cost: 0.007866

## 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	mean_cv_score	\
0	2	Extra Trees Classifier w/ Imputer + One Hot En...	0.007866	
1	3	CatBoost Classifier w/ Imputer	0.007866	
2	4	Random Forest Classifier w/ Imputer + One Hot ...	0.007866	
3	6	XGBoost Classifier w/ Imputer + One Hot Encoder	0.007866	
4	7	Elastic Net Classifier w/ Imputer + One Hot En...	0.007866	
5	8	Logistic Regression Classifier w/ Imputer + On...	0.007866	
6	5	LightGBM Classifier w/ Imputer + One Hot Encoder	0.008587	
7	1	Decision Tree Classifier w/ Imputer + One Hot ...	0.021233	
8	0	Mode Baseline Binary Classification Pipeline	0.160045	

	standard_deviation_cv_score	validation_score	\
0	0.000065	0.007819	
1	0.000065	0.007819	
2	0.000065	0.007819	
3	0.000065	0.007819	
4	0.000065	0.007819	
5	0.000065	0.007819	
6	0.001291	0.007819	
7	0.004462	0.016656	
8	0.004896	0.163582	

	percent_better_than_baseline	high_variance_cv	\
0	15.217847	False	
1	15.217847	False	
2	15.217847	False	
3	15.217847	False	
4	15.217847	False	
5	15.217847	False	
6	15.145791	False	
7	13.881148	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'}}

```

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"])

*****
* CatBoost Classifier w/ Imputer *
*****

Problem Type: binary
Model Family: CatBoost

Pipeline Steps
=====
1. Imputer
    * categorical_impute_strategy : most_frequent
    * numeric_impute_strategy : mean
    * categorical_fill_value : None
    * numeric_fill_value : None
2. CatBoost Classifier
    * n_estimators : 10
    * eta : 0.03
    * max_depth : 6
    * bootstrap_type : None
    * silent : True
    * allow_writing_files : False

Training
=====
Training for binary problems.
Objective to optimize binary classification pipeline thresholds for: <evalml.
->objectives.fraud_cost.FraudCost object at 0x7ff0e24e9d60>
Total training time (including CV): 1.1 seconds

Cross Validation
-----
```

	Fraud Cost	AUC	F1	Precision	# Training	# Validation
0	0.008	0.857	0.249	0.142	375	267
1	0.008	0.795	0.249	0.142	375	267
2	0.008	0.768	0.244	0.139	380	266
mean	0.008	0.807	0.248	0.141	-	-
std	0.000	0.046	0.003	0.002	-	-
coef of var	0.008	0.057	0.012	0.013	-	-

### 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', 0.811046511627907), ('Fraud Cost', 0.007823596455165125)])
```

### 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,
                                optimize_thresholds=True)

automl_auc.search()
```

Generating pipelines to search over...

```
*****
* 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 9 pipelines.
Allowed model families: decision_tree, xgboost, random_forest, catboost, linear_model,
↳ extra_trees, lightgbm
```

```
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 *
*****
```

Decision Tree Classifier w/ Imputer + One Hot Encoder:  
   Starting cross validation  
   Finished cross validation - mean AUC: 0.656  
   High coefficient of variation (cv >= 0.2) within cross validation scores.  
   Decision Tree Classifier w/ Imputer + One Hot Encoder may not perform as  
 ↳ estimated on unseen data.

Extra Trees Classifier w/ Imputer + One Hot Encoder:  
   Starting cross validation  
   Finished cross validation - mean AUC: 0.803

CatBoost Classifier w/ Imputer:  
   Starting cross validation  
   Finished cross validation - mean AUC: 0.825

Random Forest Classifier w/ Imputer + One Hot Encoder:  
   Starting cross validation  
   Finished cross validation - mean AUC: 0.848

(continues on next page)

(continued from previous page)

```

LightGBM Classifier w/ Imputer + One Hot Encoder:
    Starting cross validation
    Finished cross validation - mean AUC: 0.835
XGBoost Classifier w/ Imputer + One Hot Encoder:
    Starting cross validation
    Finished cross validation - mean AUC: 0.842
Elastic Net Classifier w/ Imputer + One Hot Encoder + Standard Scaler:
    Starting cross validation
    Finished cross validation - mean AUC: 0.500
Logistic Regression Classifier w/ Imputer + One Hot Encoder + Standard Scaler:
    Starting cross validation
    Finished cross validation - mean AUC: 0.731

Search finished after 00:07
Best pipeline: Random Forest Classifier w/ Imputer + One Hot Encoder
Best pipeline AUC: 0.848079

```

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  mean_cv_score  \
0    4  Random Forest Classifier w/ Imputer + One Hot ...    0.848079
1    6    XGBoost Classifier w/ Imputer + One Hot Encoder    0.842374
2    5  LightGBM Classifier w/ Imputer + One Hot Encoder    0.835312
3    3          CatBoost Classifier w/ Imputer    0.824957
4    2  Extra Trees Classifier w/ Imputer + One Hot En...    0.802696
5    8  Logistic Regression Classifier w/ Imputer + On...    0.731433
6    1  Decision Tree Classifier w/ Imputer + One Hot ...    0.656104
7    0      Mode Baseline Binary Classification Pipeline    0.500000
8    7  Elastic Net Classifier w/ Imputer + One Hot En...    0.500000

      standard_deviation_cv_score  validation_score  \
0                0.032507            0.885544
1                0.038579            0.873592
2                0.011289            0.835785
3                0.061953            0.889221
4                0.041561            0.836474
5                0.041212            0.762353
6                0.178273            0.458113
7                0.000000            0.500000
8                0.000000            0.500000

      percent_better_than_baseline  high_variance_cv  \
0                34.807935            False
1                34.237444            False
2                33.531175            False
3                32.495688            False
4                30.269628            False
5                23.143277            False
6                15.610370             True
7                 0.000000            False
8                 0.000000            False

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

```

(continues on next page)

(continued from previous page)

```

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

```

```
[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', 0.8444767441860465), ('Fraud Cost', 0.03121096501479905)])
```

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

```
[14]: OrderedDict([('AUC', 0.811046511627907), ('Fraud Cost', 0.007823596455165125)])
```

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 noticable 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=1000,
      false_positives=-10
    )
```

### 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
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.DataTable(X, semantic_tags={'job': 'category'}, logical_types={'job':
↳'Categorical'})
y = ww.DataColumn(y)
X.types
```

```
[4]: Physical Type Logical Type Semantic Tag(s)
Data Column
```

(continues on next page)

(continued from previous page)

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)

print(X.types)
```

Data Column	Physical Type	Logical Type	Semantic Tag(s)
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 `AutoMLSearch(X_train=X_train, y_train=y_train, problem_type='binary')`. When we call `.search()`, the search for the best pipeline will begin.

```
[6]: from evalml.preprocessing import BalancedClassificationDataCVSplit
    # we choose to slightly sample classes by setting the sampling ratio low
    # we use the same splitter for both cases
    data_splitter = BalancedClassificationDataCVSplit(sampling_ratio=0.1, random_seed=0)

    automl = AutoMLSearch(X_train=X_train, y_train=y_train,
        problem_type='binary',
        data_splitter=data_splitter,
        objective=lead_scoring_objective,
        additional_objectives=['auc'],
        max_batches=1,
        optimize_thresholds=True)

    automl.search()
```

Generating pipelines to search over...

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

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

```
Using SequentialEngine to train and score pipelines.
```

(continues on next page)



(continued from previous page)

```
Searching up to 1 batches for a total of 9 pipelines.
```

```
Allowed model families: decision_tree, extra_trees, linear_model, xgboost, catboost,
↳random_forest, lightgbm
```

```
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 *
```

```
*****
```

```
Decision Tree Classifier w/ Imputer + One Hot Encoder:
```

```
  Starting cross validation
```

```
  Finished cross validation - mean Lead Scoring: 6.047
```

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

```
  Decision Tree Classifier w/ Imputer + One Hot Encoder may not perform as
```

```
↳estimated on unseen data.
```

```
Extra Trees Classifier w/ Imputer + One Hot Encoder:
```

```
  Starting cross validation
```

```
  Finished cross validation - mean Lead Scoring: 6.247
```

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

```
  Extra Trees Classifier w/ Imputer + One Hot Encoder may not perform as
```

```
↳estimated on unseen data.
```

```
CatBoost Classifier w/ Imputer:
```

```
  Starting cross validation
```

```
  Finished cross validation - mean Lead Scoring: 6.138
```

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

```
  CatBoost Classifier w/ Imputer may not perform as estimated on unseen data.
```

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

```
  Starting cross validation
```

```
  Finished cross validation - mean Lead Scoring: 7.666
```

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

```
  Random Forest Classifier w/ Imputer + One Hot Encoder may not perform as
```

```
↳estimated on unseen data.
```

```
LightGBM Classifier w/ Imputer + One Hot Encoder:
```

```
  Starting cross validation
```

```
  Finished cross validation - mean Lead Scoring: 10.820
```

```
  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.
```

```
XGBoost Classifier w/ Imputer + One Hot Encoder:
```

```
  Starting cross validation
```

```
  Finished cross validation - mean Lead Scoring: 3.927
```

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

```
  XGBoost Classifier w/ Imputer + One Hot Encoder may not perform as estimated
```

```
↳on unseen data.
```

```
Elastic Net Classifier w/ Imputer + One Hot Encoder + Standard Scaler:
```

```
  Starting cross validation
```

```
  Finished cross validation - mean Lead Scoring: 10.797
```

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

(continues on next page)

(continued from previous page)

```

Elastic Net Classifier w/ Imputer + One Hot Encoder + Standard Scaler may not
↳perform as estimated on unseen data.
Logistic Regression Classifier w/ Imputer + One Hot Encoder + Standard Scaler:
    Starting cross validation
    Finished cross validation - mean Lead Scoring: 5.003
    High coefficient of variation (cv >= 0.2) within cross validation scores.
    Logistic Regression Classifier w/ Imputer + One Hot Encoder + Standard Scaler
↳may not perform as estimated on unseen data.

Search finished after 00:13
Best pipeline: LightGBM Classifier w/ Imputer + One Hot Encoder
Best pipeline Lead Scoring: 10.819907

```

## 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  mean_cv_score  \
0    5  LightGBM Classifier w/ Imputer + One Hot Encoder      10.819907
1    7  Elastic Net Classifier w/ Imputer + One Hot En...      10.796914
2    4  Random Forest Classifier w/ Imputer + One Hot ...      7.665850
3    2  Extra Trees Classifier w/ Imputer + One Hot En...      6.246608
4    3          CatBoost Classifier w/ Imputer      6.137733
5    1  Decision Tree Classifier w/ Imputer + One Hot ...      6.046804
6    8  Logistic Regression Classifier w/ Imputer + On...      5.002742
7    6  XGBoost Classifier w/ Imputer + One Hot Encoder      3.927363
8    0      Mode Baseline Binary Classification Pipeline      0.000000

      standard_deviation_cv_score  validation_score  \
0                5.258483          14.980645
1                9.366362          16.741935
2                6.685478          14.961290
3                1.841152           7.800000
4                9.191255          16.741935
5                4.303758           9.329032
6                2.303951           7.354839
7                4.989703           9.541935
8                0.000000           0.000000

      percent_better_than_baseline  high_variance_cv  \
0                inf                True
1                inf                True
2                inf                True
3                inf                True
4                inf                True
5                inf                True
6                inf                True
7                inf                True
8                0.0                False

                                parameters
0  {'Imputer': {'categorical_impute_strategy': 'm...

```

(continues on next page)

(continued from previous page)

```

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`.

```
[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"])
```

```

*****
* LightGBM Classifier w/ Imputer + One Hot Encoder *
*****

Problem Type: binary
Model Family: LightGBM

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. LightGBM Classifier
    * boosting_type : gbd
    * learning_rate : 0.1
    * n_estimators : 100
    * max_depth : 0
    * num_leaves : 31
    * min_child_samples : 20
    * n_jobs : -1
    * bagging_freq : 0
    * bagging_fraction : 0.9

Training
=====

```

(continues on next page)

(continued from previous page)

```

Training for binary problems.
Objective to optimize binary classification pipeline thresholds for: <evalml.
↳objectives.lead_scoring.LeadScoring object at 0x7fd27eb68b20>
Total training time (including CV): 1.4 seconds

```

#### Cross Validation

```

-----
                Lead Scoring   AUC # Training # Validation
0                14.981 0.711      1,760      1,550
1                 4.910 0.707      1,760      1,550
2                12.569 0.653      1,760      1,549
mean              10.820 0.690           -           -
std                5.258 0.033           -           -
coef of var        0.486 0.047           -           -

```

### 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(X_holdout, y_holdout, objectives=["auc", lead_scoring_objective])
[10]: OrderedDict([('AUC', 0.675815956482321), ('Lead Scoring', 14.600171969045572)])

```

### 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',
                                     data_splitter=data_splitter,
                                     additional_objectives=[],
                                     max_batches=1,
                                     optimize_thresholds=True)

automl_auc.search()

Generating pipelines to search over...

*****
* 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 9 pipelines.
Allowed model families: decision_tree, extra_trees, linear_model, xgboost, catboost,
↳random_forest, lightgbm

```

```
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 *
*****

Decision Tree Classifier w/ Imputer + One Hot Encoder:
  Starting cross validation
  Finished cross validation - mean AUC: 0.624
Extra Trees Classifier w/ Imputer + One Hot Encoder:
  Starting cross validation
  Finished cross validation - mean AUC: 0.702
CatBoost Classifier w/ Imputer:
  Starting cross validation
  Finished cross validation - mean AUC: 0.578
Random Forest Classifier w/ Imputer + One Hot Encoder:
  Starting cross validation
  Finished cross validation - mean AUC: 0.702
LightGBM Classifier w/ Imputer + One Hot Encoder:
  Starting cross validation
  Finished cross validation - mean AUC: 0.692
XGBoost Classifier w/ Imputer + One Hot Encoder:
  Starting cross validation
  Finished cross validation - mean AUC: 0.717
Elastic Net Classifier w/ Imputer + One Hot Encoder + Standard Scaler:
  Starting cross validation
  Finished cross validation - mean AUC: 0.500
Logistic Regression Classifier w/ Imputer + One Hot Encoder + Standard Scaler:
  Starting cross validation
  Finished cross validation - mean AUC: 0.691

Search finished after 00:06
Best pipeline: XGBoost Classifier w/ Imputer + One Hot Encoder
Best pipeline AUC: 0.717491
```

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

```
[12]:
```

	id	pipeline_name	mean_cv_score \
0	6	XGBoost Classifier w/ Imputer + One Hot Encoder	0.717491
1	2	Extra Trees Classifier w/ Imputer + One Hot En...	0.701844
2	4	Random Forest Classifier w/ Imputer + One Hot ...	0.701549
3	5	LightGBM Classifier w/ Imputer + One Hot Encoder	0.691695
4	8	Logistic Regression Classifier w/ Imputer + On...	0.691426
5	1	Decision Tree Classifier w/ Imputer + One Hot ...	0.623687
6	3	CatBoost Classifier w/ Imputer	0.577711
7	0	Mode Baseline Binary Classification Pipeline	0.500000
8	7	Elastic Net Classifier w/ Imputer + One Hot En...	0.500000

	standard_deviation_cv_score	validation_score \
0	0.012589	0.730302

(continues on next page)

(continued from previous page)

```

1          0.024722          0.726947
2          0.023929          0.716135
3          0.029679          0.725931
4          0.017411          0.710468
5          0.011813          0.611314
6          0.023839          0.582849
7          0.000000          0.500000
8          0.000000          0.500000

    percent_better_than_baseline  high_variance_cv  \
0          21.749124          False
1          20.184406          False
2          20.154879          False
3          19.169462          False
4          19.142604          False
5          12.368704          False
6           7.771084          False
7           0.000000          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      {'Baseline Classifier': {'strategy': 'mode'}}
8  {'Imputer': {'categorical_impute_strategy': 'm...

```

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(X_holdout, y_holdout, objectives=["auc", lead_scoring_
→objective])

[14]: OrderedDict([('AUC', 0.6813387730432154),
                  ('Lead Scoring', -0.034393809114359415)])

```

When we optimize for AUC, we can see that the AUC score from this pipeline is better than 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 = X.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')
automl.search()

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

Using default limit of max_batches=1.

Generating pipelines to search over...

*****
* 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: random_forest, catboost, extra_trees, lightgbm, linear_model,
    ↳decision_tree, xgboost
```



```

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 *
*****

Decision Tree Classifier w/ Imputer + One Hot Encoder:
  Starting cross validation
  Finished cross validation - mean Log Loss Binary: 0.648
Extra Trees Classifier w/ Imputer + One Hot Encoder:
  Starting cross validation
  Finished cross validation - mean Log Loss Binary: 0.432
CatBoost Classifier w/ Imputer:
  Starting cross validation
  Finished cross validation - mean Log Loss Binary: 0.601
Random Forest Classifier w/ Imputer + One Hot Encoder:
  Starting cross validation
  Finished cross validation - mean Log Loss Binary: 0.425
LightGBM Classifier w/ Imputer + One Hot Encoder:
  Starting cross validation
  Finished cross validation - mean Log Loss Binary: 0.458
XGBoost Classifier w/ Imputer + One Hot Encoder:
  Starting cross validation
  Finished cross validation - mean Log Loss Binary: 0.445
Elastic Net Classifier w/ Imputer + One Hot Encoder + Standard Scaler:
  Starting cross validation
  Finished cross validation - mean Log Loss Binary: 0.579
Logistic Regression Classifier w/ Imputer + One Hot Encoder + Standard Scaler:
  Starting cross validation
  Finished cross validation - mean Log Loss Binary: 0.423

Search finished after 00:22
Best pipeline: Logistic Regression Classifier w/ Imputer + One Hot Encoder + Standard_
↪Scaler
Best pipeline Log Loss Binary: 0.423324

[3]: OrderedDict([('Log Loss Binary', 0.41624237859129104)])

```

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', 24.798864174159966)])

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

```

(continues on next page)

(continued from previous page)

```
print (total_profit_ll)
174658.40037860864
```

## 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)
    automl.search()

cbm_pipeline = automl.best_pipeline

Using default limit of max_batches=1.

Generating pipelines to search over...

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

Optimizing for Cost Benefit Matrix.
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: random_forest, catboost, extra_trees, lightgbm, linear_model,
    ↪decision_tree, xgboost

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 *
*****

Decision Tree Classifier w/ Imputer + One Hot Encoder:
  Starting cross validation
  Finished cross validation - mean Cost Benefit Matrix: 52.982
Extra Trees Classifier w/ Imputer + One Hot Encoder:
  Starting cross validation
  Finished cross validation - mean Cost Benefit Matrix: 58.459
CatBoost Classifier w/ Imputer:
  Starting cross validation
  Finished cross validation - mean Cost Benefit Matrix: 57.688
Random Forest Classifier w/ Imputer + One Hot Encoder:
  Starting cross validation
  Finished cross validation - mean Cost Benefit Matrix: 59.048
```

(continues on next page)

(continued from previous page)

```

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.
XGBoost Classifier w/ Imputer + One Hot Encoder:
    Starting cross validation
    Finished cross validation - mean Cost Benefit Matrix: 54.444
Elastic Net Classifier w/ Imputer + One Hot Encoder + Standard Scaler:
    Starting cross validation
    Finished cross validation - mean Cost Benefit Matrix: 32.657
Logistic Regression Classifier w/ Imputer + One Hot Encoder + Standard Scaler:
    Starting cross validation
    Finished cross validation - mean Cost Benefit Matrix: 59.311

Search finished after 00:28
Best pipeline: Logistic Regression Classifier w/ Imputer + One Hot Encoder + Standard_
    ↪ Scaler
Best pipeline Cost Benefit Matrix: 59.310679

```

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', 61.523899668717476)])

```

```

[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)

433312.8253667772

```

```

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

258654.42498816855

```

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)

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]: ham      0.750084
     spam     0.249916
     Name: Category, dtype: float64
```

### 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.

```
[4]: 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.

```
[5]: X_train.types
```

```
[5]:
```

	Physical Type	Logical Type	Semantic Tag(s)
Data 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.

```
[6]: 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...

*****
* 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: random_forest, linear_model, decision_tree, xgboost, extra_
    ↳ trees, catboost, lightgbm

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: 8.638

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

Decision Tree Classifier w/ Text Featurization Component:
```

(continues on next page)

(continued from previous page)

```
Starting cross validation
Finished cross validation - mean Log Loss Binary: 0.802
High coefficient of variation (cv >= 0.2) within cross validation scores.
Decision Tree Classifier w/ Text Featurization Component may not perform as
↳estimated on unseen data.
Extra Trees Classifier w/ Text Featurization Component:
Starting cross validation
Finished cross validation - mean Log Loss Binary: 0.252
High coefficient of variation (cv >= 0.2) within cross validation scores.
Extra Trees Classifier w/ Text Featurization Component may not perform as
↳estimated on unseen data.
CatBoost Classifier w/ Text Featurization Component:
Starting cross validation
Finished cross validation - mean Log Loss Binary: 0.526
Random Forest Classifier w/ Text Featurization Component:
Starting cross validation
Finished cross validation - mean Log Loss Binary: 0.155
High coefficient of variation (cv >= 0.2) within cross validation scores.
Random Forest Classifier w/ Text Featurization Component may not perform as
↳estimated on unseen data.
LightGBM Classifier w/ Text Featurization Component:
Starting cross validation
Finished cross validation - mean Log Loss Binary: 0.215
High coefficient of variation (cv >= 0.2) within cross validation scores.
LightGBM Classifier w/ Text Featurization Component may not perform as
↳estimated on unseen data.
XGBoost Classifier w/ Text Featurization Component:
Starting cross validation
Finished cross validation - mean Log Loss Binary: 0.179
High coefficient of variation (cv >= 0.2) within cross validation scores.
XGBoost Classifier w/ Text Featurization Component may not perform as
↳estimated on unseen data.
Elastic Net Classifier w/ Text Featurization Component + Standard Scaler:
Starting cross validation
Finished cross validation - mean Log Loss Binary: 0.543
Logistic Regression Classifier w/ Text Featurization Component + Standard Scaler:
Starting cross validation
Finished cross validation - mean Log Loss Binary: 0.214
High coefficient of variation (cv >= 0.2) within cross validation scores.
Logistic Regression Classifier w/ Text Featurization Component + Standard
↳Scaler may not perform as estimated on unseen data.

Search finished after 01:15
Best pipeline: Random Forest Classifier w/ Text Featurization Component
Best pipeline Log Loss Binary: 0.154849
```

## View rankings and select pipeline

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

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

```
[7]:
```

	id	pipeline_name	mean_cv_score	\
0	4	Random Forest Classifier w/ Text Featurization...	0.154849	
1	6	XGBoost Classifier w/ Text Featurization Compo...	0.178639	
2	8	Logistic Regression Classifier w/ Text Featuri...	0.214011	
3	5	LightGBM Classifier w/ Text Featurization Comp...	0.214580	
4	2	Extra Trees Classifier w/ Text Featurization C...	0.252206	
5	3	CatBoost Classifier w/ Text Featurization Comp...	0.526403	
6	7	Elastic Net Classifier w/ Text Featurization C...	0.542803	
7	1	Decision Tree Classifier w/ Text Featurization...	0.801766	
8	0	Mode Baseline Binary Classification Pipeline	8.638305	

	standard_deviation_cv_score	validation_score	\
0	0.050536	0.110302	
1	0.063117	0.113254	
2	0.051316	0.165624	
3	0.072715	0.136260	
4	0.059094	0.216198	
5	0.015739	0.512717	
6	0.012330	0.529152	
7	0.213655	0.555179	
8	0.025020	8.623860	

	percent_better_than_baseline	high_variance_cv	\
0	98.207418	True	
1	97.932010	True	
2	97.522538	True	
3	97.515944	True	
4	97.080377	True	
5	93.906174	False	
6	93.716325	False	
7	90.718481	True	
8	0.000000	False	

```

parameters
0 {'Random Forest Classifier': {'n_estimators': ...
1 {'XGBoost Classifier': {'eta': 0.1, 'max_depth...
2 {'Logistic Regression Classifier': {'penalty':...
3 {'LightGBM Classifier': {'boosting_type': 'gbd...
4 {'Extra Trees Classifier': {'n_estimators': 10...
5 {'CatBoost Classifier': {'n_estimators': 10, '...
6 {'Elastic Net Classifier': {'alpha': 0.5, 'l1_...
7 {'Decision Tree Classifier': {'criterion': 'gi...
8 {'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.

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

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

Problem Type: binary
Model Family: Random Forest

Pipeline Steps
=====
1. Text Featurization Component
2. Random Forest Classifier
    * n_estimators : 100
    * max_depth : 6
    * n_jobs : -1

Training
=====
Training for binary problems.
Total training time (including CV): 9.4 seconds

Cross Validation
-----

```

	Log Loss Binary	MCC Binary	AUC	Precision	F1	Balanced Accuracy
↪ Binary	Accuracy Binary	Sensitivity	at Low Alert Rates	# Training	# Validation	
0	0.110	0.895	0.987	0.938	0.921	0.
↪ 942	0.961			0.246	1,594	797
1	0.144	0.854	0.980	0.919	0.888	0.
↪ 917	0.946			0.246	1,594	797
2	0.210	0.783	0.962	0.839	0.837	0.
↪ 891	0.918			0.266	1,594	797
mean	0.155	0.844	0.977	0.899	0.882	0.
↪ 917	0.942			0.252	-	-
std	0.051	0.057	0.013	0.052	0.042	0.
↪ 026	0.022			0.011	-	-
coef of var	0.326	0.067	0.013	0.058	0.048	0.
↪ 028	0.023			0.045	-	-

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

[10]:

Notice above that there is a Text Featurization Component as the first step in the pipeline. The Woodwork DataTable passed in to AutoML search recognizes 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.

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

Accuracy Binary: 0.9732441471571907
```

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

### 3.4.4 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`.

```
[12]: 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)
```

```
[13]: 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...

*****
* 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: random_forest, linear_model, decision_tree, xgboost, extra_
      ↪trees, catboost, lightgbm

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: 8.638

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

(continues on next page)

(continued from previous page)

```

Decision Tree Classifier w/ Imputer + One Hot Encoder:
    Starting cross validation
    Finished cross validation - mean Log Loss Binary: 0.561
Extra Trees Classifier w/ Imputer + One Hot Encoder:
    Starting cross validation
    Finished cross validation - mean Log Loss Binary: 0.561
CatBoost Classifier w/ Imputer:
    Starting cross validation
    Finished cross validation - mean Log Loss Binary: 0.622
Random Forest Classifier w/ Imputer + One Hot Encoder:
    Starting cross validation
    Finished cross validation - mean Log Loss Binary: 0.561
LightGBM Classifier w/ Imputer + One Hot Encoder:
    Starting cross validation
    Finished cross validation - mean Log Loss Binary: 0.562
XGBoost Classifier w/ Imputer + One Hot Encoder:
    Starting cross validation
    Finished cross validation - mean Log Loss Binary: 0.562
Elastic Net Classifier w/ Imputer + One Hot Encoder + Standard Scaler:
    Starting cross validation
    Finished cross validation - mean Log Loss Binary: 0.563
Logistic Regression Classifier w/ Imputer + One Hot Encoder + Standard Scaler:
    Starting cross validation
    Finished cross validation - mean Log Loss Binary: 0.561

Search finished after 00:05
Best pipeline: Logistic Regression Classifier w/ Imputer + One Hot Encoder + Standard_
↳Scaler
Best pipeline Log Loss Binary: 0.560554

```

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

```
[14]: automl_no_text.rankings
```

```

[14]:   id      pipeline_name  mean_cv_score  \
0    8  Logistic Regression Classifier w/ Imputer + On...    0.560554
1    1  Decision Tree Classifier w/ Imputer + One Hot ...    0.561003
2    4  Random Forest Classifier w/ Imputer + One Hot ...    0.561179
3    2  Extra Trees Classifier w/ Imputer + One Hot En...    0.561247
4    6  XGBoost Classifier w/ Imputer + One Hot Encoder    0.562197
5    5  LightGBM Classifier w/ Imputer + One Hot Encoder    0.562451
6    7  Elastic Net Classifier w/ Imputer + One Hot En...    0.562556
7    3  CatBoost Classifier w/ Imputer                    0.622449
8    0  Mode Baseline Binary Classification Pipeline        8.638305

      standard_deviation_cv_score  validation_score  \
0                0.002239          0.558141
1                0.002716          0.558148
2                0.001985          0.559141
3                0.002123          0.559029
4                0.000707          0.561991
5                0.000796          0.561991
6                0.000774          0.562070
7                0.001186          0.622812
8                0.025020          8.623860

```

(continues on next page)

(continued from previous page)

```

percent_better_than_baseline  high_variance_cv  \
0          93.510838          False
1          93.505636          False
2          93.503593          False
3          93.502811          False
4          93.491811          False
5          93.488872          False
6          93.487658          False
7          92.794318          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'}}

```

```
[15]: 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.

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

```
[16]:
```

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

```

*****
* Logistic Regression Classifier w/ Imputer + One Hot Encoder + 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
    * 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

```

(continues on next page)

(continued from previous page)

```

* C : 1.0
* n_jobs : -1
* multi_class : auto
* solver : lbfgs

Training
=====
Training for binary problems.
Total training time (including CV): 0.4 seconds

Cross Validation
-----

```

	Log Loss Binary	MCC Binary	AUC	Precision	F1	Balanced Accuracy
→ Binary	Accuracy Binary	Sensitivity	at Low Alert Rates	# Training	# Validation	
0	0.558	0.061	0.508	1.000	0.010	0.
→ 503	0.752			0.000	1,594	797
1	0.561	0.000	0.503	0.000	0.000	0.
→ 500	0.750			0.000	1,594	797
2	0.563	0.000	0.503	0.000	0.000	0.
→ 500	0.749			0.000	1,594	797
mean	0.561	0.020	0.504	0.333	0.003	0.
→ 501	0.750			0.000	-	-
std	0.002	0.035	0.003	0.577	0.006	0.
→ 001	0.001			0.000	-	-
coef of var	0.004	1.732	0.006	1.732	1.732	0.
→ 003	0.002			inf	-	-

```

[18]: # 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: 0.7525083612040134

```

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.

**Note:** To provide data to EvalML, it is recommended that you create a `DataTable` object using the [Woodwork project](#).

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 as follows. The `infer_feature_types` utility method takes `pandas` or `numpy` input and converts it to a Woodwork data structure. It takes in a `feature_types` parameter which can be used to specify what types specific columns should be. In the example below, we specify that the `provider`, which would have otherwise been inferred as a column with natural language, is a categorical column.

```
[1]: import evalml
from evalml.utils import infer_feature_types
X, y = evalml.demos.load_fraud(n_rows=1000, return_pandas=True)
X = infer_feature_types(X, feature_types={'provider': 'categorical'})
```

```

      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
```

### 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.

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

data_checks = DefaultDataChecks("binary", "log loss binary")
data_checks.validate(X, y)

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

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

```
[3]: automl = evalml.automl.AutoMLSearch(X_train=X, y_train=y, problem_type='binary')
      automl.search()

Using default limit of max_batches=1.

Generating pipelines to search over...

*****
* 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, extra_trees, catboost, random_forest, lightgbm,
↳ decision_tree, xgboost

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: 4.870

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

Decision Tree Classifier w/ Imputer + DateTime Featurization Component + One Hot
↳ Encoder:
  Starting cross validation
  Finished cross validation - mean Log Loss Binary: 1.321
  High coefficient of variation (cv >= 0.2) within cross validation scores.
  Decision Tree Classifier w/ Imputer + DateTime Featurization Component + One
↳ Hot Encoder may not perform as estimated on unseen data.
Extra Trees Classifier w/ Imputer + DateTime Featurization Component + One Hot
↳ Encoder:
  Starting cross validation
  Finished cross validation - mean Log Loss Binary: 0.367
CatBoost Classifier w/ Imputer + DateTime Featurization Component:
  Starting cross validation
  Finished cross validation - mean Log Loss Binary: 0.558
Random Forest Classifier w/ Imputer + DateTime Featurization Component + One Hot
↳ Encoder:
  Starting cross validation
  Finished cross validation - mean Log Loss Binary: 0.266
LightGBM Classifier w/ Imputer + DateTime Featurization Component + One Hot Encoder:
  Starting cross validation
  Finished cross validation - mean Log Loss Binary: 0.313
  High coefficient of variation (cv >= 0.2) within cross validation scores.
  LightGBM Classifier w/ Imputer + DateTime Featurization Component + One Hot
↳ Encoder may not perform as estimated on unseen data.
XGBoost Classifier w/ Imputer + DateTime Featurization Component + One Hot Encoder:
```

(continues on next page)

(continued from previous page)

```

Starting cross validation
Finished cross validation - mean Log Loss Binary: 0.250
High coefficient of variation (cv >= 0.2) within cross validation scores.
XGBoost Classifier w/ Imputer + DateTime Featurization Component + One Hot
↳Encoder may not perform as estimated on unseen data.
Elastic Net Classifier w/ Imputer + DateTime Featurization Component + One Hot
↳Encoder + Standard Scaler:
Starting cross validation
Finished cross validation - mean Log Loss Binary: 0.423
Logistic Regression Classifier w/ Imputer + DateTime Featurization Component + One
↳Hot Encoder + Standard Scaler:
Starting cross validation
Finished cross validation - mean Log Loss Binary: 0.387

Search finished after 00:17
Best pipeline: XGBoost Classifier w/ Imputer + DateTime Featurization Component + One
↳Hot Encoder
Best pipeline Log Loss Binary: 0.249678

```

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](#).

## 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.

```

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

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

[4]: <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.

```
[5]: 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_pipelines` to a list of *custom pipeline* classes. Note: this will prevent AutoML from generating other pipelines to search over.

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

class CustomMulticlassClassificationPipeline(MulticlassClassificationPipeline):
    component_graph = ['Simple Imputer', 'Random Forest Classifier']

automl_custom = evalml.automl.AutoMLSearch(X_train=X, y_train=y, problem_type=
    ↪ 'multiclass', allowed_pipelines=[CustomMulticlassClassificationPipeline])

Using default limit of max_batches=1.
```

## 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.

```
[7]: ## 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.

```
[8]: ## 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)

```
[9]: # 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.

```
[10]: automl.rankings
```

```
[10]:
```

	id	pipeline_name	mean_cv_score	\
0	6	XGBoost Classifier w/ Imputer + DateTime Featu...	0.249678	
1	4	Random Forest Classifier w/ Imputer + DateTime...	0.265755	
2	5	LightGBM Classifier w/ Imputer + DateTime Feat...	0.313435	
3	2	Extra Trees Classifier w/ Imputer + DateTime F...	0.367375	
4	8	Logistic Regression Classifier w/ Imputer + Da...	0.387335	
5	7	Elastic Net Classifier w/ Imputer + DateTime F...	0.423396	
6	3	CatBoost Classifier w/ Imputer + DateTime Feat...	0.557628	
7	1	Decision Tree Classifier w/ Imputer + DateTime...	1.321054	
8	0	Mode Baseline Binary Classification Pipeline	4.869977	

	standard_deviation_cv_score	validation_score	\
0	0.054240	0.223708	
1	0.009307	0.256599	
2	0.075785	0.289615	
3	0.004340	0.369769	
4	0.033444	0.382680	
5	0.006751	0.419519	
6	0.006400	0.554817	
7	1.161946	2.654672	
8	0.008427	4.860247	

	percent_better_than_baseline	high_variance_cv	\
0	94.873116	True	
1	94.543000	False	
2	93.563937	True	
3	92.456329	False	
4	92.046474	False	
5	91.305990	False	
6	88.549675	False	
7	72.873502	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...

(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'}}

```

## 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`.

```

[11]: automl.describe_pipeline(1)

*****
* Decision Tree Classifier w/ Imputer + DateTime Featurization Component + One Hot_
↪Encoder *
*****

Problem Type: binary
Model Family: Decision Tree

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
3. One Hot Encoder
    * top_n : 10
    * features_to_encode : None
    * categories : None
    * drop : if_binary
    * handle_unknown : ignore
    * handle_missing : error
4. Decision Tree Classifier
    * criterion : gini
    * max_features : auto
    * max_depth : 6
    * min_samples_split : 2
    * min_weight_fraction_leaf : 0.0

Training
=====
Training for binary problems.
Total training time (including CV): 1.5 seconds

Cross Validation
-----

```

	Log Loss Binary	MCC Binary	AUC	Precision	F1	Balanced Accuracy
↪Binary	Accuracy Binary	Sensitivity	at Low Alert Rates	# Training	# Validation	
0	2.655	-0.029	0.496	0.107	0.080	0.
↪488	0.793			0.000	470	334

(continues on next page)

(continued from previous page)

1	0.782	0.567	0.771	0.675	0.621		0.
↪ 765	0.901			0.118	470	333	
2	0.527	-0.022	0.483	0.000	0.000		0.
↪ 498	0.856			0.000	470	333	
mean	1.321	0.172	0.583	0.261	0.234		0.
↪ 584	0.850			0.039	-	-	
std	1.162	0.342	0.163	0.363	0.338		0.
↪ 157	0.054			0.068	-	-	
coef of var	0.880	1.992	0.279	1.391	1.446		0.
↪ 268	0.063			1.732	-	-	

### 4.1.5 Get Pipeline

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

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

Decision Tree Classifier w/ Imputer + DateTime Featurization Component + One Hot_
↪Encoder
{'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}, 'One Hot Encoder': {'top_n': 10, 'features_
↪to_encode': None, 'categories': None, 'drop': 'if_binary', 'handle_unknown': 'ignore
↪', 'handle_missing': 'error'}, 'Decision Tree Classifier': {'criterion': 'gini',
↪'max_features': 'auto', 'max_depth': 6, 'min_samples_split': 2, 'min_weight_
↪fraction_leaf': 0.0}}
```

### 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.

```
[13]: best_pipeline = automl.best_pipeline
print(best_pipeline.name)
print(best_pipeline.parameters)
best_pipeline.predict(X)

XGBoost Classifier w/ Imputer + DateTime Featurization Component + One Hot Encoder
{'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}, 'One Hot Encoder': {'top_n': 10, 'features_
↪to_encode': None, 'categories': None, 'drop': 'if_binary', 'handle_unknown': 'ignore
↪', 'handle_missing': 'error'}, 'XGBoost Classifier': {'eta': 0.1, 'max_depth': 6,
↪'min_child_weight': 1, 'n_estimators': 100}}
```

```
[13]: <DataColumn: fraud (Physical Type = boolean) (Logical Type = Boolean) (Semantic Tags_
↪= set())>
```

## 4.1.6 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.

```
[14]: # saving the best pipeline using .save()
best_pipeline.save("file_path_here")
```

## 4.1.7 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 `pipeline_parameters` argument with your pipeline parameters. These parameters will also 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.

```
[15]: 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
pipeline_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
pipeline_hyperparameters = {'Imputer': {
    'numeric_impute_strategy': Categorical(['median', 'most_frequent'])
}}

# using this pipeline parameter means that our Imputer components in the pipelines_
# ↳ will only search through 'median' and 'most_frequent' strategies for 'numeric_
# ↳ impute_strategy'
automl = AutoMLSearch(X_train=X, y_train=y, problem_type='binary', pipeline_
    ↳ parameters=pipeline_hyperparameters)
automl.search()
automl.best_pipeline.hyperparameters
```

	Number of Features
Boolean	1
Categorical	6
Numeric	5

Number of training examples: 1000

(continues on next page)

(continued from previous page)

```

Targets
False      85.90%
True       14.10%
Name: fraud, dtype: object
Using default limit of max_batches=1.

Generating pipelines to search over...

*****
* 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, extra_trees, catboost, random_forest, lightgbm,
↳decision_tree, xgboost

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: 4.870

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

Decision Tree Classifier w/ Imputer + DateTime Featurization Component + One Hot_
↳Encoder:
  Starting cross validation
  Finished cross validation - mean Log Loss Binary: 1.321
  High coefficient of variation (cv >= 0.2) within cross validation scores.
  Decision Tree Classifier w/ Imputer + DateTime Featurization Component + One_
↳Hot Encoder may not perform as estimated on unseen data.
Extra Trees Classifier w/ Imputer + DateTime Featurization Component + One Hot_
↳Encoder:
  Starting cross validation
  Finished cross validation - mean Log Loss Binary: 0.367
CatBoost Classifier w/ Imputer + DateTime Featurization Component:
  Starting cross validation
  Finished cross validation - mean Log Loss Binary: 0.558
Random Forest Classifier w/ Imputer + DateTime Featurization Component + One Hot_
↳Encoder:
  Starting cross validation
  Finished cross validation - mean Log Loss Binary: 0.266
LightGBM Classifier w/ Imputer + DateTime Featurization Component + One Hot Encoder:
  Starting cross validation
  Finished cross validation - mean Log Loss Binary: 0.313
  High coefficient of variation (cv >= 0.2) within cross validation scores.
  LightGBM Classifier w/ Imputer + DateTime Featurization Component + One Hot_
↳Encoder may not perform as estimated on unseen data.

```

(continues on next page)

(continued from previous page)

```

XGBoost Classifier w/ Imputer + DateTime Featurization Component + One Hot Encoder:
  Starting cross validation
  Finished cross validation - mean Log Loss Binary: 0.250
  High coefficient of variation (cv >= 0.2) within cross validation scores.
  XGBoost Classifier w/ Imputer + DateTime Featurization Component + One Hot
↳Encoder may not perform as estimated on unseen data.
Elastic Net Classifier w/ Imputer + DateTime Featurization Component + One Hot
↳Encoder + Standard Scaler:
  Starting cross validation
  Finished cross validation - mean Log Loss Binary: 0.423
Logistic Regression Classifier w/ Imputer + DateTime Featurization Component + One
↳Hot Encoder + Standard Scaler:
  Starting cross validation
  Finished cross validation - mean Log Loss Binary: 0.387

Search finished after 00:16
Best pipeline: XGBoost Classifier w/ Imputer + DateTime Featurization Component + One
↳Hot Encoder
Best pipeline Log Loss Binary: 0.249678

```

```

[15]: {'Imputer': {'categorical_impute_strategy': ['most_frequent'],
  'numeric_impute_strategy': Categorical(categories=('median', 'most_frequent'),
↳prior=None)},
  'DateTime Featurization Component': {},
  'One Hot Encoder': {},
  'XGBoost Classifier': {'eta': Real(low=1e-06, high=1, prior='uniform', transform=
↳'identity'),
  'max_depth': Integer(low=1, high=10, prior='uniform', transform='identity'),
  'min_child_weight': Real(low=1, high=10, prior='uniform', transform='identity'),
  'n_estimators': Integer(low=1, high=1000, prior='uniform', transform='identity')}}

```

## 4.1.8 Access raw results

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

```

[16]: automl.results
[16]: {'pipeline_results': {0: {'id': 0,
  'pipeline_name': 'Mode Baseline Binary Classification Pipeline',
  'pipeline_class': evalml.pipelines.classification.baseline_binary.
↳ModeBaselineBinaryPipeline,
  'pipeline_summary': 'Baseline Classifier',
  'parameters': {'Baseline Classifier': {'strategy': 'mode'}},
  'mean_cv_score': 4.869977201906586,
  'standard_deviation_cv_score': 0.00842662132408031,
  'high_variance_cv': False,
  'training_time': 0.14518237113952637,
  'cv_data': [{'all_objective_scores': OrderedDict([('Log Loss Binary',
    4.860246977726953),
    ('MCC Binary', 0.0),
    ('AUC', 0.5),
    ('Precision', 0.0),
    ('F1', 0.0),
    ('Balanced Accuracy Binary', 0.5),
    ('Accuracy Binary', 0.8592814371257484),

```

(continues on next page)



(continued from previous page)

```

        ('Sensitivity at Low Alert Rates', 0.0),
        ('# Training', 470),
        ('# Validation', 334)]),
    'mean_cv_score': 4.860246977726953,
    'binary_classification_threshold': None},
{'all_objective_scores': OrderedDict([('Log Loss Binary',
        4.874842313996403),
        ('MCC Binary', 0.0),
        ('AUC', 0.5),
        ('Precision', 0.0),
        ('F1', 0.0),
        ('Balanced Accuracy Binary', 0.5),
        ('Accuracy Binary', 0.8588588588588588),
        ('Sensitivity at Low Alert Rates', 0.0),
        ('# Training', 470),
        ('# Validation', 333)]),
    'mean_cv_score': 4.874842313996403,
    'binary_classification_threshold': None},
{'all_objective_scores': OrderedDict([('Log Loss Binary',
        4.874842313996403),
        ('MCC Binary', 0.0),
        ('AUC', 0.5),
        ('Precision', 0.0),
        ('F1', 0.0),
        ('Balanced Accuracy Binary', 0.5),
        ('Accuracy Binary', 0.8588588588588588),
        ('Sensitivity at Low Alert Rates', 0.0),
        ('# Training', 470),
        ('# Validation', 333)]),
    'mean_cv_score': 4.874842313996403,
    'binary_classification_threshold': None}},
'percent_better_than_baseline_all_objectives': {'Log Loss Binary': 0,
'MCC Binary': 0,
'AUC': 0,
'Precision': 0,
'F1': 0,
'Balanced Accuracy Binary': 0,
'Accuracy Binary': 0,
'Sensitivity at Low Alert Rates': 0},
'percent_better_than_baseline': 0,
'validation_score': 4.860246977726953},
1: {'id': 1,
    'pipeline_name': 'Decision Tree Classifier w/ Imputer + DateTime Featurization_
↳Component + One Hot Encoder',
    'pipeline_class': evalml.pipelines.utils.make_pipeline.<locals>.GeneratedPipeline,
    'pipeline_summary': 'Decision Tree Classifier w/ Imputer + DateTime Featurization_
↳Component + One Hot Encoder',
    'parameters': {'Imputer': {'categorical_impute_strategy': 'most_frequent',
        'numeric_impute_strategy': 'most_frequent',
        'categorical_fill_value': None,
        'numeric_fill_value': None},
        'DateTime Featurization Component': {'features_to_extract': ['year',
            'month',
            'day_of_week',
            'hour'],
        'encode_as_categories': False},
        'One Hot Encoder': {'top_n': 10,

```

(continues on next page)

(continued from previous page)

```

'features_to_encode': None,
'categories': None,
'drop': 'if_binary',
'handle_unknown': 'ignore',
'handle_missing': 'error'},
'Decision Tree Classifier': {'criterion': 'gini',
'max_features': 'auto',
'max_depth': 6,
'min_samples_split': 2,
'min_weight_fraction_leaf': 0.0}},
'mean_cv_score': 1.3210542665452085,
'standard_deviation_cv_score': 1.1619460783767976,
'high_variance_cv': True,
'training_time': 1.5409376621246338,
'cv_data': [{'all_objective_scores': OrderedDict([('Log Loss Binary',
2.6546720440702334),
('MCC Binary', -0.02920788041505265),
('AUC', 0.4955519312032027),
('Precision', 0.10714285714285714),
('F1', 0.08),
('Balanced Accuracy Binary', 0.4883608866483802),
('Accuracy Binary', 0.7934131736526946),
('Sensitivity at Low Alert Rates', 0.0),
('# Training', 470),
('# Validation', 334)]),
'mean_cv_score': 2.6546720440702334,
'binary_classification_threshold': None},
{'all_objective_scores': OrderedDict([('Log Loss Binary',
0.7815890335223087),
('MCC Binary', 0.5665456733923655),
('AUC', 0.7705326588305312),
('Precision', 0.675),
('F1', 0.6206896551724138),
('Balanced Accuracy Binary', 0.7645067698259188),
('Accuracy Binary', 0.9009009009009009),
('Sensitivity at Low Alert Rates', 0.11764705882352941),
('# Training', 470),
('# Validation', 333)]),
'mean_cv_score': 0.7815890335223087,
'binary_classification_threshold': None},
{'all_objective_scores': OrderedDict([('Log Loss Binary',
0.5269017220430833),
('MCC Binary', -0.02224830197498392),
('AUC', 0.482629073054605),
('Precision', 0.0),
('F1', 0.0),
('Balanced Accuracy Binary', 0.4982517482517482),
('Accuracy Binary', 0.8558558558558559),
('Sensitivity at Low Alert Rates', 0.0),
('# Training', 470),
('# Validation', 333)]),
'mean_cv_score': 0.5269017220430833,
'binary_classification_threshold': None}],
'percent_better_than_baseline_all_objectives': {'Log Loss Binary': 72.
→87350203553277,
'MCC Binary': inf,
'AUC': 8.290455436277966,

```

(continues on next page)

(continued from previous page)

```

'Precision': 26.071428571428573,
'F1': 23.35632183908046,
'Balanced Accuracy Binary': 8.370646824201577,
'Accuracy Binary': -0.8943074811338048,
'Sensitivity at Low Alert Rates': 3.9215686274509802},
'percent_better_than_baseline': 72.87350203553277,
'validation_score': 2.6546720440702334},
2: {'id': 2,
'pipeline_name': 'Extra Trees Classifier w/ Imputer + DateTime Featurization_
↳Component + One Hot Encoder',
'pipeline_class': evalml.pipelines.utils.make_pipeline.<locals>.GeneratedPipeline,
'pipeline_summary': 'Extra Trees Classifier w/ Imputer + DateTime Featurization_
↳Component + One Hot Encoder',
'parameters': {'Imputer': {'categorical_impute_strategy': 'most_frequent',
'numeric_impute_strategy': 'most_frequent',
'categorical_fill_value': None,
'numeric_fill_value': None},
'Datetime Featurization Component': {'features_to_extract': ['year',
'month',
'day_of_week',
'hour'],
'encode_as_categories': False},
'One Hot Encoder': {'top_n': 10,
'features_to_encode': None,
'categories': None,
'drop': 'if_binary',
'handle_unknown': 'ignore',
'handle_missing': 'error'}},
'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': 0.3673750416327947,
'standard_deviation_cv_score': 0.004340027380001022,
'high_variance_cv': False,
'training_time': 2.0252888202667236,
'cv_data': [{'all_objective_scores': OrderedDict([('Log Loss Binary',
0.36976942662449674),
('MCC Binary', 0.0),
('AUC', 0.7941285491882275),
('Precision', 0.0),
('F1', 0.0),
('Balanced Accuracy Binary', 0.5),
('Accuracy Binary', 0.8592814371257484),
('Sensitivity at Low Alert Rates', 0.0),
('# Training', 470),
('# Validation', 334)]),
'mean_cv_score': 0.36976942662449674,
'binary_classification_threshold': None},
{'all_objective_scores': OrderedDict([('Log Loss Binary',
0.36236523480517696),
('MCC Binary', 0.1353832843584128),
('AUC', 0.8080642761493826),
('Precision', 1.0),
('F1', 0.04166666666666667),

```

(continues on next page)

(continued from previous page)

```

        ('Balanced Accuracy Binary', 0.5106382978723404),
        ('Accuracy Binary', 0.8618618618618619),
        ('Sensitivity at Low Alert Rates', 0.0),
        ('# Training', 470),
        ('# Validation', 333)]),
    'mean_cv_score': 0.36236523480517696,
    'binary_classification_threshold': None},
{'all_objective_scores': OrderedDict([('Log Loss Binary',
    0.36999046346871023),
    ('MCC Binary', 0.0),
    ('AUC', 0.8213063532212468),
    ('Precision', 0.0),
    ('F1', 0.0),
    ('Balanced Accuracy Binary', 0.5),
    ('Accuracy Binary', 0.8588588588588588),
    ('Sensitivity at Low Alert Rates', 0.0),
    ('# Training', 470),
    ('# Validation', 333)]),
    'mean_cv_score': 0.36999046346871023,
    'binary_classification_threshold': None}},
'percent_better_than_baseline_all_objectives': {'Log Loss Binary': 92.
↪45632933376015,
    'MCC Binary': inf,
    'AUC': 30.783305951961903,
    'Precision': 33.33333333333333,
    'F1': 1.3888888888888889,
    'Balanced Accuracy Binary': 0.35460992907800915,
    'Accuracy Binary': 0.10010010010010895,
    'Sensitivity at Low Alert Rates': 0},
'percent_better_than_baseline': 92.45632933376015,
'validation_score': 0.36976942662449674),
3: {'id': 3,
    'pipeline_name': 'CatBoost Classifier w/ Imputer + DateTime Featurization Component
↪',
    'pipeline_class': evalml.pipelines.utils.make_pipeline.<locals>.GeneratedPipeline,
    'pipeline_summary': 'CatBoost Classifier w/ Imputer + DateTime Featurization_
↪Component',
    'parameters': {'Imputer': {'categorical_impute_strategy': 'most_frequent',
    'numeric_impute_strategy': 'most_frequent',
    'categorical_fill_value': None,
    'numeric_fill_value': None},
    'DateTime Featurization Component': {'features_to_extract': ['year',
    'month',
    'day_of_week',
    'hour'],
    'encode_as_categories': False},
    'CatBoost Classifier': {'n_estimators': 10,
    'eta': 0.03,
    'max_depth': 6,
    'bootstrap_type': None,
    'silent': True,
    'allow_writing_files': False}},
    'mean_cv_score': 0.5576282401325289,
    'standard_deviation_cv_score': 0.0064004352378003015,
    'high_variance_cv': False,
    'training_time': 0.9364278316497803,
    'cv_data': [{'all_objective_scores': OrderedDict([('Log Loss Binary',

```

(continues on next page)

(continued from previous page)

```

        0.5548165447783588),
        ('MCC Binary', 0.7474984936795003),
        ('AUC', 0.815145674253095),
        ('Precision', 1.0),
        ('F1', 0.7466666666666666),
        ('Balanced Accuracy Binary', 0.7978723404255319),
        ('Accuracy Binary', 0.9431137724550899),
        ('Sensitivity at Low Alert Rates', 0.0),
        ('# Training', 470),
        ('# Validation', 334)]),
    'mean_cv_score': 0.5548165447783588,
    'binary_classification_threshold': None},
{'all_objective_scores': OrderedDict([('Log Loss Binary',
        0.564953238775739),
        ('MCC Binary', 0.7474173647473783),
        ('AUC', 0.8136438030055051),
        ('Precision', 1.0),
        ('F1', 0.7466666666666666),
        ('Balanced Accuracy Binary', 0.7978723404255319),
        ('Accuracy Binary', 0.9429429429429429),
        ('Sensitivity at Low Alert Rates', 0.11764705882352941),
        ('# Training', 470),
        ('# Validation', 333)]),
    'mean_cv_score': 0.564953238775739,
    'binary_classification_threshold': None},
{'all_objective_scores': OrderedDict([('Log Loss Binary',
        0.5531149368434888),
        ('MCC Binary', 0.790335451188355),
        ('AUC', 0.8285597381342062),
        ('Precision', 1.0),
        ('F1', 0.7948717948717948),
        ('Balanced Accuracy Binary', 0.8297872340425532),
        ('Accuracy Binary', 0.9519519519519519),
        ('Sensitivity at Low Alert Rates', 0.14285714285714285),
        ('# Training', 470),
        ('# Validation', 333)]),
    'mean_cv_score': 0.5531149368434888,
    'binary_classification_threshold': None}},
'percent_better_than_baseline_all_objectives': {'Log Loss Binary': 88.
→54967452590499,
    'MCC Binary': inf,
    'AUC': 31.911640513093552,
    'Precision': 100.0,
    'F1': 76.27350427350427,
    'Balanced Accuracy Binary': 30.85106382978724,
    'Accuracy Binary': 8.700317083550624,
    'Sensitivity at Low Alert Rates': 8.68347338935574},
'percent_better_than_baseline': 88.54967452590499,
'validation_score': 0.5548165447783588},
4: {'id': 4,
    'pipeline_name': 'Random Forest Classifier w/ Imputer + DateTime Featurization_
→Component + One Hot Encoder',
    'pipeline_class': evalml.pipelines.utils.make_pipeline.<locals>.GeneratedPipeline,
    'pipeline_summary': 'Random Forest Classifier w/ Imputer + DateTime Featurization_
→Component + One Hot Encoder',
    'parameters': {'Imputer': {'categorical_impute_strategy': 'most_frequent',
    'numeric_impute_strategy': 'most_frequent',

```

(continues on next page)

(continued from previous page)

```

'categorical_fill_value': None,
'numeric_fill_value': None},
'Datetime Featurization Component': {'features_to_extract': ['year',
'month',
'day_of_week',
'hour'],
'encode_as_categories': False},
'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}},
'mean_cv_score': 0.2657546770718203,
'standard_deviation_cv_score': 0.009306871235747024,
'high_variance_cv': False,
'training_time': 2.262765884399414,
'cv_data': [{'all_objective_scores': OrderedDict([('Log Loss Binary',
0.2565988400333464),
('MCC Binary', 0.7619756598302211),
('AUC', 0.8469864333901698),
('Precision', 1.0),
('F1', 0.763157894736842),
('Balanced Accuracy Binary', 0.8085106382978724),
('Accuracy Binary', 0.9461077844311377),
('Sensitivity at Low Alert Rates', 0.0),
('# Training', 470),
('# Validation', 334)]),
'mean_cv_score': 0.2565988400333464,
'binary_classification_threshold': None},
{'all_objective_scores': OrderedDict([('Log Loss Binary',
0.27520556576257044),
('MCC Binary', 0.7159491186351445),
('AUC', 0.8250260377919952),
('Precision', 0.9333333333333333),
('F1', 0.7272727272727273),
('Balanced Accuracy Binary', 0.7943758369290284),
('Accuracy Binary', 0.9369369369369369),
('Sensitivity at Low Alert Rates', 0.11764705882352941),
('# Training', 470),
('# Validation', 333)]),
'mean_cv_score': 0.27520556576257044,
'binary_classification_threshold': None},
{'all_objective_scores': OrderedDict([('Log Loss Binary',
0.26545962541954415),
('MCC Binary', 0.8043143718855572),
('AUC', 0.8359619104299956),
('Precision', 1.0),
('F1', 0.810126582278481),
('Balanced Accuracy Binary', 0.8404255319148937),
('Accuracy Binary', 0.954954954954955),
('Sensitivity at Low Alert Rates', 0.14285714285714285),
('# Training', 470),
('# Validation', 333)]),

```

(continues on next page)

(continued from previous page)

```

    'mean_cv_score': 0.26545962541954415,
    'binary_classification_threshold': None}},
    'percent_better_than_baseline_all_objectives': {'Log Loss Binary': 94.
↪54299956542347,
    'MCC Binary': inf,
    'AUC': 33.59914605373868,
    'Precision': 97.77777777777779,
    'F1': 76.68524014293502,
    'Balanced Accuracy Binary': 31.44373357139315,
    'Accuracy Binary': 8.700017382652126,
    'Sensitivity at Low Alert Rates': 8.68347338935574},
    'percent_better_than_baseline': 94.54299956542347,
    'validation_score': 0.2565988400333464},
5: {'id': 5,
    'pipeline_name': 'LightGBM Classifier w/ Imputer + DateTime Featurization_
↪Component + One Hot Encoder',
    'pipeline_class': evalml.pipelines.utils.make_pipeline.<locals>.GeneratedPipeline,
    'pipeline_summary': 'LightGBM Classifier w/ Imputer + DateTime Featurization_
↪Component + One Hot Encoder',
    'parameters': {'Imputer': {'categorical_impute_strategy': 'most_frequent',
    'numeric_impute_strategy': 'most_frequent',
    'categorical_fill_value': None,
    'numeric_fill_value': None},
    'DateTime Featurization Component': {'features_to_extract': ['year',
    'month',
    'day_of_week',
    'hour'],
    'encode_as_categories': False},
    'One Hot Encoder': {'top_n': 10,
    'features_to_encode': None,
    'categories': None,
    'drop': 'if_binary',
    'handle_unknown': 'ignore',
    'handle_missing': 'error'},
    'LightGBM Classifier': {'boosting_type': 'gbdt',
    'learning_rate': 0.1,
    'n_estimators': 100,
    'max_depth': 0,
    'num_leaves': 31,
    'min_child_samples': 20,
    'n_jobs': -1,
    'bagging_freq': 0,
    'bagging_fraction': 0.9}},
    'mean_cv_score': 0.31343481346709856,
    'standard_deviation_cv_score': 0.07578466383232263,
    'high_variance_cv': True,
    'training_time': 1.8752498626708984,
    'cv_data': [{'all_objective_scores': OrderedDict([('Log Loss Binary',
    0.28961516455260206),
    ('MCC Binary', 0.7310633695590257),
    ('AUC', 0.8404625991548669),
    ('Precision', 0.9354838709677419),
    ('F1', 0.7435897435897436),
    ('Balanced Accuracy Binary', 0.805026317740381),
    ('Accuracy Binary', 0.9401197604790419),
    ('Sensitivity at Low Alert Rates', 0.0),
    ('# Training', 470),

```

(continues on next page)

(continued from previous page)

```

        ('# Validation', 334)]),
    'mean_cv_score': 0.28961516455260206,
    'binary_classification_threshold': None},
    {'all_objective_scores': OrderedDict([('Log Loss Binary',
        0.3982677722259575),
        ('MCC Binary', 0.648530742413196),
        ('AUC', 0.781431334622824),
        ('Precision', 0.8),
        ('F1', 0.6829268292682927),
        ('Balanced Accuracy Binary', 0.7856345781877696),
        ('Accuracy Binary', 0.9219219219219219),
        ('Sensitivity at Low Alert Rates', 0.11764705882352941),
        ('# Training', 470),
        ('# Validation', 333)]),
    'mean_cv_score': 0.3982677722259575,
    'binary_classification_threshold': None},
    {'all_objective_scores': OrderedDict([('Log Loss Binary',
        0.2524215036227362),
        ('MCC Binary', 0.7349200629211959),
        ('AUC', 0.8600654664484452),
        ('Precision', 0.8648648648648649),
        ('F1', 0.7619047619047619),
        ('Balanced Accuracy Binary', 0.8316842731736349),
        ('Accuracy Binary', 0.93993993993994),
        ('Sensitivity at Low Alert Rates', 0.14285714285714285),
        ('# Training', 470),
        ('# Validation', 333)]),
    'mean_cv_score': 0.2524215036227362,
    'binary_classification_threshold': None}},
    'percent_better_than_baseline_all_objectives': {'Log Loss Binary': 93.
→56393673990117,
    'MCC Binary': inf,
    'AUC': 32.731980007537864,
    'Precision': 86.67829119442023,
    'F1': 72.94737782542661,
    'Balanced Accuracy Binary': 30.74483897005952,
    'Accuracy Binary': 7.4994155832479255,
    'Sensitivity at Low Alert Rates': 8.68347338935574},
    'percent_better_than_baseline': 93.56393673990117,
    'validation_score': 0.28961516455260206},
6: {'id': 6,
    'pipeline_name': 'XGBoost Classifier w/ Imputer + DateTime Featurization Component_
→+ One Hot Encoder',
    'pipeline_class': evalml.pipelines.utils.make_pipeline.<locals>.GeneratedPipeline,
    'pipeline_summary': 'XGBoost Classifier w/ Imputer + DateTime Featurization_
→Component + One Hot Encoder',
    'parameters': {'Imputer': {'categorical_impute_strategy': 'most_frequent',
    'numeric_impute_strategy': 'most_frequent',
    'categorical_fill_value': None,
    'numeric_fill_value': None},
    'DateTime Featurization Component': {'features_to_extract': ['year',
    'month',
    'day_of_week',
    'hour'],
    'encode_as_categories': False},
    'One Hot Encoder': {'top_n': 10,
    'features_to_encode': None,

```

(continues on next page)



(continued from previous page)

```

'categories': None,
'drop': 'if_binary',
'handle_unknown': 'ignore',
'handle_missing': 'error'},
'XGBoost Classifier': {'eta': 0.1,
'max_depth': 6,
'min_child_weight': 1,
'n_estimators': 100}},
'mean_cv_score': 0.2496780620661376,
'standard_deviation_cv_score': 0.05423968083588209,
'high_variance_cv': True,
'training_time': 2.0243005752563477,
'cv_data': [{'all_objective_scores': OrderedDict([('Log Loss Binary',
0.22370773034682934),
('MCC Binary', 0.7607358893588453),
('AUC', 0.8446882645118244),
('Precision', 0.967741935483871),
('F1', 0.7692307692307693),
('Balanced Accuracy Binary', 0.8174067758914672),
('Accuracy Binary', 0.9461077844311377),
('Sensitivity at Low Alert Rates', 0.0),
('# Training', 470),
('# Validation', 334)]),
'mean_cv_score': 0.22370773034682934,
'binary_classification_threshold': None},
{'all_objective_scores': OrderedDict([('Log Loss Binary',
0.31202008540244003),
('MCC Binary', 0.6766542919930364),
('AUC', 0.7967564350543073),
('Precision', 0.8285714285714286),
('F1', 0.7073170731707318),
('Balanced Accuracy Binary', 0.7980211278083619),
('Accuracy Binary', 0.9279279279279279),
('Sensitivity at Low Alert Rates', 0.11764705882352941),
('# Training', 470),
('# Validation', 333)]),
'mean_cv_score': 0.31202008540244003,
'binary_classification_threshold': None},
{'all_objective_scores': OrderedDict([('Log Loss Binary',
0.21330637044914344),
('MCC Binary', 0.789278744518655),
('AUC', 0.8579824430888261),
('Precision', 0.9696969696969697),
('F1', 0.7999999999999999),
('Balanced Accuracy Binary', 0.8386772801666418),
('Accuracy Binary', 0.9519519519519519),
('Sensitivity at Low Alert Rates', 0.14285714285714285),
('# Training', 470),
('# Validation', 333)]),
'mean_cv_score': 0.21330637044914344,
'binary_classification_threshold': None}],
'percent_better_than_baseline_all_objectives': {'Log Loss Binary': 94.
↪87311640866842,
'MCC Binary': inf,
'AUC': 33.31423808849859,
'Precision': 92.20034445840898,
'F1': 75.88492808005003,

```

(continues on next page)

(continued from previous page)

```

    'Balanced Accuracy Binary': 31.803506128882354,
    'Accuracy Binary': 8.299616982251711,
    'Sensitivity at Low Alert Rates': 8.68347338935574},
    'percent_better_than_baseline': 94.87311640866842,
    'validation_score': 0.22370773034682934},
7: {'id': 7,
    'pipeline_name': 'Elastic Net Classifier w/ Imputer + DateTime Featurization_
↳Component + One Hot Encoder + Standard Scaler',
    'pipeline_class': evalml.pipelines.utils.make_pipeline.<locals>.GeneratedPipeline,
    'pipeline_summary': 'Elastic Net Classifier w/ Imputer + DateTime Featurization_
↳Component + One Hot Encoder + Standard Scaler',
    'parameters': {'Imputer': {'categorical_impute_strategy': 'most_frequent',
    'numeric_impute_strategy': 'most_frequent',
    'categorical_fill_value': None,
    'numeric_fill_value': None},
    'DateTime Featurization Component': {'features_to_extract': ['year',
    'month',
    'day_of_week',
    'hour'],
    'encode_as_categories': False},
    'One Hot Encoder': {'top_n': 10,
    'features_to_encode': None,
    'categories': None,
    'drop': 'if_binary',
    'handle_unknown': 'ignore',
    'handle_missing': 'error'}},
    'Elastic Net Classifier': {'alpha': 0.5,
    'l1_ratio': 0.5,
    'n_jobs': -1,
    'max_iter': 1000,
    'penalty': 'elasticnet',
    'loss': 'log'}},
    'mean_cv_score': 0.4233962942773955,
    'standard_deviation_cv_score': 0.0067510251914937505,
    'high_variance_cv': False,
    'training_time': 2.220989227294922,
    'cv_data': [{'all_objective_scores': OrderedDict([('Log Loss Binary',
    0.4195186207236776),
    ('MCC Binary', 0.0),
    ('AUC', 0.5),
    ('Precision', 0.0),
    ('F1', 0.0),
    ('Balanced Accuracy Binary', 0.5),
    ('Accuracy Binary', 0.8592814371257484),
    ('Sensitivity at Low Alert Rates', 0.0),
    ('# Training', 470),
    ('# Validation', 334)]),
    'mean_cv_score': 0.4195186207236776,
    'binary_classification_threshold': None},
    {'all_objective_scores': OrderedDict([('Log Loss Binary',
    0.4311916724392599),
    ('MCC Binary', 0.0),
    ('AUC', 0.5),
    ('Precision', 0.0),
    ('F1', 0.0),
    ('Balanced Accuracy Binary', 0.5),
    ('Accuracy Binary', 0.8588588588588588),

```

(continues on next page)

(continued from previous page)

```

        ('Sensitivity at Low Alert Rates', 0.0),
        ('# Training', 470),
        ('# Validation', 333)]),
    'mean_cv_score': 0.4311916724392599,
    'binary_classification_threshold': None},
{'all_objective_scores': OrderedDict([('Log Loss Binary',
    0.4194785896692491),
    ('MCC Binary', 0.0),
    ('AUC', 0.5),
    ('Precision', 0.0),
    ('F1', 0.0),
    ('Balanced Accuracy Binary', 0.5),
    ('Accuracy Binary', 0.8588588588588588),
    ('Sensitivity at Low Alert Rates', 0.0),
    ('# Training', 470),
    ('# Validation', 333)]),
    'mean_cv_score': 0.4194785896692491,
    'binary_classification_threshold': None}},
'percent_better_than_baseline_all_objectives': {'Log Loss Binary': 91.
↪30599021877069,
'MCC Binary': 0,
'AUC': 0,
'Precision': 0,
'F1': 0,
'Balanced Accuracy Binary': 0,
'Accuracy Binary': 0,
'Sensitivity at Low Alert Rates': 0},
'percent_better_than_baseline': 91.30599021877069,
'validation_score': 0.4195186207236776},
8: {'id': 8,
    'pipeline_name': 'Logistic Regression Classifier w/ Imputer + DateTime_
↪Featurization Component + One Hot Encoder + Standard Scaler',
    'pipeline_class': evalml.pipelines.utils.make_pipeline.<locals>.GeneratedPipeline,
    'pipeline_summary': 'Logistic Regression Classifier w/ Imputer + DateTime_
↪Featurization Component + One Hot Encoder + Standard Scaler',
    'parameters': {'Imputer': {'categorical_impute_strategy': 'most_frequent',
    'numeric_impute_strategy': 'most_frequent',
    'categorical_fill_value': None,
    'numeric_fill_value': None},
    'DateTime Featurization Component': {'features_to_extract': ['year',
    'month',
    'day_of_week',
    'hour'],
    'encode_as_categories': False},
    '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'}}},
    'mean_cv_score': 0.387334884247545,
    'standard_deviation_cv_score': 0.03344445835807682,

```

(continues on next page)

(continued from previous page)

```

'high_variance_cv': False,
'training_time': 2.355400800704956,
'cv_data': [{'all_objective_scores': OrderedDict([('Log Loss Binary',
0.38267981789139965),
('MCC Binary', 0.42377953609295965),
('AUC', 0.7401586477870857),
('Precision', 0.5714285714285714),
('F1', 0.4878048780487805),
('Balanced Accuracy Binary', 0.6866335532656238),
('Accuracy Binary', 0.874251497005988),
('Sensitivity at Low Alert Rates', 0.16666666666666666),
('# Training', 470),
('# Validation', 334)]),
'mean_cv_score': 0.38267981789139965,
'binary_classification_threshold': None},
{'all_objective_scores': OrderedDict([('Log Loss Binary',
0.4228630132452719),
('MCC Binary', 0.3068030454821329),
('AUC', 0.6881416455884541),
('Precision', 0.46875),
('F1', 0.379746835443038),
('Balanced Accuracy Binary', 0.6298541883648267),
('Accuracy Binary', 0.8528528528528528),
('Sensitivity at Low Alert Rates', 0.058823529411764705),
('# Training', 470),
('# Validation', 333)]),
'mean_cv_score': 0.4228630132452719,
'binary_classification_threshold': None},
{'all_objective_scores': OrderedDict([('Log Loss Binary',
0.3564618216059633),
('MCC Binary', 0.2696810734978073),
('AUC', 0.760377919952388),
('Precision', 0.42424242424242425),
('F1', 0.35),
('Balanced Accuracy Binary', 0.6157193869959827),
('Accuracy Binary', 0.8438438438438438),
('Sensitivity at Low Alert Rates', 0.14285714285714285),
('# Training', 470),
('# Validation', 333)]),
'mean_cv_score': 0.3564618216059633,
'binary_classification_threshold': None}},
'percent_better_than_baseline_all_objectives': {'Log Loss Binary': 92.
↪04647438398882,
'MCC Binary': inf,
'AUC': 22.955940444264268,
'Precision': 48.81403318903319,
'F1': 40.58505711639395,
'Balanced Accuracy Binary': 14.406904287547773,
'Accuracy Binary': -0.2016987046927099,
'Sensitivity at Low Alert Rates': 12.278244631185807},
'percent_better_than_baseline': 92.04647438398882,
'validation_score': 0.38267981789139965}},
'search_order': [0, 1, 2, 3, 4, 5, 6, 7, 8]]

```

## 4.1.9 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.

```
[17]: X, y = evalml.demos.load_breast_cancer()
      automl_with_ensembling = AutoMLSearch(X_train=X, y_train=y,
                                           problem_type="binary",
                                           allowed_model_families=[ModelFamily.RANDOM_
      ↪FOREST, ModelFamily.LINEAR_MODEL],
                                           max_batches=5,
                                           ensembling=True)

      automl_with_ensembling.search()
```

```
Generating pipelines to search over...
Ensembling will run every 4 batches.
```

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

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

```
Using SequentialEngine to train and score pipelines.
Searching up to 5 batches for a total of 20 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 Log Loss Binary: 12.904
```

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

```
Elastic Net Classifier w/ Imputer + Standard Scaler:
  Starting cross validation
  Finished cross validation - mean Log Loss Binary: 0.508
Logistic Regression Classifier w/ Imputer + Standard Scaler:
  Starting cross validation
  Finished cross validation - mean Log Loss Binary: 0.074
```

(continues on next page)

(continued from previous page)

```

    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.
Random Forest Classifier w/ Imputer:
    Starting cross validation
    Finished cross validation - mean Log Loss Binary: 0.116
    High coefficient of variation (cv >= 0.2) within cross validation scores.
    Random Forest Classifier w/ Imputer 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: 0.106
    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: 0.107
    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: 0.098
    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: 0.075
    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 3 *
*****

Random Forest Classifier w/ Imputer:
    Starting cross validation
    Finished cross validation - mean Log Loss Binary: 0.121
    High coefficient of variation (cv >= 0.2) within cross validation scores.
    Random Forest Classifier w/ Imputer may not perform as estimated on unseen
↳data.
Random Forest Classifier w/ Imputer:
    Starting cross validation

```

(continues on next page)

(continued from previous page)

```

    Finished cross validation - mean Log Loss Binary: 0.165
Random Forest Classifier w/ Imputer:
    Starting cross validation
    Finished cross validation - mean Log Loss Binary: 0.117
    High coefficient of variation (cv >= 0.2) within cross validation scores.
    Random Forest Classifier w/ Imputer may not perform as estimated on unseen_
↪data.
Random Forest Classifier w/ Imputer:
    Starting cross validation
    Finished cross validation - mean Log Loss Binary: 0.165
Random Forest Classifier w/ Imputer:
    Starting cross validation
    Finished cross validation - mean Log Loss Binary: 0.117
    High coefficient of variation (cv >= 0.2) within cross validation scores.
    Random Forest Classifier w/ Imputer may not perform as estimated on unseen_
↪data.

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

Elastic Net Classifier w/ Imputer + Standard Scaler:
    Starting cross validation
    Finished cross validation - mean Log Loss Binary: 0.663
Elastic Net Classifier w/ Imputer + Standard Scaler:
    Starting cross validation
    Finished cross validation - mean Log Loss Binary: 0.662
Elastic Net Classifier w/ Imputer + Standard Scaler:
    Starting cross validation
    Finished cross validation - mean Log Loss Binary: 0.324
Elastic Net Classifier w/ Imputer + Standard Scaler:
    Starting cross validation
    Finished cross validation - mean Log Loss Binary: 0.458
Elastic Net Classifier w/ Imputer + Standard Scaler:
    Starting cross validation
    Finished cross validation - mean Log Loss Binary: 0.217

*****
* Evaluating Batch Number 5 *
*****

Stacked Ensemble Classification Pipeline:
    Starting cross validation
    Finished cross validation - mean Log Loss Binary: 0.111

Search finished after 00:42
Best pipeline: Logistic Regression Classifier w/ Imputer + Standard Scaler
Best pipeline Log Loss Binary: 0.073615

```

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

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

```

*****
* Logistic Regression Classifier w/ Imputer + Standard Scaler *

```

(continues on next page)

(continued from previous page)

```
*****

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

Pipeline Steps
=====
1. Imputer
    * categorical_impute_strategy : most_frequent
    * numeric_impute_strategy : mean
    * categorical_fill_value : None
    * numeric_fill_value : None
2. Standard Scaler
3. Logistic Regression Classifier
    * penalty : l2
    * C : 1.0
    * n_jobs : -1
    * multi_class : auto
    * solver : lbfgs
```

## 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
AUC
Precision
F1
Balanced Accuracy Binary
Accuracy Binary
Sensitivity at Low Alert Rates

```

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

      class RFBinaryClassificationPipeline(BinaryClassificationPipeline):
          component_graph = ['Simple Imputer', 'DateTime Featurization Component', 'One Hot_
          ↪Encoder', 'Random Forest Classifier']

      X, y = load_fraud(n_rows=100)
      objective = F1()
      pipeline = RFBinaryClassificationPipeline({})
      pipeline.fit(X, y)
      print(pipeline.threshold)
      print(pipeline.score(X, y, objectives=[objective]))

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

(continues on next page)

(continued from previous page)

```

        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)

        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)

```

(continues on next page)

(continued from previous page)

```

    # 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

```

(continues on next page)

(continued from previous page)

```
from evalml.pipelines.components import SimpleImputer
```

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

```
   0    1    2
0  1  2.0  3
1  1  NaN  3
```

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

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

	Physical Type	Logical Type	Semantic Tag(s)
Data Column			
0	Int64	Integer	['numeric']
1	float64	Double	['numeric']
2	Int64	Integer	['numeric']

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: Polynomial Detrender
Transformer: DFS Transformer
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 Transformer
Transformer: Drop Columns Transformer
Transformer: Undersampler
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
```

### 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]: <DataColumn: None (Physical Type = Int64) (Logical Type = Integer) (Semantic Tags = {
  ↳ 'numeric'})>
```

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: Stacked Ensemble Regressor
Estimator: 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,
    _convert_woodwork_types_wrapper
)

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.
                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 (list, ww.DataTable, pd.DataFrame): The input training data of shape [n_
            samples, n_features]
            y (list, ww.DataColumn, pd.Series, np.ndarray, optional): The target
            training data of length [n_samples]

        Returns:
            self
        """
        pct_null_threshold = self.parameters["pct_null_threshold"]
```

(continues on next page)

(continued from previous page)

```

X_t = infer_feature_types(X)
X_t = _convert_woodwork_types_wrapper(X_t.to_dataframe())
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 (ww.DataTable, pd.DataFrame): Data to transform
        y (ww.DataColumn, pd.Series, optional): Ignored.

    Returns:
        ww.DataTable: Transformed X
    """
    X_t = infer_feature_types(X)
    return X_t.drop(self._cols_to_drop)

```

## Required fields

For a transformer you must provide a class attribute name indicating a human-readable name.

## 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 Woodwork DataTable. 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:
            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)
        y = _convert_woodwork_types_wrapper(y.to_series())

        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
```

(continues on next page)

(continued from previous page)

```

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 indicating 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

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

Each component and pipeline class 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 Class Definition

Pipeline definitions must inherit from the proper pipeline base class, `RegressionPipeline`, `BinaryClassificationPipeline` or `MulticlassClassificationPipeline`. They must also include a `component_graph` class variable, which can either be 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

class CustomMulticlassClassificationPipeline(MulticlassClassificationPipeline):
    component_graph = ['Imputer', 'Random Forest Classifier']

[2]: class_
      ↪ CustomNonlinearMulticlassClassificationPipeline(MulticlassClassificationPipeline):
    component_graph = {
        'Imputer': ['Imputer'],
        'Encoder': ['One Hot Encoder', 'Imputer'],
        'Random Forest Clf': ['Random Forest Classifier', 'Encoder'],
        'Elastic Net Clf': ['Elastic Net Classifier', 'Encoder'],
        'Final Estimator': ['Logistic Regression Classifier', 'Random Forest Clf',
      ↪ 'Elastic Net Clf']
    }
```

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, random_seed):
        transformer = ThirdPartyTransformer(parameter_1)
        parameters = {"parameter_1": parameter_1}
        super().__init__(parameters=parameters,
                         component_obj=transformer,
                         random_seed=random_seed)

class_
      ↪ CustomComponentMulticlassClassificationPipeline(MulticlassClassificationPipeline):
    component_graph = [NewTransformer, 'Random Forest Classifier']
```

## 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 = CustomMulticlassClassificationPipeline({})
pipeline.fit(X, y)
print(pipeline.predict(X))
print(pipeline.score(X, y, objectives=['log loss multiclass']))

<DataColumn: None (Physical Type = category) (Logical Type = Categorical) (Semantic_
↳Tags = {'category'})>
OrderedDict([('Log Loss Multiclass', 0.04132737017536148)])
```

## 4.4.3 Custom Name

By default, a pipeline class's name property is the result of adding spaces between each Pascal case capitalization in the class name. E.g. `LogisticRegressionPipeline.name` will return 'Logistic Regression Pipeline'. Therefore, we suggest custom pipelines use Pascal case for their class names.

If you'd like to override the pipeline classes name attribute so it isn't derived from the class name, you can set the `custom_name` attribute, like so:

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

class CustomPipeline(MulticlassClassificationPipeline):
    component_graph = ['Imputer', 'One Hot Encoder', 'Logistic Regression Classifier']
    custom_name = 'A custom pipeline name'

print(CustomPipeline.name)

A custom pipeline name
```

## 4.4.4 Override Component Hyperparameter Ranges

To specify custom hyperparameter ranges, set the `custom_hyperparameters` property to be a dictionary where each key-value pair consists of a parameter name and range. AutoML will use this dictionary to override the hyperparameter ranges collected from each component in the component graph.

If the hyperparameter ranges are categorical values, they can be passed in as `skopt.space.Categorical` values.

```
[6]: from skopt.space import Categorical
class CustomPipeline(MulticlassClassificationPipeline):
    component_graph = ['Imputer', 'One Hot Encoder', 'Standard Scaler', 'Logistic_
↳Regression Classifier']

print("Without custom hyperparameters:")
print(CustomPipeline.hyperparameters)
```

(continues on next page)

(continued from previous page)

```
class CustomPipeline(MulticlassClassificationPipeline):
    component_graph = ['Imputer', 'One Hot Encoder', 'Standard Scaler', 'Logistic_
↳Regression Classifier']
    custom_hyperparameters = {
        'Simple Imputer' : {
            'impute_strategy': Categorical(['most_frequent'])
        }
    }

print()
print("With custom hyperparameters:")
print(CustomPipeline.hyperparameters)
```

Without custom hyperparameters:

```
{'Imputer': {'categorical_impute_strategy': ['most_frequent'], 'numeric_impute_
↳strategy': ['mean', 'median', 'most_frequent']}, 'One Hot Encoder': {}, 'Standard_
↳Scaler': {}, 'Logistic Regression Classifier': {'penalty': ['l2'], 'C': Real(low=0.
↳0.1, high=10, prior='uniform', transform='identity')}}
```

With custom hyperparameters:

```
{'Imputer': {'categorical_impute_strategy': ['most_frequent'], 'numeric_impute_
↳strategy': ['mean', 'median', 'most_frequent']}, 'One Hot Encoder': {}, 'Standard_
↳Scaler': {}, 'Logistic Regression Classifier': {'penalty': ['l2'], 'C': Real(low=0.
↳0.1, high=10, prior='uniform', transform='identity')}}
```

To initialize our new custom pipeline class, we must pass in a `parameters` argument. If we want to use the defaults for each component, we can simply pass in an empty dictionary.

```
[7]: CustomPipeline(parameters={})
```

```
[7]: CustomPipeline(parameters={'Imputer':{'categorical_impute_strategy': 'most_frequent',
↳'numeric_impute_strategy': 'mean', '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'},})
```

## 4.4.5 Pipeline Parameters

You can also pass in custom parameters, 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](#).

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

(continues on next page)

(continued from previous page)

```
cp = CustomPipeline(parameters=parameters, random_seed=5)
```

#### 4.4.6 Pipeline Description

You can call `.graph()` to see each component and its parameters. Each component takes in data and feeds it to the next.

```
[9]: cp.graph()
```

```
[9]:
```

```
[10]: nonlinear_cp = CustomNonlinearMulticlassClassificationPipeline({})
      nonlinear_cp.graph()
```

```
[10]:
```

You can see a textual representation of the pipeline by calling `.describe()`:

```
[11]: cp.describe()
```

```
*****
* Custom Pipeline *
*****

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
```

```
[12]: nonlinear_cp.describe()
```

```
*****
* Custom Nonlinear Multiclass Classification Pipeline *
*****
```

(continues on next page)



(continued from previous page)

```

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. Elastic Net Classifier
    * alpha : 0.5
    * l1_ratio : 0.5
    * n_jobs : -1
    * max_iter : 1000
    * penalty : elasticnet
    * loss : log
4. Random Forest Classifier
    * n_estimators : 100
    * max_depth : 6
    * n_jobs : -1
5. Logistic Regression Classifier
    * penalty : l2
    * C : 1.0
    * n_jobs : -1
    * multi_class : auto
    * solver : lbfgs

```

#### 4.4.7 Component Graph

You can use `pipeline.get_component(name)` and provide the component name to access any component (API reference [here](#)):

```

[13]: cp.get_component('Imputer')
[13]: Imputer(categorical_impute_strategy='most_frequent', numeric_impute_strategy='median',
    ↪ categorical_fill_value=None, numeric_fill_value=None)

[14]: nonlinear_cp.get_component('Elastic Net Clf')
[14]: ElasticNetClassifier(alpha=0.5, l1_ratio=0.5, n_jobs=-1, max_iter=1000, penalty=
    ↪ 'elasticnet', loss='log')

```

Alternatively, you can index directly into the pipeline to get a component

```

[15]: first_component = cp[0]
    print(first_component.name)

```

```
Imputer
```

```
[16]: nonlinear_cp['Final Estimator']
```

```
[16]: LogisticRegressionClassifier(penalty='l2', C=1.0, n_jobs=-1, multi_class='auto',  
    ↪ solver='lbfgs')
```

### 4.4.8 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`.

```
[17]: cp.estimator
```

```
[17]: LogisticRegressionClassifier(penalty='l2', C=1.0, n_jobs=-1, multi_class='auto',  
    ↪ solver='lbfgs')
```

### 4.4.9 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.

```
[18]: pipeline.input_feature_names
```

```
[18]: {'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',  
    'od280/od315_of_diluted_wines',  
    'proline']}
```

### 4.4.10 Generate Code

Once you have a pipeline defined in EvalML, 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 code generation is not yet supported for nonlinear pipelines

```
[19]: from evalml.pipelines.utils import generate_pipeline_code
from evalml.pipelines import MulticlassClassificationPipeline
import pandas as pd
from evalml.utils import infer_feature_types, _convert_woodwork_types_wrapper

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.

        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)
        X = _convert_woodwork_types_wrapper(X.to_dataframe())
        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:
```

(continues on next page)

(continued from previous page)

```

        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)

class CustomPipeline(MulticlassClassificationPipeline):
    name = "Custom Pipeline"
    component_graph = ['Imputer', MyDropNullColumns, 'DateTime Featurization Component',
↳ 'One Hot Encoder', 'Random Forest Classifier']
    custom_hyperparameters={
        "Imputer": {
            "numeric_impute_strategy": ['mean', 'median']
        }
    }

pipeline_instance = CustomPipeline(parameters={"Imputer": {"numeric_impute_strategy":
↳ "median"}})
code = generate_pipeline_code(pipeline_instance)
print(code)

# 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()
exec(code)
pipeline.fit(X, y)

import json
from evalml.pipelines.multiclass_classification_pipeline import_
↳ MulticlassClassificationPipeline

class CustomPipeline(MulticlassClassificationPipeline):
    component_graph = [
        'Imputer',
        MyDropNullColumns,
        'DateTime Featurization Component',
        'One Hot Encoder',
        'Random Forest Classifier'
    ]
    custom_hyperparameters = {'Imputer': {'numeric_impute_strategy': ['mean',
↳ 'median']}}
    name = 'Custom Pipeline'

parameters = json.loads("""{
    "Imputer": {
        "categorical_impute_strategy": "most_frequent",
        "numeric_impute_strategy": "median",
        "categorical_fill_value": null,
        "numeric_fill_value": null
    },

```

(continues on next page)

(continued from previous page)

```

    "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
    },
    "One Hot Encoder": {
        "top_n": 10,
        "features_to_encode": null,
        "categories": null,
        "drop": "if_binary",
        "handle_unknown": "ignore",
        "handle_missing": "error"
    },
    "Random Forest Classifier": {
        "n_estimators": 100,
        "max_depth": 6,
        "n_jobs": -1
    }
}

```

```
pipeline = CustomPipeline(parameters)
```

	Number of Features
Boolean	1
Categorical	6
Numeric	5

Number of training examples: 99992

Targets

False 84.82%

True 15.18%

Name: fraud, dtype: object

```

[19]: CustomPipeline(parameters={'Imputer':{'categorical_impute_strategy': 'most_frequent',
↪ 'numeric_impute_strategy': 'median', '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}, '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}},)

```

## 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

class DTBinaryClassificationPipeline(evalml.pipelines.BinaryClassificationPipeline):
    component_graph = ['Simple Imputer', 'Decision Tree Classifier']

X, y = evalml.demos.load_breast_cancer()

pipeline_dt = DTBinaryClassificationPipeline({})
pipeline_dt.fit(X, y)

[1]: DTBinaryClassificationPipeline(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},})
```

#### Tree Visualization

We can visualize the structure of the Decision Tree that was fit to that data, and save it if necessary.

```
[2]: from evalml.model_understanding.graphs import visualize_decision_tree

visualize_decision_tree(pipeline_dt.estimator, max_depth=2, rotate=False, filled=True,
↪ filepath=None)
```

```
[2]:
```

Lets replace the Decision Tree Classifier with a Random Forest Classifier.

```
[3]: class RFBinaryClassificationPipeline(evalml.pipelines.BinaryClassificationPipeline):
    component_graph = ['Simple Imputer', 'Random Forest Classifier']

pipeline = RFBinaryClassificationPipeline({})
pipeline.fit(X, y)
print(pipeline.score(X, y, objectives=['log loss binary']))

OrderedDict([('Log Loss Binary', 0.038403828027876195)])
```

## Feature Importance

We can get the importance associated with each feature of the resulting pipeline

```
[4]: pipeline.feature_importance
[4]:
```

	feature	importance
0	worst perimeter	0.176488
1	worst concave points	0.125260
2	worst radius	0.124161
3	mean concave points	0.086443
4	worst area	0.072465
5	mean concavity	0.072320
6	mean perimeter	0.056685
7	mean area	0.049599
8	area error	0.037229
9	worst concavity	0.028181
10	mean radius	0.023294
11	radius error	0.019457
12	worst texture	0.014990
13	perimeter error	0.014103
14	mean texture	0.013618
15	worst compactness	0.011310
16	worst smoothness	0.011139
17	worst fractal dimension	0.008118
18	worst symmetry	0.007818
19	mean smoothness	0.006152
20	concave points error	0.005887
21	fractal dimension error	0.005059
22	concavity error	0.004510
23	smoothness error	0.004493
24	texture error	0.004476
25	mean compactness	0.004050
26	compactness error	0.003559
27	mean symmetry	0.003243
28	symmetry error	0.003124
29	mean fractal dimension	0.002768

We can also create a bar plot of the feature importances

```
[5]: pipeline.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.

```
[6]: from evalml.model_understanding.graphs import calculate_permutation_importance
calculate_permutation_importance(pipeline, X, y, 'log loss binary')
[6]:
```

	feature	importance
0	worst perimeter	0.083152
1	worst radius	0.078690
2	worst area	0.071237

(continues on next page)

(continued from previous page)

3	worst concave points	0.071188
4	mean concave points	0.043834
5	worst concavity	0.040660
6	mean concavity	0.039079
7	area error	0.037576
8	mean area	0.027190
9	mean perimeter	0.026886
10	worst texture	0.017269
11	mean texture	0.013273
12	perimeter error	0.011904
13	mean radius	0.011215
14	radius error	0.011004
15	worst compactness	0.009072
16	worst smoothness	0.008203
17	mean smoothness	0.005717
18	worst symmetry	0.004561
19	worst fractal dimension	0.004273
20	concavity error	0.004138
21	compactness error	0.003855
22	concave points error	0.003221
23	mean compactness	0.003207
24	smoothness error	0.002949
25	fractal dimension error	0.002712
26	texture error	0.002541
27	mean fractal dimension	0.002305
28	symmetry error	0.002077
29	mean symmetry	0.001675

```
[7]: from evalml.model_understanding.graphs import graph_permutation_importance
graph_permutation_importance(pipeline, X, y, '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.

```
[8]: from evalml.model_understanding.graphs import partial_dependence
partial_dependence(pipeline, X, features='mean radius')
```

```
[8]: feature_values  partial_dependence  class_label
0          9.498540          0.371141    malignant
1          9.610488          0.371141    malignant
2          9.722436          0.371141    malignant
3          9.834384          0.371141    malignant
4          9.946332          0.371141    malignant
..          ...          ...          ...
95         20.133608          0.399560    malignant
96         20.245556          0.399560    malignant
97         20.357504          0.399560    malignant
98         20.469452          0.399560    malignant
99         20.581400          0.399560    malignant
```

[100 rows x 3 columns]



```
[9]: from evalml.model_understanding.graphs import graph_partial_dependence
graph_partial_dependence(pipeline, X, features='mean radius')
```

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.

```
[10]: class FraudPipeline(evalml.pipelines.BinaryClassificationPipeline):
        component_graph = ["DateTime Featurization Component", "One Hot Encoder", "Random_
        ↳Forest Classifier"]

X_fraud, y_fraud = evalml.demos.load_fraud(100, verbose=False)

fraud_pipeline = FraudPipeline({})
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.

```
[11]: partial_dependence(pipeline, X, features=('worst perimeter', 'worst radius'), grid_
        ↳resolution=10)
```

```
[11]:      10.5072  12.193377777777776  13.879555555555555  \
67.733600  0.264908                0.267211          0.274328
79.363867  0.265840                0.268142          0.275260
90.994133  0.273397                0.275699          0.282817
102.624400 0.298379                0.300681          0.307799
114.254667 0.395976                0.398278          0.404798
125.884933 0.426266                0.428569          0.435089
137.515200 0.442004                0.444307          0.450827
149.145467 0.442004                0.444307          0.450827
160.775733 0.442004                0.444307          0.450827
172.406000 0.442004                0.444307          0.450827

      15.565733333333334  17.251911111111113  18.938088888888892  \
67.733600                0.286943          0.405865          0.442701
79.363867                0.287875          0.405865          0.442701
90.994133                0.295432          0.411805          0.448641
102.624400                0.323472          0.436371          0.473207
114.254667                0.417739          0.530867          0.567702
125.884933                0.450433          0.556594          0.593430
137.515200                0.466171          0.574301          0.611137
149.145467                0.466171          0.574301          0.611137
160.775733                0.466171          0.574301          0.611137
172.406000                0.466171          0.574301          0.611137

      20.624266666666667  22.310444444444443  23.996622222222222  \
67.733600                0.444406          0.444406          0.444406
79.363867                0.444406          0.444406          0.444406
90.994133                0.450346          0.450346          0.450346
102.624400                0.474911          0.474911          0.474911
```

(continues on next page)

(continued from previous page)

114.254667	0.571516	0.571797	0.571797
125.884933	0.597244	0.597525	0.597525
137.515200	0.614950	0.615232	0.615232
149.145467	0.614950	0.615232	0.615232
160.775733	0.614950	0.615232	0.615232
172.406000	0.614950	0.615232	0.615232
	25.6828	class_label	
67.733600	0.444406	malignant	
79.363867	0.444406	malignant	
90.994133	0.450346	malignant	
102.624400	0.474911	malignant	
114.254667	0.571797	malignant	
125.884933	0.597525	malignant	
137.515200	0.615232	malignant	
149.145467	0.615232	malignant	
160.775733	0.615232	malignant	
172.406000	0.615232	malignant	

```
[12]: graph_partial_dependence(pipeline, X, features=('worst_perimeter', 'worst_radius'),
    ↪ grid_resolution=10)
```

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.

```
[13]: from evalml.model_understanding.graphs import confusion_matrix
y_pred = pipeline.predict(X)
confusion_matrix(y, y_pred)
```

```
[13]:
```

	benign	malignant
benign	1.000000	0.000000
malignant	0.009434	0.990566

```
[14]: from evalml.model_understanding.graphs import graph_confusion_matrix
y_pred = pipeline.predict(X)
graph_confusion_matrix(y, 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.

```
[15]: 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.to_series().map({'benign': 0, 'malignant': 1})
y_encoded = ww.DataColumn(y_encoded)
y_pred_proba = pipeline.predict_proba(X) ["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.

```
[16]: from evalml.model_understanding.graphs import graph_roc_curve
# get the predicted probabilities associated with the "malignant" label
y_pred_proba = pipeline.predict_proba(X) ["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.

```
[17]: class RFMulticlassClassificationPipeline(evalml.pipelines.
↳ MulticlassClassificationPipeline):
    component_graph = ['Simple Imputer', 'Random Forest Classifier']

X_multi, y_multi = evalml.demos.load_wine()

pipeline_multi = RFMulticlassClassificationPipeline({})
pipeline_multi.fit(X_multi, y_multi)

y_pred_proba = pipeline_multi.predict_proba(X_multi)
graph_roc_curve(y_multi, y_pred_proba)
```

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`.

```
[18]: from evalml.model_understanding.graphs import binary_objective_vs_threshold
      binary_objective_vs_threshold(pipeline, X, y, 'f1', steps=100)
```

```
[18]:
```

	threshold	score
0	0.00	0.542894
1	0.01	0.750442
2	0.02	0.815385
3	0.03	0.848000
4	0.04	0.874227
..	...	...
96	0.96	0.854054
97	0.97	0.835165
98	0.98	0.805634
99	0.99	0.722892
100	1.00	0.000000

```
[101 rows x 2 columns]
```

```
[19]: from evalml.model_understanding.graphs import graph_binary_objective_vs_threshold
      graph_binary_objective_vs_threshold(pipeline, X, y, '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.

```
[20]: from evalml.model_understanding.graphs import graph_prediction_vs_actual

class LinearRegressionPipeline(evalml.pipelines.RegressionPipeline):
    component_graph = ['One Hot Encoder', 'Linear Regressor']

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 = LinearRegressionPipeline({})
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)
```

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

## 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, input_features=X, y=None, indices_to_
    explain=[3],
                           top_k_features=6, include_shap_values=True)

print(table)
```

RFBinary Classification Pipeline

```
{'Simple Imputer': {'impute_strategy': 'most_frequent', 'fill_value': None}, 'Random_
    Forest Classifier': {'n_estimators': 100, 'max_depth': 6, 'n_jobs': -1}}
```

1 of 1

	Feature Name	Feature Value	Contribution to Prediction
SHAP Value			
	=====		
0.20	worst concave points	0.26	++
0.11	mean concave points	0.11	+
0.08	mean concavity	0.24	+
0.05	worst concavity	0.69	+
-0.05	worst perimeter	98.87	-
-0.05	worst radius	14.91	-

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.

This functionality is currently **not supported** for **XGBoost** models or **CatBoost multiclass** classifiers.

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, input_features=X, y=y, indices_to_
    explain=[0, 4, 9], include_shap_values=True,
```

(continues on next page)

(continued from previous page)

```
output_format='text')
print(report)
```

RFBinary Classification Pipeline

```
{'Simple Imputer': {'impute_strategy': 'most_frequent', 'fill_value': None}, 'Random_
↳Forest Classifier': {'n_estimators': 100, 'max_depth': 6, 'n_jobs': -1}}
```

1 of 3

	Feature Name	Feature Value	Contribution to Prediction	
↳SHAP Value				
↳	=====			
↳ 0.09	worst concave points	0.27	+	↳
↳ 0.09	worst perimeter	184.60	+	↳
↳ 0.08	worst radius	25.38	+	↳

2 of 3

	Feature Name	Feature Value	Contribution to Prediction	
↳SHAP Value				
↳	=====			
↳ 0.11	worst perimeter	152.20	+	↳
↳ 0.09	worst radius	22.54	+	↳
↳ 0.08	worst concave points	0.16	+	↳

3 of 3

	Feature Name	Feature Value	Contribution to Prediction	
↳SHAP Value				
↳	=====			
↳ 0.20	worst concave points	0.22	++	↳
↳ 0.11	mean concave points	0.09	+	↳
↳ 0.08	mean concavity	0.23	+	↳

## 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, input_features=X, y_true=y,
      ↪ include_shap_values=True, top_k_features=6,
      ↪ num_to_explain=2)

print(report)
```

RFBinary Classification Pipeline

```
{'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: 0.0, malignant: 1.0]
Predicted Value: malignant
Target Value: malignant
Cross Entropy: 0.0
Index ID: 168
```

	Feature Name	Feature Value	Contribution to Prediction
↪ SHAP Value			
↪ =====			
↪ 0.10	worst perimeter	155.30	+
↪ 0.08	worst radius	23.14	+
↪ 0.08	worst concave points	0.17	+
↪ 0.06	worst area	1660.00	+
↪ 0.05	mean concave points	0.10	+
↪ 0.04	area error	122.30	+

Best 2 of 2

```
Predicted Probabilities: [benign: 0.0, malignant: 1.0]
Predicted Value: malignant
```

(continues on next page)

(continued from previous page)

Target Value: malignant Cross Entropy: 0.0 Index ID: 564			
	Feature Name	Feature Value	Contribution to Prediction
↪SHAP Value			
↪	=====		
↪ 0.10	worst perimeter	166.10	+
↪ 0.08	worst radius	25.45	+
↪ 0.08	worst concave points	0.22	+
↪ 0.06	worst area	2027.00	+
↪ 0.05	mean concave points	0.14	+
↪ 0.05	mean concavity	0.24	+
Worst 1 of 2			
Predicted Probabilities: [benign: 0.552, malignant: 0.448] Predicted Value: benign Target Value: malignant Cross Entropy: 0.802 Index ID: 40			
	Feature Name	Feature Value	Contribution to Prediction
↪SHAP Value			
↪	=====		
↪0.04	smoothness error	0.00	+
↪0.03	mean texture	21.58	+
↪0.02	worst texture	30.25	+
↪0.02	worst area	787.90	+
↪0.03	worst radius	15.93	-
↪0.03	mean concave points	0.02	-
Worst 2 of 2			
Predicted Probabilities: [benign: 0.788, malignant: 0.212] Predicted Value: benign Target Value: malignant Cross Entropy: 1.55 Index ID: 135			
	Feature Name	Feature Value	Contribution to Prediction
↪SHAP Value			

(continues on next page)



(continued from previous page)

```

=====
↪          worst texture          33.37          +          ↪
↪ 0.05          mean texture      22.47          +          ↪
↪ 0.03          mean concave points 0.03          -          ↪
↪ -0.03         worst concave points 0.09          -          ↪
↪ -0.04         worst radius       14.49          -          ↪
↪ -0.05         worst perimeter    92.04          -          ↪
↪ -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)
    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, input_features=X, ↪
↪ indices_to_explain=[3],
↪ y=y, top_k_features=6, include_shap_
↪ values=True,
↪ output_format="dict")
print(json.dumps(single_prediction_report, indent=2))

{
  "explanations": [

```

(continues on next page)

(continued from previous page)

```

{
  "explanations": [
    {
      "feature_names": [
        "worst concave points",
        "mean concave points",
        "mean concavity",
        "worst concavity",
        "worst perimeter",
        "worst radius"
      ],
      "feature_values": [
        0.2575,
        0.1052,
        0.2414,
        0.6869,
        98.87,
        14.91
      ],
      "qualitative_explanation": [
        "++",
        "+",
        "+",
        "+",
        "-",
        "-"
      ],
      "quantitative_explanation": [
        0.19966729417702012,
        0.10648831456429969,
        0.07869244977813485,
        0.05150874542350735,
        -0.04930428857229847,
        -0.05034083333027343
      ],
      "drill_down": {},
      "class_name": "malignant"
    }
  ]
}

```

### Single prediction as a dataframe

```

[25]: single_prediction_report = explain_predictions(pipeline=pipeline, input_features=X,
↳ indices_to_explain=[3],
                                                    y=y, top_k_features=6, include_shap_
↳ values=True,
                                                    output_format="dataframe")
single_prediction_report

```

```

[25]:
   feature_names  feature_values  qualitative_explanation \
0  worst concave points          0.2575                ++
1   mean concave points          0.1052                 +

```

(continues on next page)

(continued from previous page)

2	mean concavity	0.2414	+
3	worst concavity	0.6869	+
4	worst perimeter	98.8700	-
5	worst radius	14.9100	-

	quantitative_explanation	class_name	prediction_number
0	0.199667	malignant	0
1	0.106488	malignant	0
2	0.078692	malignant	0
3	0.051509	malignant	0
4	-0.049304	malignant	0
5	-0.050341	malignant	0

### Best and worst predictions as a dictionary

```
[26]: report = explain_predictions_best_worst(pipeline=pipeline, 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": 0.0,
          "malignant": 1.0
        },
        "predicted_value": "malignant",
        "target_value": "malignant",
        "error_name": "Cross Entropy",
        "error_value": 9.95074382629983e-05,
        "index_id": 168
      },
      "explanations": [
        {
          "feature_names": [
            "worst perimeter",
            "worst radius",
            "worst concave points",
            "worst area",
            "mean concave points",
            "area error"
          ],
          "feature_values": [
            155.3,
            23.14,
            0.1721,
            1660.0,
            0.1043,
```

(continues on next page)

(continued from previous page)

```

        122.3
    ],
    "qualitative_explanation": [
        "+",
        "+",
        "+",
        "+",
        "+",
        "+"
    ],
    "quantitative_explanation": [
        0.09988982304983156,
        0.08240174808629956,
        0.07868368954615064,
        0.06242860386204596,
        0.051970789425386396,
        0.04459155806887927
    ],
    "drill_down": {},
    "class_name": "malignant"
}
]
},
{
    "rank": {
        "prefix": "worst",
        "index": 1
    },
    "predicted_values": {
        "probabilities": {
            "benign": 0.788,
            "malignant": 0.212
        },
        "predicted_value": "benign",
        "target_value": "malignant",
        "error_name": "Cross Entropy",
        "error_value": 1.5499050281608746,
        "index_id": 135
    },
    "explanations": [
        {
            "feature_names": [
                "worst texture",
                "mean texture",
                "mean concave points",
                "worst concave points",
                "worst radius",
                "worst perimeter"
            ],
            "feature_values": [
                33.37,
                22.47,
                0.02704,
                0.09331,
                14.49,
                92.04
            ]
        }
    ]
}

```

(continues on next page)

(continued from previous page)

```

    "qualitative_explanation": [
        "+",
        "+",
        "-",
        "-",
        "-",
        "-"
    ],
    "quantitative_explanation": [
        0.05245422607466413,
        0.03035933540832274,
        -0.03461744299818247,
        -0.04174884967530769,
        -0.0491285663898271,
        -0.05666940833106337
    ],
    "drill_down": {},
    "class_name": "malignant"
}
]
}
]
}

```

### Best and worst predictions as a dataframe

```

[27]: report = explain_predictions_best_worst(pipeline=pipeline, input_features=X, y_true=y,
                                             num_to_explain=1, top_k_features=6,
                                             include_shap_values=True, output_format=
↳ "dataframe")
report

```

```

[27]:
   feature_names  feature_values  qualitative_explanation  \
0    worst perimeter      155.30000                +
1    worst radius        23.14000                +
2  worst concave points      0.17210                +
3    worst area      1660.00000                +
4  mean concave points      0.10430                +
5    area error      122.30000                +
6  worst texture      33.37000                +
7    mean texture      22.47000                +
8  mean concave points      0.02704                -
9  worst concave points      0.09331                -
10   worst radius      14.49000                -
11   worst perimeter      92.04000                -

   quantitative_explanation  class_name  label_benign_probability  \
0          0.099890  malignant                0.000
1          0.082402  malignant                0.000
2          0.078684  malignant                0.000
3          0.062429  malignant                0.000
4          0.051971  malignant                0.000
5          0.044592  malignant                0.000
6          0.052454  malignant                0.788
7          0.030359  malignant                0.788

```

(continues on next page)

(continued from previous page)

8	-0.034617	malignant		0.788
9	-0.041749	malignant		0.788
10	-0.049129	malignant		0.788
11	-0.056669	malignant		0.788

	label_malignant_probability	predicted_value	target_value	error_name	\
0	1.000	malignant	malignant	Cross Entropy	
1	1.000	malignant	malignant	Cross Entropy	
2	1.000	malignant	malignant	Cross Entropy	
3	1.000	malignant	malignant	Cross Entropy	
4	1.000	malignant	malignant	Cross Entropy	
5	1.000	malignant	malignant	Cross Entropy	
6	0.212	benign	malignant	Cross Entropy	
7	0.212	benign	malignant	Cross Entropy	
8	0.212	benign	malignant	Cross Entropy	
9	0.212	benign	malignant	Cross Entropy	
10	0.212	benign	malignant	Cross Entropy	
11	0.212	benign	malignant	Cross Entropy	

	error_value	index_id	rank	prefix
0	0.000100	168	1	best
1	0.000100	168	1	best
2	0.000100	168	1	best
3	0.000100	168	1	best
4	0.000100	168	1	best
5	0.000100	168	1	best
6	1.549905	135	1	worst
7	1.549905	135	1	worst
8	1.549905	135	1	worst
9	1.549905	135	1	worst
10	1.549905	135	1	worst
11	1.549905	135	1	worst

## 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
```

(continues on next page)

(continued from previous page)

```

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_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

### 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'])

```

(continues on next page)

(continued from previous page)

```
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']:
    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']:
```

(continues on next page)



(continued from previous page)

```
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: [1, 0]

## 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])
```

(continues on next page)

(continued from previous page)

```

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 Data Check

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)

```

(continues on next page)

(continued from previous page)

```

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 Data Check

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 Data Check

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]

```

(continues on next page)

(continued from previous page)

```

        'not_sparse': [float(1) for x in range(10)]) # [1,1,1,1,1,1,1,1,1,1,
→ 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.

### 4.6.3 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.

### 4.6.4 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.

```
[13]: 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.
```

### 4.6.5 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.

```
[14]: 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])
```

## 4.6.6 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`
- `ClassImbalanceDataCheck` (for classification problem types)
- `NoVarianceDataCheck`
- `DateTimeNaNDataCheck`
- `NaturalLanguageNaNDataCheck`

## 4.6.7 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.

```
[15]: # 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,
                                                                           "objective
↳": objective}})
        else:
```

(continues on next page)

(continued from previous page)

```

        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

```

[16]: # 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: 0.23.0
EvalML installation directory: /home/docs/checkouts/readthedocs.org/user_builds/
  ↳ feature-labs-inc-evalml/envs/v0.23.0/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: 6.3G
```

### INSTALLED VERSIONS

```
-----
zict: 2.0.0
xgboost: 1.2.1
woodwork: 0.0.11
widgetsnbextension: 3.5.1
wheel: 0.36.2
webencodings: 0.5.1
wcwidth: 0.2.5
urllib3: 1.26.4
traitlets: 5.0.5
tqdm: 4.60.0
tornado: 6.1
toolz: 0.11.1
threadpoolctl: 2.1.0
texttable: 1.6.3
testpath: 0.4.4
terminado: 0.9.4
tblib: 1.7.0
statsmodels: 0.12.2
sphinxcontrib-websupport: 1.2.4
sphinxcontrib-serializinghtml: 1.1.4
sphinxcontrib-qthelp: 1.0.3
sphinxcontrib-jsmath: 1.0.1
sphinxcontrib-htmlhelp: 1.0.3
sphinxcontrib-devhelp: 1.0.2
sphinxcontrib-applehelp: 1.0.2
sphinx: 3.5.4
sphinx-rtd-theme: 0.4.3
soupsieve: 2.2.1
sortedcontainers: 2.3.0
```

(continues on next page)



(continued from previous page)

```
snowballstemmer: 2.1.0
slicer: 0.0.7
sktime: 0.6.0
six: 1.15.0
shap: 0.39.0
setuptools: 56.0.0
send2trash: 1.5.0
seaborn: 0.11.1
scipy: 1.6.2
scikit-optimize: 0.8.1
scikit-learn: 0.24.1
retrying: 1.3.3
requirements-parser: 0.2.0
requests: 2.25.1
regex: 2021.4.4
recommonmark: 0.5.0
readthedocs-sphinx-ext: 2.1.4
pyzmq: 21.0.2
pyyaml: 5.4.1
pytz: 2021.1
python-dateutil: 2.8.1
pyrsistent: 0.17.3
pyparsing: 2.4.7
pygments: 2.8.1
pydata-sphinx-theme: 0.6.1
pycparser: 2.20
pyaml: 20.4.0
ptyprocess: 0.7.0
psutil: 5.8.0
prompt-toolkit: 3.0.18
prometheus-client: 0.10.1
plotly: 4.14.3
pip: 21.0.1
pillow: 8.2.0
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: 1.2.4
packaging: 20.9
numpy: 1.20.2
numba: 0.53.1
notebook: 6.3.0
nltk: 3.6.2
nlp-primitives: 1.1.0
networkx: 2.5.1
nest-asyncio: 1.5.1
nbsphinx: 0.8.3
nbformat: 5.1.3
nbconvert: 6.0.7
nbclient: 0.5.3
msgpack: 1.0.2
mock: 1.0.1
mistune: 0.8.4
matplotlib: 3.4.1
```

(continues on next page)

(continued from previous page)

```
markupsafe: 1.1.1
loket: 0.2.1
llvmlite: 0.36.0
lightgbm: 3.0.0
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: 2.11.3
jedi: 0.18.0
ipywidgets: 7.6.3
ipython: 7.22.0
ipython-genutils: 0.2.0
ipykernel: 5.5.3
imbalanced-learn: 0.8.0
imagesize: 1.2.0
idna: 2.10
heapdict: 1.0.1
graphviz: 0.16
future: 0.18.2
fsspec: 2021.4.0
featuretools: 0.23.3
evalml: 0.23.0
entrypoints: 0.3
docutils: 0.16
distributed: 2021.4.0
defusedxml: 0.7.1
decorator: 4.4.2
dask: 2021.4.0
cyclor: 0.10.0
commonmark: 0.8.1
colorama: 0.4.4
cloudpickle: 1.6.0
click: 7.1.2
chardet: 4.0.0
cffi: 1.14.5
certifi: 2020.12.5
category-encoders: 2.2.2
catboost: 0.25.1
bleach: 3.3.0
beautifulsoup4: 4.9.3
backcall: 0.2.0
babel: 2.9.0
attrs: 20.3.0
async-generator: 1.10
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](#).

### 4.8.3 Q: How does EvalML handle categorical encoding?

EvalML provides a *one-hot-encoding component* 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](#).

### 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

<code>load_fraud</code>	Load credit card fraud dataset.
<code>load_wine</code>	Load wine dataset.
<code>load_breast_cancer</code>	Load breast cancer dataset.
<code>load_diabetes</code>	Load diabetes dataset.
<code>load_churn</code>	Load credit card fraud dataset.

#### 5.1.1 `evalml.demos.load_fraud`

`evalml.demos.load_fraud(n_rows=None, verbose=True, return_pandas=False)`

**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** Union[(`ww.DataTable`, `ww.DataColumn`), (`pd.DataFrame`, `pd.Series`)]

#### 5.1.2 `evalml.demos.load_wine`

`evalml.demos.load_wine(return_pandas=False)`

Load wine dataset. Multiclass problem.

**Returns** X and y

**Return type** Union[(`ww.DataTable`, `ww.DataColumn`), (`pd.DataFrame`, `pd.Series`)]

### 5.1.3 evalml.demos.load\_breast\_cancer

`evalml.demos.load_breast_cancer(return_pandas=False)`

Load breast cancer dataset. Binary classification problem.

**Returns** X and y

**Return type** Union[(ww.DataTable, ww.DataColumn), (pd.DataFrame, pd.Series)]

### 5.1.4 evalml.demos.load\_diabetes

`evalml.demos.load_diabetes(return_pandas=False)`

Load diabetes dataset. Regression problem

**Returns** X and y

**Return type** Union[(ww.DataTable, ww.DataColumn), (pd.DataFrame, pd.Series)]

### 5.1.5 evalml.demos.load\_churn

`evalml.demos.load_churn(n_rows=None, verbose=True, return_pandas=False)`

**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** Union[(ww.DataTable, ww.DataColumn), (pd.DataFrame, pd.Series)]

## 5.2 Preprocessing

Utilities to preprocess data before using evalml.

<code>load_data</code>	Load features and target from file.
<code>drop_nan_target_rows</code>	Drops rows in X and y when row in the target y has a value of NaN.
<code>target_distribution</code>	Get the target distributions.
<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.

### 5.2.1 evalml.preprocessing.load\_data

`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** ww.DataTable, ww.DataColumn

### 5.2.2 evalml.preprocessing.drop\_nan\_target\_rows

`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

### 5.2.3 evalml.preprocessing.target\_distribution

`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

### 5.2.4 evalml.preprocessing.number\_of\_features

`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

### 5.2.5 evalml.preprocessing.split\_data

`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** (*ww.DataTable*, *pd.DataFrame* or *np.ndarray*) – data of shape [n\_samples, n\_features]
- **y** (*ww.DataColumn*, *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 `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** *ww.DataTable*, *ww.DataTable*, *ww.DataColumn*, *ww.DataColumn*

### 5.2.6 Data Splitter Classes

Data splitter classes for imbalanced classification datasets.

<i>BalancedClassificationDataTVSplit</i>	Data splitter for generating training and validation split using Balanced Classification Data Sampler.
<i>BalancedClassificationDataCVSplit</i>	Data splitter for generating k-fold cross-validation split using Balanced Classification Data Sampler.
<i>KMeansSMOTECVSplit</i>	Splits the data into KFold cross validation sets and balances the training data using K-Means SMOTE.
<i>KMeansSMOTETVSplit</i>	Splits the data into training and validation sets and balances the training data using K-Means SMOTE.
<i>SMOTETomekCVSplit</i>	Splits the data into KFold cross validation sets and uses SMOTE + Tomek links to balance the training data.
<i>SMOTETomekTVSplit</i>	Splits the data into training and validation sets and uses SMOTE + Tomek links balance the training data.
<i>RandomUnderSamplerCVSplit</i>	Splits the training data into KFold cross validation sets and uses RandomUnderSampler to balance the training data.
<i>RandomUnderSamplerTVSplit</i>	Splits the data into training and validation sets and uses RandomUnderSampler to balance the training data.
<i>SMOTENCCVSplit</i>	Splits the data into KFold cross validation sets and uses SMOTENC to balance the training data.
<i>SMOTENCTVSplit</i>	Splits the data into training and validation sets and uses SMOTENC to balance the training data.



**evalml.preprocessing.BalancedClassificationDataTVSplit**

```
class evalml.preprocessing.BalancedClassificationDataTVSplit (sampling_ratio=0.25,  

min_samples=100,  

min_percentage=0.1,  

test_size=0.25,  

shuffle=True,  

random_seed=0)
```

Data splitter for generating training and validation split using Balanced Classification Data Sampler.

**Methods**

<code>__init__</code>	Create Balanced Classification Data TV splitter
<code>get_n_splits</code>	Returns the number of splits of this object.
<code>split</code>	Splits and returns the indices of the training and testing using the data sampler provided.
<code>transform_sample</code>	Transforms the input data with the balancing strategy.

**evalml.preprocessing.BalancedClassificationDataTVSplit.\_\_init\_\_**

```
BalancedClassificationDataTVSplit.__init__ (sampling_ratio=0.25,  

min_samples=100, min_percentage=0.1,  

test_size=0.25, shuffle=True, random_seed=0)
```

Create Balanced Classification Data TV splitter

**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.
- **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.
- **test\_size** (*float*) – The size of the test split. Defaults to 0.25.
- **shuffle** (*bool*) – Whether or not to shuffle the data before splitting. Defaults to True.
- **random\_seed** (*int*) – The seed to use for random sampling. Defaults to 0.

**evalml.preprocessing.BalancedClassificationDataTVSplit.get\_n\_splits**

```
BalancedClassificationDataTVSplit.get_n_splits()
```

Returns the number of splits of this object.

**evalml.preprocessing.BalancedClassificationDataTVSplit.split**

```
BalancedClassificationDataTVSplit.split(X, y)
```

Splits and returns the indices of the training and testing using the data sampler provided. :param X: DataTable of points to split :type X: ww.DataTable :param y: DataColumn of points to split :type y: ww.DataTable

**Returns** A tuple containing the resulting train and test indices, post sampling.

**Return type** tuple(train, test)

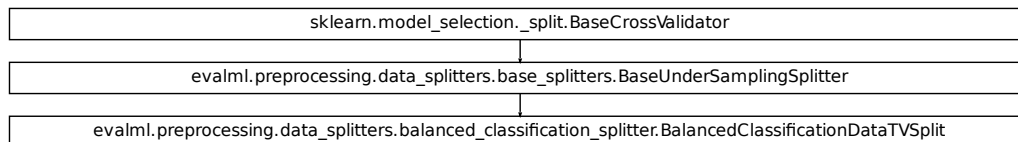
**evalml.preprocessing.BalancedClassificationDataTVSplit.transform\_sample**

```
BalancedClassificationDataTVSplit.transform_sample(X, y)
```

Transforms the input data with the balancing strategy. :param X: DataTable of points to split :type X: ww.DataTable :param y: DataColumn of points to split :type y: ww.DataTable

**Returns** List of indices to keep

**Return type** list

**Class Inheritance****evalml.preprocessing.BalancedClassificationDataCVSplit**

```
class evalml.preprocessing.BalancedClassificationDataCVSplit (sampling_ratio=0.25,  
                                                             min_samples=100,  
                                                             min_percentage=0.1,  
                                                             n_splits=3,    shuffle=True,    random_seed=0)
```

Data splitter for generating k-fold cross-validation split using Balanced Classification Data Sampler.

## Methods

<code>__init__</code>	Create Balanced Classification Data CV splitter
<code>get_n_splits</code>	Returns the number of splits of this object.
<code>split</code>	Splits and returns the indices of the training and testing using the data sampler provided.
<code>transform_sample</code>	Transforms the input data with the balancing strategy.

### `evalml.preprocessing.BalancedClassificationDataCVSplit.__init__`

`BalancedClassificationDataCVSplit.__init__` (*sampling\_ratio=0.25*,  
*min\_samples=100*, *min\_percentage=0.1*,  
*n\_splits=3*, *shuffle=True*, *random\_seed=0*)

Create Balanced Classification Data CV splitter

#### 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.
- **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.
- **n\_splits** (*int*) – The number of splits to use for cross validation. Defaults to 3.
- **shuffle** (*bool*) – Whether or not to shuffle the data before splitting. Defaults to True.
- **random\_seed** (*int*) – The seed to use for random sampling. Defaults to 0.

### `evalml.preprocessing.BalancedClassificationDataCVSplit.get_n_splits`

`BalancedClassificationDataCVSplit.get_n_splits` ()  
 Returns the number of splits of this object.

### `evalml.preprocessing.BalancedClassificationDataCVSplit.split`

`BalancedClassificationDataCVSplit.split(X, y)`

Splits and returns the indices of the training and testing using the data sampler provided. :param X: DataTable of points to split :type X: ww.DataTable :param y: DataColumn of points to split :type y: ww.DataTable

**Returns** A tuple containing the resulting train and test indices, post sampling.

**Return type** tuple(train, test)

### `evalml.preprocessing.BalancedClassificationDataCVSplit.transform_sample`

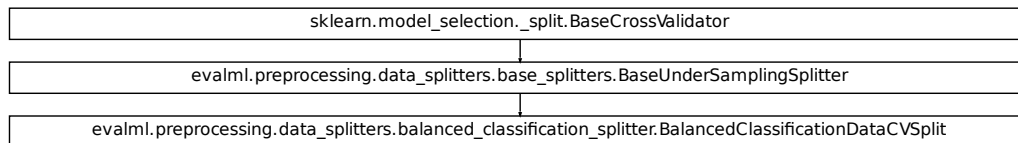
`BalancedClassificationDataCVSplit.transform_sample(X, y)`

Transforms the input data with the balancing strategy. :param X: DataTable of points to split :type X: ww.DataTable :param y: DataColumn of points to split :type y: ww.DataTable

**Returns** List of indices to keep

**Return type** list

## Class Inheritance



### `evalml.preprocessing.KMeansSMOTECVSplit`

**class** `evalml.preprocessing.KMeansSMOTECVSplit` (*sampling\_strategy='auto',  
k\_neighbors=2, n\_splits=3, shuffle=True,  
random\_seed=0, \*\*kwargs*)

Splits the data into KFold cross validation sets and balances the training data using K-Means SMOTE. Keeps the validation data the same. Works only on continuous, numeric data.

## Methods

<code>__init__</code>	Create a TV or CV data splitter instance
<code>get_n_splits</code>	Returns the number of splits of this object.
<code>split</code>	Splits and returns the sampled training data using the data sampler provided.
<code>transform_sample</code>	Transforms the input data with the balancing strategy.

### `evalml.preprocessing.KMeansSMOTECVSplit.__init__`

`KMeansSMOTECVSplit.__init__(sampling_strategy='auto', k_neighbors=2, n_splits=3, shuffle=True, random_seed=0, **kwargs)`

Create a TV or CV data splitter instance

#### Parameters

- **sampler** (*sampler instance*) – The sampler instance to use for resampling the training data. Must have a *fit\_resample* method. Defaults to None, which is equivalent to regular TV split.
- **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.
- **n\_splits** (*int*) – How many CV folds to use. Defaults to 3.
- **shuffle** (*bool*) – Whether or not to shuffle the data. Defaults to True.
- **split\_type** (*str*) – Whether to use TV or CV split. Defaults to TV.
- **random\_seed** (*int*) – Random seed for the splitter. Defaults to 0

### `evalml.preprocessing.KMeansSMOTECVSplit.get_n_splits`

`KMeansSMOTECVSplit.get_n_splits()`

Returns the number of splits of this object.

### `evalml.preprocessing.KMeansSMOTECVSplit.split`

`KMeansSMOTECVSplit.split(X, y)`

Splits and returns the sampled training data using the data sampler provided.

#### Parameters

- **X** (*ww.DataTable*) – DataTable of points to split
- **y** (*ww.DataTable*) – DataColumn of points to split

**Returns** A tuple containing the resulting ((X\_train, y\_train), (X\_test, y\_test)) post-transformation.

**Return type** tuple((pd.DataFrame, pd.Series), (pd.DataFrame, pd.Series))

**evalml.preprocessing.KMeansSMOTECVSplit.transform\_sample**

KMeansSMOTECVSplit.**transform\_sample**(X, y)

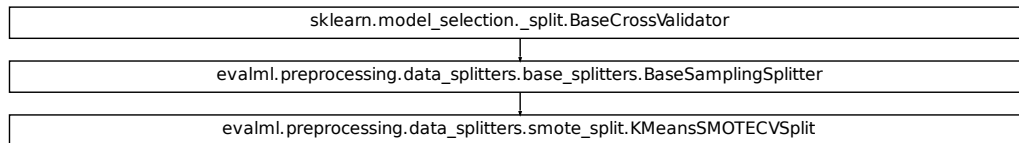
Transforms the input data with the balancing strategy.

**Parameters**

- **X** (*ww.DataTable*) – DataTable of points to split
- **y** (*ww.DataTable*) – DataColumn of points to split

**Returns** A tuple containing the resulting X and y post-transformation.

**Return type** tuple(pd.DataFrame, pd.Series)

**Class Inheritance****evalml.preprocessing.KMeansSMOTETVSplit**

**class** evalml.preprocessing.**KMeansSMOTETVSplit** (*sampling\_strategy='auto', k\_neighbors=2, test\_size=None, random\_seed=0, \*\*kwargs*)

Splits the data into training and validation sets and balances the training data using K-Means SMOTE. Keeps the validation data the same. Works only on continuous, numeric data.

**Methods**

<code>__init__</code>	Create a TV or CV data splitter instance
<code>get_n_splits</code>	Returns the number of splits of this object.
<code>split</code>	Splits and returns the sampled training data using the data sampler provided.
<code>transform_sample</code>	Transforms the input data with the balancing strategy.

**evalml.preprocessing.KMeansSMOTETVSplit.\_\_init\_\_**

`KMeansSMOTETVSplit.__init__(sampling_strategy='auto', k_neighbors=2, test_size=None, random_seed=0, **kwargs)`

Create a TV or CV data splitter instance

**Parameters**

- **sampler** (*sampler instance*) – The sampler instance to use for resampling the training data. Must have a *fit\_resample* method. Defaults to None, which is equivalent to regular TV split.
- **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.
- **n\_splits** (*int*) – How many CV folds to use. Defaults to 3.
- **shuffle** (*bool*) – Whether or not to shuffle the data. Defaults to True.
- **split\_type** (*str*) – Whether to use TV or CV split. Defaults to TV.
- **random\_seed** (*int*) – Random seed for the splitter. Defaults to 0

**evalml.preprocessing.KMeansSMOTETVSplit.get\_n\_splits**

`KMeansSMOTETVSplit.get_n_splits()`

Returns the number of splits of this object.

**evalml.preprocessing.KMeansSMOTETVSplit.split**

`KMeansSMOTETVSplit.split(X, y)`

Splits and returns the sampled training data using the data sampler provided.

**Parameters**

- **X** (*ww.DataTable*) – DataTable of points to split
- **y** (*ww.DataTable*) – DataColumn of points to split

**Returns** A tuple containing the resulting ((X\_train, y\_train), (X\_test, y\_test)) post-transformation.

**Return type** tuple((pd.DataFrame, pd.Series), (pd.DataFrame, pd.Series))

**evalml.preprocessing.KMeansSMOTETVSplit.transform\_sample**

`KMeansSMOTETVSplit.transform_sample(X, y)`

Transforms the input data with the balancing strategy.

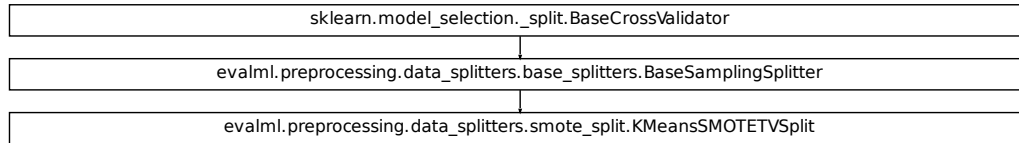
**Parameters**

- **X** (*ww.DataTable*) – DataTable of points to split
- **y** (*ww.DataTable*) – DataColumn of points to split

**Returns** A tuple containing the resulting X and y post-transformation.

**Return type** tuple(pd.DataFrame, pd.Series)

## Class Inheritance



### evalml.preprocessing.SMOTETomekCVSplit

**class** evalml.preprocessing.**SMOTETomekCVSplit** (*sampling\_strategy='auto', n\_splits=3, shuffle=True, n\_jobs=-1, random\_seed=0*)

Splits the data into KFold cross validation sets and uses SMOTE + Tomek links to balance the training data. Keeps the validation data the same. Works only on continuous, numeric data.

#### Methods

<code>__init__</code>	Create a TV or CV data splitter instance
<code>get_n_splits</code>	Returns the number of splits of this object.
<code>split</code>	Splits and returns the sampled training data using the data sampler provided.
<code>transform_sample</code>	Transforms the input data with the balancing strategy.

### evalml.preprocessing.SMOTETomekCVSplit.\_\_init\_\_

SMOTETomekCVSplit.**\_\_init\_\_** (*sampling\_strategy='auto', n\_splits=3, shuffle=True, n\_jobs=-1, random\_seed=0*)

Create a TV or CV data splitter instance

#### Parameters

- **sampler** (*sampler instance*) – The sampler instance to use for resampling the training data. Must have a *fit\_resample* method. Defaults to None, which is equivalent to regular TV split.
- **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.
- **n\_splits** (*int*) – How many CV folds to use. Defaults to 3.
- **shuffle** (*bool*) – Whether or not to shuffle the data. Defaults to True.
- **split\_type** (*str*) – Whether to use TV or CV split. Defaults to TV.
- **random\_seed** (*int*) – Random seed for the splitter. Defaults to 0



**evalml.preprocessing.SMOTETomekCVSplit.get\_n\_splits**

`SMOTETomekCVSplit.get_n_splits()`

Returns the number of splits of this object.

**evalml.preprocessing.SMOTETomekCVSplit.split**

`SMOTETomekCVSplit.split(X, y)`

Splits and returns the sampled training data using the data sampler provided.

**Parameters**

- **X** (`ww.DataTable`) – DataTable of points to split
- **y** (`ww.DataTable`) – DataColumn of points to split

**Returns** A tuple containing the resulting  $((X_{\text{train}}, y_{\text{train}}), (X_{\text{test}}, y_{\text{test}}))$  post-transformation.

**Return type** tuple((pd.DataFrame, pd.Series), (pd.DataFrame, pd.Series))

**evalml.preprocessing.SMOTETomekCVSplit.transform\_sample**

`SMOTETomekCVSplit.transform_sample(X, y)`

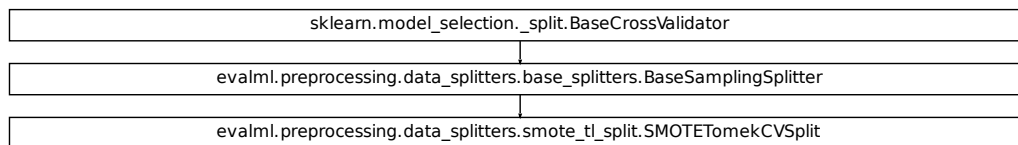
Transforms the input data with the balancing strategy.

**Parameters**

- **X** (`ww.DataTable`) – DataTable of points to split
- **y** (`ww.DataTable`) – DataColumn of points to split

**Returns** A tuple containing the resulting X and y post-transformation.

**Return type** tuple(pd.DataFrame, pd.Series)

**Class Inheritance**

**evalml.preprocessing.SMOTETomekTVSplit**

**class** evalml.preprocessing.SMOTETomekTVSplit (*sampling\_strategy='auto', test\_size=None, n\_jobs=- 1, random\_seed=0*)

Splits the data into training and validation sets and uses SMOTE + Tomek links balance the training data. Keeps the validation data the same. Works only on continuous, numeric data.

**Methods**

<code>__init__</code>	Create a TV or CV data splitter instance
<code>get_n_splits</code>	Returns the number of splits of this object.
<code>split</code>	Splits and returns the sampled training data using the data sampler provided.
<code>transform_sample</code>	Transforms the input data with the balancing strategy.

**evalml.preprocessing.SMOTETomekTVSplit.\_\_init\_\_**

SMOTETomekTVSplit.**\_\_init\_\_** (*sampling\_strategy='auto', test\_size=None, n\_jobs=- 1, random\_seed=0*)

Create a TV or CV data splitter instance

**Parameters**

- **sampler** (*sampler instance*) – The sampler instance to use for resampling the training data. Must have a *fit\_resample* method. Defaults to None, which is equivalent to regular TV split.
- **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.
- **n\_splits** (*int*) – How many CV folds to use. Defaults to 3.
- **shuffle** (*bool*) – Whether or not to shuffle the data. Defaults to True.
- **split\_type** (*str*) – Whether to use TV or CV split. Defaults to TV.
- **random\_seed** (*int*) – Random seed for the splitter. Defaults to 0

**evalml.preprocessing.SMOTETomekTVSplit.get\_n\_splits**

`SMOTETomekTVSplit.get_n_splits()`

Returns the number of splits of this object.

**evalml.preprocessing.SMOTETomekTVSplit.split**

`SMOTETomekTVSplit.split(X, y)`

Splits and returns the sampled training data using the data sampler provided.

**Parameters**

- **X** (`ww.DataTable`) – DataTable of points to split
- **y** (`ww.DataTable`) – DataColumn of points to split

**Returns** A tuple containing the resulting  $((X_{\text{train}}, y_{\text{train}}), (X_{\text{test}}, y_{\text{test}}))$  post-transformation.

**Return type** tuple((pd.DataFrame, pd.Series), (pd.DataFrame, pd.Series))

**evalml.preprocessing.SMOTETomekTVSplit.transform\_sample**

`SMOTETomekTVSplit.transform_sample(X, y)`

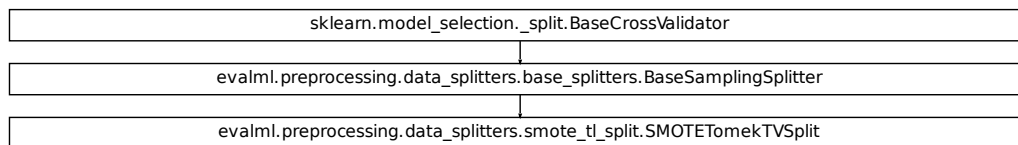
Transforms the input data with the balancing strategy.

**Parameters**

- **X** (`ww.DataTable`) – DataTable of points to split
- **y** (`ww.DataTable`) – DataColumn of points to split

**Returns** A tuple containing the resulting X and y post-transformation.

**Return type** tuple(pd.DataFrame, pd.Series)

**Class Inheritance**

**evalml.preprocessing.RandomUnderSamplerCVSplit**

```
class evalml.preprocessing.RandomUnderSamplerCVSplit (sampling_strategy='auto',  
                                                    replacement=False,  
                                                    n_splits=3,      shuffle=True,  
                                                    random_seed=0)
```

Splits the training data into KFold cross validation sets and uses RandomUnderSampler to balance the training data. Keeps the validation data the same. Works only on continuous, numeric data.

**Methods**

<code>__init__</code>	Create a TV or CV data splitter instance
<code>get_n_splits</code>	Returns the number of splits of this object.
<code>split</code>	Splits and returns the sampled training data using the data sampler provided.
<code>transform_sample</code>	Transforms the input data with the balancing strategy.

**evalml.preprocessing.RandomUnderSamplerCVSplit.\_\_init\_\_**

```
RandomUnderSamplerCVSplit.__init__ (sampling_strategy='auto',      replacement=False,  
                                     n_splits=3, shuffle=True, random_seed=0)
```

Create a TV or CV data splitter instance

**Parameters**

- **sampler** (*sampler instance*) – The sampler instance to use for resampling the training data. Must have a *fit\_resample* method. Defaults to None, which is equivalent to regular TV split.
- **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.
- **n\_splits** (*int*) – How many CV folds to use. Defaults to 3.
- **shuffle** (*bool*) – Whether or not to shuffle the data. Defaults to True.
- **split\_type** (*str*) – Whether to use TV or CV split. Defaults to TV.
- **random\_seed** (*int*) – Random seed for the splitter. Defaults to 0

**evalml.preprocessing.RandomUnderSamplerCVSplit.get\_n\_splits**

```
RandomUnderSamplerCVSplit.get_n_splits ()
```

Returns the number of splits of this object.

**evalml.preprocessing.RandomUnderSamplerCVSplit.split**

`RandomUnderSamplerCVSplit.split(X, y)`

Splits and returns the sampled training data using the data sampler provided.

**Parameters**

- **X** (`ww.DataTable`) – DataTable of points to split
- **y** (`ww.DataTable`) – DataColumn of points to split

**Returns** A tuple containing the resulting `((X_train, y_train), (X_test, y_test))` post-transformation.

**Return type** `tuple((pd.DataFrame, pd.Series), (pd.DataFrame, pd.Series))`

**evalml.preprocessing.RandomUnderSamplerCVSplit.transform\_sample**

`RandomUnderSamplerCVSplit.transform_sample(X, y)`

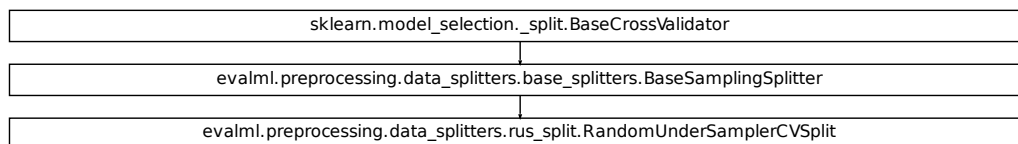
Transforms the input data with the balancing strategy.

**Parameters**

- **X** (`ww.DataTable`) – DataTable of points to split
- **y** (`ww.DataTable`) – DataColumn of points to split

**Returns** A tuple containing the resulting X and y post-transformation.

**Return type** `tuple(pd.DataFrame, pd.Series)`

**Class Inheritance****evalml.preprocessing.RandomUnderSamplerTVSplit**

**class** `evalml.preprocessing.RandomUnderSamplerTVSplit` (`sampling_strategy='auto'`,  
`test_size=None`, `replacement=False`, `random_seed=0`)

Splits the data into training and validation sets and uses `RandomUnderSampler` to balance the training data. Keeps the validation data the same. Works only on continuous, numeric data.

## Methods

<code>__init__</code>	Create a TV or CV data splitter instance
<code>get_n_splits</code>	Returns the number of splits of this object.
<code>split</code>	Splits and returns the sampled training data using the data sampler provided.
<code>transform_sample</code>	Transforms the input data with the balancing strategy.

### `evalml.preprocessing.RandomUnderSamplerTVSplit.__init__`

`RandomUnderSamplerTVSplit.__init__(sampling_strategy='auto', test_size=None, replacement=False, random_seed=0)`

Create a TV or CV data splitter instance

#### Parameters

- **sampler** (*sampler instance*) – The sampler instance to use for resampling the training data. Must have a *fit\_resample* method. Defaults to None, which is equivalent to regular TV split.
- **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.
- **n\_splits** (*int*) – How many CV folds to use. Defaults to 3.
- **shuffle** (*bool*) – Whether or not to shuffle the data. Defaults to True.
- **split\_type** (*str*) – Whether to use TV or CV split. Defaults to TV.
- **random\_seed** (*int*) – Random seed for the splitter. Defaults to 0

### `evalml.preprocessing.RandomUnderSamplerTVSplit.get_n_splits`

`RandomUnderSamplerTVSplit.get_n_splits()`

Returns the number of splits of this object.

### `evalml.preprocessing.RandomUnderSamplerTVSplit.split`

`RandomUnderSamplerTVSplit.split(X, y)`

Splits and returns the sampled training data using the data sampler provided.

#### Parameters

- **X** (*ww.DataTable*) – DataTable of points to split
- **y** (*ww.DataTable*) – DataColumn of points to split

**Returns** A tuple containing the resulting ((X\_train, y\_train), (X\_test, y\_test)) post-transformation.

**Return type** tuple((pd.DataFrame, pd.Series), (pd.DataFrame, pd.Series))

**evalml.preprocessing.RandomUnderSamplerTVSplit.transform\_sample**

`RandomUnderSamplerTVSplit.transform_sample(X, y)`

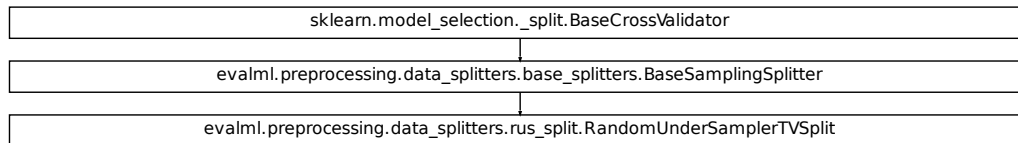
Transforms the input data with the balancing strategy.

**Parameters**

- **X** (`ww.DataTable`) – DataTable of points to split
- **y** (`ww.DataTable`) – DataColumn of points to split

**Returns** A tuple containing the resulting X and y post-transformation.

**Return type** tuple(pd.DataFrame, pd.Series)

**Class Inheritance****evalml.preprocessing.SMOTENCCVSplit**

**class** evalml.preprocessing.**SMOTENCCVSplit** (*categorical\_features=None, sampling\_strategy='auto', n\_splits=3, shuffle=True, n\_jobs=-1, random\_seed=0*)

Splits the data into KFold cross validation sets and uses SMOTENC to balance the training data. Keeps the validation data the same. Works on numeric and categorical data, but categorical data must be numerical

**Methods**

<code>__init__</code>	Create a TV or CV data splitter instance
<code>get_n_splits</code>	Returns the number of splits of this object.
<code>split</code>	Splits and returns the sampled training data using the data sampler provided.
<code>transform_sample</code>	Transforms the input data with the balancing strategy.

### `evalml.preprocessing.SMOTENCCVSplit.__init__`

`SMOTENCCVSplit.__init__(categorical_features=None, sampling_strategy='auto', n_splits=3, shuffle=True, n_jobs=-1, random_seed=0)`

Create a TV or CV data splitter instance

#### Parameters

- **sampler** (*sampler instance*) – The sampler instance to use for resampling the training data. Must have a *fit\_resample* method. Defaults to None, which is equivalent to regular TV split.
- **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.
- **n\_splits** (*int*) – How many CV folds to use. Defaults to 3.
- **shuffle** (*bool*) – Whether or not to shuffle the data. Defaults to True.
- **split\_type** (*str*) – Whether to use TV or CV split. Defaults to TV.
- **random\_seed** (*int*) – Random seed for the splitter. Defaults to 0

### `evalml.preprocessing.SMOTENCCVSplit.get_n_splits`

`SMOTENCCVSplit.get_n_splits()`

Returns the number of splits of this object.

### `evalml.preprocessing.SMOTENCCVSplit.split`

`SMOTENCCVSplit.split(X, y)`

Splits and returns the sampled training data using the data sampler provided.

#### Parameters

- **X** (*ww.DataTable*) – DataTable of points to split
- **y** (*ww.DataTable*) – DataColumn of points to split

**Returns** A tuple containing the resulting ((X\_train, y\_train), (X\_test, y\_test)) post-transformation.

**Return type** tuple((pd.DataFrame, pd.Series), (pd.DataFrame, pd.Series))

### `evalml.preprocessing.SMOTENCCVSplit.transform_sample`

`SMOTENCCVSplit.transform_sample(X, y)`

Transforms the input data with the balancing strategy.

#### Parameters

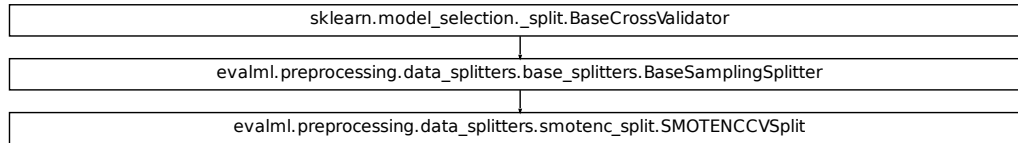
- **X** (*ww.DataTable*) – DataTable of points to split
- **y** (*ww.DataTable*) – DataColumn of points to split

**Returns** A tuple containing the resulting X and y post-transformation.

**Return type** tuple(pd.DataFrame, pd.Series)



## Class Inheritance



### evalml.preprocessing.SMOTENCTVSplit

**class** evalml.preprocessing.**SMOTENCTVSplit** (*categorical\_features=None*, *sampling\_strategy='auto'*, *test\_size=None*, *n\_jobs=-1*, *random\_seed=0*)

Splits the data into training and validation sets and uses SMOTENC to balance the training data. Keeps the validation data the same. Works on numeric and categorical data, but categorical data must be numerical

#### Methods

<code>__init__</code>	Create a TV or CV data splitter instance
<code>get_n_splits</code>	Returns the number of splits of this object.
<code>split</code>	Splits and returns the sampled training data using the data sampler provided.
<code>transform_sample</code>	Transforms the input data with the balancing strategy.

### evalml.preprocessing.SMOTENCTVSplit.\_\_init\_\_

SMOTENCTVSplit.**\_\_init\_\_** (*categorical\_features=None*, *sampling\_strategy='auto'*, *test\_size=None*, *n\_jobs=-1*, *random\_seed=0*)

Create a TV or CV data splitter instance

#### Parameters

- **sampler** (*sampler instance*) – The sampler instance to use for resampling the training data. Must have a *fit\_resample* method. Defaults to None, which is equivalent to regular TV split.
- **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.
- **n\_splits** (*int*) – How many CV folds to use. Defaults to 3.
- **shuffle** (*bool*) – Whether or not to shuffle the data. Defaults to True.
- **split\_type** (*str*) – Whether to use TV or CV split. Defaults to TV.
- **random\_seed** (*int*) – Random seed for the splitter. Defaults to 0

### evalml.preprocessing.SMOTENCTVSplit.get\_n\_splits

`SMOTENCTVSplit.get_n_splits()`  
Returns the number of splits of this object.

### evalml.preprocessing.SMOTENCTVSplit.split

`SMOTENCTVSplit.split(X, y)`  
Splits and returns the sampled training data using the data sampler provided.

#### Parameters

- **X** (*ww.DataTable*) – DataTable of points to split
- **y** (*ww.DataTable*) – DataColumn of points to split

**Returns** A tuple containing the resulting ((X\_train, y\_train), (X\_test, y\_test)) post-transformation.

**Return type** tuple((pd.DataFrame, pd.Series), (pd.DataFrame, pd.Series))

### evalml.preprocessing.SMOTENCTVSplit.transform\_sample

`SMOTENCTVSplit.transform_sample(X, y)`  
Transforms the input data with the balancing strategy.

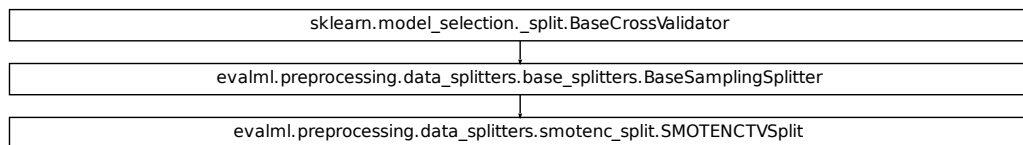
#### Parameters

- **X** (*ww.DataTable*) – DataTable of points to split
- **y** (*ww.DataTable*) – DataColumn of points to split

**Returns** A tuple containing the resulting X and y post-transformation.

**Return type** tuple(pd.DataFrame, pd.Series)

## Class Inheritance



## 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>IllFormattedClassNameError</i>	Exception to raise when a class name does not comply with EvalML standards
<i>MissingComponentError</i>	An exception raised when a component is not found in <code>all_components()</code>
<i>ComponentNotYetFittedError</i>	An exception to be raised when <code>predict/predict_proba/transform</code> is called on a component without fitting first.
<i>PipelineNotYetFittedError</i>	An exception to be raised when <code>predict/predict_proba/transform</code> 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

### 5.3.1 evalml.exceptions.MethodPropertyNotFoundError

**class** evalml.exceptions.**MethodPropertyNotFoundError**

Exception to raise when a class is does not have an expected method or property.

#### Class Inheritance

```
evalml.exceptions.exceptions.MethodPropertyNotFoundError
```

### 5.3.2 evalml.exceptions.PipelineNotFoundError

**class** evalml.exceptions.PipelineNotFoundError

An exception raised when a particular pipeline is not found in automl search results

#### Class Inheritance

evalml.exceptions.exceptions.PipelineNotFoundError

### 5.3.3 evalml.exceptions.ObjectiveNotFoundError

**class** evalml.exceptions.ObjectiveNotFoundError

Exception to raise when specified objective does not exist.

#### Class Inheritance

evalml.exceptions.exceptions.ObjectiveNotFoundError

### 5.3.4 evalml.exceptions.IllFormattedClassNameError

**class** evalml.exceptions.IllFormattedClassNameError

Exception to raise when a class name does not comply with EvalML standards

#### Class Inheritance

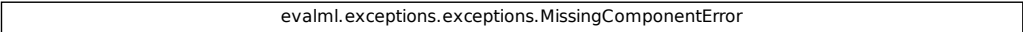
evalml.exceptions.exceptions.IllFormattedClassNameError

### 5.3.5 evalml.exceptions.MissingComponentError

**class** evalml.exceptions.MissingComponentError

An exception raised when a component is not found in all\_components()

#### Class Inheritance



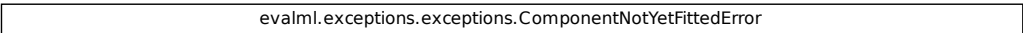
```
evalml.exceptions.exceptions.MissingComponentError
```

### 5.3.6 evalml.exceptions.ComponentNotYetFittedError

**class** evalml.exceptions.ComponentNotYetFittedError

An exception to be raised when predict/predict\_proba/transform is called on a component without fitting first.

#### Class Inheritance



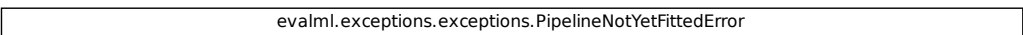
```
evalml.exceptions.exceptions.ComponentNotYetFittedError
```

### 5.3.7 evalml.exceptions.PipelineNotYetFittedError

**class** evalml.exceptions.PipelineNotYetFittedError

An exception to be raised when predict/predict\_proba/transform is called on a pipeline without fitting first.

#### Class Inheritance



```
evalml.exceptions.exceptions.PipelineNotYetFittedError
```

### 5.3.8 evalml.exceptions.AutoMLSearchException

**class** evalml.exceptions.AutoMLSearchException

Exception raised when all pipelines in an automl batch return a score of NaN for the primary objective.

#### Class Inheritance

evalml.exceptions.exceptions.AutoMLSearchException

### 5.3.9 evalml.exceptions.EnsembleMissingPipelinesError

**class** evalml.exceptions.EnsembleMissingPipelinesError

An exception raised when an ensemble is missing *estimators* (list) as a parameter.

#### Class Inheritance

evalml.exceptions.exceptions.EnsembleMissingPipelinesError

### 5.3.10 evalml.exceptions.PipelineScoreError

**class** 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.

## Class Inheritance

evalml.exceptions.exceptions.PipelineScoreError

### 5.3.11 evalml.exceptions.DataCheckInitError

**class** evalml.exceptions.DataCheckInitError

Exception raised when a data check can't initialize with the parameters given.

## Class Inheritance

evalml.exceptions.exceptions.DataCheckInitError

### 5.3.12 evalml.exceptions.NullsInColumnWarning

**class** evalml.exceptions.NullsInColumnWarning

Warning thrown when there are null values in the column of interest

## Class Inheritance

evalml.exceptions.exceptions.NullsInColumnWarning

## 5.4 AutoML

### 5.4.1 AutoML Search Classes

<i>AutoMLSearch</i>	Automated Pipeline search.
---------------------	----------------------------

#### evalml.autoaml.AutoMLSearch

**class** evalml.autoaml.**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\_pipelines=None, allowed\_model\_families=None, start\_iteration\_callback=None, add\_result\_callback=None, error\_callback=None, additional\_objectives=None, 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, \_ensembling\_split\_size=0.2, \_pipelines\_per\_batch=5, engine=None*)

Automated Pipeline search.

#### Methods

<i>__init__</i>	Automated pipeline search
<i>add_to_rankings</i>	Fits and evaluates a given pipeline then adds the results to the automl rankings with the requirement that automl search has been run.
<i>describe_pipeline</i>	Describe a pipeline
<i>get_pipeline</i>	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.
<i>load</i>	Loads AutoML object at file path
<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.



**evalml.automl.AutoMLSearch.\_\_init\_\_**

```
AutoMLSearch.__init__(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_pipelines=None, allowed_model_families=None, start_iteration_callback=None, add_result_callback=None, error_callback=None, additional_objectives=None, random_seed=0, n_jobs=-1, tuner_class=None, optimize_thresholds=True, ensemble=False, max_batches=None, problem_configuration=None, train_best_pipeline=True, pipeline_parameters=None, _ensembling_split_size=0.2, _pipelines_per_batch=5, engine=None)
```

Automated pipeline search

**Parameters**

- **X\_train** (*pd.DataFrame*, *ww.DataTable*) – The input training data of shape [n\_samples, n\_features]. Required.
- **y\_train** (*pd.Series*, *ww.DataColumn*) – 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\_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. Setting this field will cause allowed\_model\_families to be ignored.
- **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 class, the pipeline parameters, 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.
- **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 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.
- **\_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.

**evalml automl.AutoMLSearch.add\_to\_rankings**`AutoMLSearch.add_to_rankings(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.

**evalml automl.AutoMLSearch.describe\_pipeline**`AutoMLSearch.describe_pipeline(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.

**evalml automl.AutoMLSearch.get\_pipeline**`AutoMLSearch.get_pipeline(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`

**evalml automl.AutoMLSearch.load**`static AutoMLSearch.load(file_path)`

Loads AutoML object at file path

**Parameters** `file_path` (`str`) – location to find file to load

**Returns** AutoSearchBase object

**evalml automl.AutoMLSearch.save**`AutoMLSearch.save(file_path, pickle_protocol=5)`

Saves AutoML object at file path

**Parameters**

- `file_path` (`str`) – location to save file
- `pickle_protocol` (`int`) – the pickle data stream format.

**Returns** None

### evalml automl.AutoMLSearch.score\_pipelines

`AutoMLSearch.score_pipelines(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** (`ww.DataTable`, `pd.DataFrame`) – Holdout features.
- **y\_holdout** (`ww.DataTable`, `pd.DataFrame`) – 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]]

### evalml automl.AutoMLSearch.search

`AutoMLSearch.search(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.

### evalml automl.AutoMLSearch.train\_pipelines

`AutoMLSearch.train_pipelines(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`]

### Attributes

<code>best_pipeline</code>	Returns a trained instance of the best pipeline and parameters found during automl search.
<code>full_rankings</code>	Returns a <code>pandas.DataFrame</code> with scoring results from all pipelines searched
<code>rankings</code>	Returns a <code>pandas.DataFrame</code> 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 <i>automl_search</i> .

### Class Inheritance

```
evalml.automl.automl_search.AutoMLSearch
```

## 5.4.2 AutoML Utils

<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.

### evalml.automl.get\_default\_primary\_search\_objective

`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

```
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** (*ww.DataTable*, *pd.DataFrame*) – The input training data of shape [n\_samples, n\_features].
- **y** (*ww.DataColumn*, *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 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`

## 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.

### evalml.automl.automl\_algorithm.AutoMLAlgorithm

```
class evalml.automl.automl_algorithm.AutoMLAlgorithm(allowed_pipelines=None,  
                                                    max_iterations=None,  
                                                    tuner_class=None,      ran-  
                                                    dom_seed=0)
```

Base class for the automl algorithms which power evalml.

## Methods

<code>__init__</code>	This class represents an automated machine learning (AutoML) algorithm.
<code>add_result</code>	Register results from evaluating a pipeline
<code>next_batch</code>	Get the next batch of pipelines to evaluate

### `evalml.automl.automl_algorithm.AutoMLAlgorithm.__init__`

`AutoMLAlgorithm.__init__(allowed_pipelines=None, max_iterations=None, tuner_class=None, random_seed=0)`

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.
- **`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.

### `evalml.automl.automl_algorithm.AutoMLAlgorithm.add_result`

`AutoMLAlgorithm.add_result(score_to_minimize, pipeline, trained_pipeline_results)`

Register results from evaluating a pipeline

#### Parameters

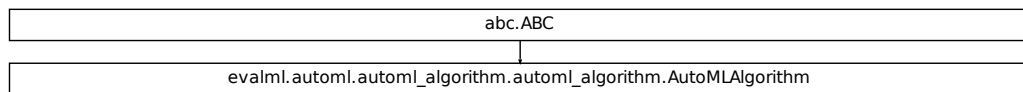
- **`score_to_minimize`** (*float*) – The score obtained by this pipeline on the primary objective, converted so that lower values indicate better pipelines.
- **`pipeline`** (*PipelineBase*) – The trained pipeline object which was used to compute the score.
- **`trained_pipeline_results`** (*dict*) – Results from training a pipeline.

**evalml.automl.automl\_algorithm.AutoMLAlgorithm.next\_batch****abstract** AutoMLAlgorithm.next\_batch()

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*)**Attributes**

batch_number	Returns the number of batches which have been recommended so far.
pipeline_number	Returns the number of pipelines which have been recommended so far.

**Class Inheritance****evalml.automl.automl\_algorithm.IterativeAlgorithm**

```
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,      num-  
                                                         ber_features=None,  
                                                         ensembling=False,  
                                                         pipeline_params=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.



## Methods

<code>__init__</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.
<code>add_result</code>	Register results from evaluating a pipeline
<code>next_batch</code>	Get the next batch of pipelines to evaluate

### `evalml.automl.automl_algorithm.IterativeAlgorithm.__init__`

`IterativeAlgorithm.__init__(allowed_pipelines=None, max_iterations=None, tuner_class=None, random_seed=0, pipelines_per_batch=5, n_jobs=-1, number_features=None, ensembling=False, pipeline_params=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 subclasses 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.
- **`n_jobs`** (*int or None*) – Non-negative integer describing level of parallelism used for pipelines.
- **`number_features`** (*int*) – The number of columns in the input features.
- **`ensembling`** (*boolean*) – If True, runs ensembling in a separate batch after every allowed pipeline class has been iterated over. Defaults to False.
- **`pipeline_params`** (*dict or None*) – Pipeline-level parameters that should be passed to the proposed pipelines.

### `evalml.automl.automl_algorithm.IterativeAlgorithm.add_result`

`IterativeAlgorithm.add_result(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.

### evalml.automl.automl\_algorithm.IterativeAlgorithm.next\_batch

IterativeAlgorithm.**next\_batch**()

Get the next batch of pipelines to evaluate

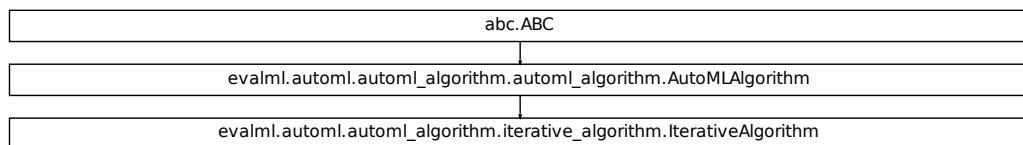
**Returns** a list of instances of PipelineBase subclasses, ready to be trained and evaluated.

**Return type** list(*PipelineBase*)

### Attributes

batch_number	Returns the number of batches which have been recommended so far.
pipeline_number	Returns the number of pipelines which have been recommended so far.

### Class Inheritance



## 5.4.4 AutoML Callbacks

<i>silent_error_callback</i>	No-op.
<i>log_error_callback</i>	Logs the exception thrown as an error.
<i>raise_error_callback</i>	Raises the exception thrown by the AutoMLSearch object.

**evalml.automl.callbacks.silent\_error\_callback**

`evalml.automl.callbacks.silent_error_callback(exception, traceback, automl, **kwargs)`  
 No-op.

**evalml.automl.callbacks.log\_error\_callback**

`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.raise\_error\_callback**

`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.

## 5.5 Pipelines

### 5.5.1 Pipeline Base Classes

<i>PipelineBase</i>	Base class for all pipelines.
<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>	
<i>TimeSeriesMulticlassClassificationPipeline</i>	
<i>TimeSeriesRegressionPipeline</i>	Pipeline base class for time series regression problems.

**evalml.pipelines.PipelineBase**

**class** `evalml.pipelines.PipelineBase(parameters, random_seed=0)`

Base class for all pipelines.

**name** = 'Pipeline Base'

**custom\_name** = None

**problem\_type** = None

### Instance attributes

<code>feature_importance</code>	Return importance associated with each feature.
<code>parameters</code>	Returns parameter dictionary for this pipeline

### Methods:

<code>__init__</code>	Machine learning pipeline made out of transformers and a estimator.
<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>describe</code>	Outputs pipeline details including component parameters
<code>fit</code>	Build a model
<code>get_component</code>	Returns component by name
<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>load</code>	Loads pipeline at file path
<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

### `evalml.pipelines.PipelineBase.__init__`

`PipelineBase.__init__(parameters, random_seed=0)`

Machine learning pipeline made out of transformers and a estimator.

#### Required Class Variables:

**`component_graph` (list or dict): List of components in order. Accepts strings or `ComponentBase` subclasses in the**

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

- **`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.
- **`random_seed` (*int*)** – Seed for the random number generator. Defaults to 0.

**evalml.pipelines.PipelineBase.can\_tune\_threshold\_with\_objective**

`PipelineBase.can_tune_threshold_with_objective(objective)`

Determine whether the threshold of a binary classification pipeline can be tuned.

**Parameters**

- **pipeline** (`PipelineBase`) – Binary classification pipeline.
- **objective** – Primary AutoMLSearch objective.

**evalml.pipelines.PipelineBase.clone**

`PipelineBase.clone()`

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.

**evalml.pipelines.PipelineBase.compute\_estimator\_features**

`PipelineBase.compute_estimator_features(X, y=None)`

Transforms the data by applying all pre-processing components.

**Parameters** **X** (`ww.DataTable`, `pd.DataFrame`) – Input data to the pipeline to transform.

**Returns** New transformed features.

**Return type** `ww.DataTable`

**evalml.pipelines.PipelineBase.create\_objectives**

**static** `PipelineBase.create_objectives(objectives)`

**evalml.pipelines.PipelineBase.describe**

`PipelineBase.describe(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`

### `evalml.pipelines.PipelineBase.fit`

**abstract** `PipelineBase.fit` (*X*, *y*)

Build a model

**Parameters**

- **X** (*ww.DataTable*, *pd.DataFrame* or *np.ndarray*) – The input training data of shape [*n\_samples*, *n\_features*]
- **y** (*ww.DataColumn*, *pd.Series*, *np.ndarray*) – The target training data of length [*n\_samples*]

**Returns** `self`

### `evalml.pipelines.PipelineBase.get_component`

`PipelineBase.get_component` (*name*)

Returns component by name

**Parameters** **name** (*str*) – Name of component

**Returns** Component to return

**Return type** Component

### `evalml.pipelines.PipelineBase.graph`

`PipelineBase.graph` (*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`

### `evalml.pipelines.PipelineBase.graph_feature_importance`

`PipelineBase.graph_feature_importance` (*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

**evalml.pipelines.PipelineBase.load****static** PipelineBase.load(*file\_path*)

Loads pipeline at file path

**Parameters** *file\_path* (*str*) – location to load file**Returns** PipelineBase object**evalml.pipelines.PipelineBase.predict**PipelineBase.predict(*X*, *objective=None*)

Make predictions using selected features.

**Parameters**

- **X** (*ww.DataTable*, *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** ww.DataColumn**evalml.pipelines.PipelineBase.save**PipelineBase.save(*file\_path*, *pickle\_protocol=5*)

Saves pipeline at file path

**Parameters**

- **file\_path** (*str*) – location to save file
- **pickle\_protocol** (*int*) – the pickle data stream format.

**Returns** None**evalml.pipelines.PipelineBase.score****abstract** PipelineBase.score(*X*, *y*, *objectives*)

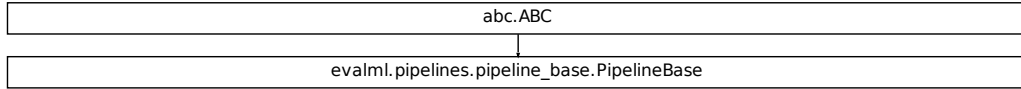
Evaluate model performance on current and additional objectives

**Parameters**

- **X** (*ww.DataTable*, *pd.DataFrame* or *np.ndarray*) – Data of shape [n\_samples, n\_features]
- **y** (*pd.Series*, *ww.DataColumn*, or *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

## Class Inheritance



### evalml.pipelines.ClassificationPipeline

**class** evalml.pipelines.ClassificationPipeline(*parameters*, *random\_seed=0*)

Pipeline subclass for all classification pipelines.

**name** = 'Classification Pipeline'

**custom\_name** = None

**problem\_type** = None

#### Instance attributes

<code>classes_</code>	Gets the class names for the problem.
<code>feature_importance</code>	Return importance associated with each feature.
<code>parameters</code>	Returns parameter dictionary for this pipeline

#### Methods:

<code>__init__</code>	Machine learning classification pipeline made out of transformers and a classifier.
<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>describe</code>	Outputs pipeline details including component parameters
<code>fit</code>	Build a classification model. For string and categorical targets, classes are sorted
<code>get_component</code>	Returns component by name
<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

continues on next page



Table 29 – continued from previous page

<code>load</code>	Loads pipeline at file path
<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

### `evalml.pipelines.ClassificationPipeline.__init__`

`ClassificationPipeline.__init__(parameters, random_seed=0)`

Machine learning classification pipeline made out of transformers and a classifier.

**Required Class Variables:** `component_graph` (list): List of components in order. Accepts strings or `ComponentBase` subclasses in the list

#### Parameters

- **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.
- **random\_state** (*int*) – Deprecated - use `random_seed` instead.
- **random\_seed** (*int*) – Seed for the random number generator. Defaults to 0.

### `evalml.pipelines.ClassificationPipeline.can_tune_threshold_with_objective`

`ClassificationPipeline.can_tune_threshold_with_objective(objective)`

Determine whether the threshold of a binary classification pipeline can be tuned.

#### Parameters

- **pipeline** (`PipelineBase`) – Binary classification pipeline.
- **objective** – Primary `AutoMLSearch` objective.

### `evalml.pipelines.ClassificationPipeline.clone`

`ClassificationPipeline.clone()`

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.

### `evalml.pipelines.ClassificationPipeline.compute_estimator_features`

`ClassificationPipeline.compute_estimator_features(X, y=None)`

Transforms the data by applying all pre-processing components.

**Parameters** **X** (*ww.DataTable*, *pd.DataFrame*) – Input data to the pipeline to transform.

**Returns** New transformed features.

**Return type** *ww.DataTable*

### `evalml.pipelines.ClassificationPipeline.create_objectives`

**static** `ClassificationPipeline.create_objectives` (*objectives*)

### `evalml.pipelines.ClassificationPipeline.describe`

`ClassificationPipeline.describe` (*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

### `evalml.pipelines.ClassificationPipeline.fit`

`ClassificationPipeline.fit` (*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** (*ww.DataTable, pd.DataFrame or np.ndarray*) – The input training data of shape `[n_samples, n_features]`
- **y** (*ww.DataColumn, pd.Series, np.ndarray*) – The target training labels of length `[n_samples]`

**Returns** self

### `evalml.pipelines.ClassificationPipeline.get_component`

`ClassificationPipeline.get_component` (*name*)

Returns component by name

**Parameters** **name** (*str*) – Name of component

**Returns** Component to return

**Return type** Component

### `evalml.pipelines.ClassificationPipeline.graph`

`ClassificationPipeline.graph` (*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

**evalml.pipelines.ClassificationPipeline.graph\_feature\_importance**

`ClassificationPipeline.graph_feature_importance(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

**evalml.pipelines.ClassificationPipeline.load**

**static** `ClassificationPipeline.load(file_path)`

Loads pipeline at file path

**Parameters** `file_path` (*str*) – location to load file

**Returns** `PipelineBase` object

**evalml.pipelines.ClassificationPipeline.predict**

`ClassificationPipeline.predict(X, objective=None)`

Make predictions using selected features.

**Parameters**

- **X** (*ww.DataTable, 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** `ww.DataColumn`

**evalml.pipelines.ClassificationPipeline.predict\_proba**

`ClassificationPipeline.predict_proba(X)`

Make probability estimates for labels.

**Parameters** **X** (*ww.DataTable, pd.DataFrame or np.ndarray*) – Data of shape `[n_samples, n_features]`

**Returns** Probability estimates

**Return type** `ww.DataTable`

### `evalml.pipelines.ClassificationPipeline.save`

`ClassificationPipeline.save` (*file\_path*, *pickle\_protocol=5*)

Saves pipeline at file path

#### Parameters

- **file\_path** (*str*) – location to save file
- **pickle\_protocol** (*int*) – the pickle data stream format.

**Returns** None

### `evalml.pipelines.ClassificationPipeline.score`

`ClassificationPipeline.score` (*X*, *y*, *objectives*)

Evaluate model performance on objectives

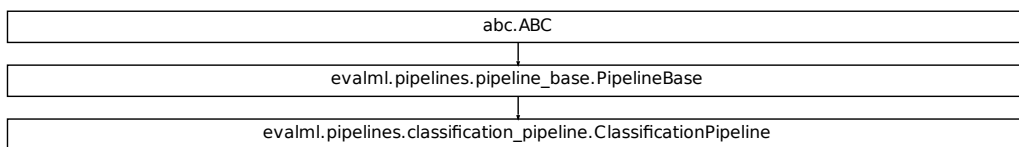
#### Parameters

- **X** (*ww.DataTable*, *pd.DataFrame* or *np.ndarray*) – Data of shape [n\_samples, n\_features]
- **y** (*ww.DataColumn*, *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

## Class Inheritance



**evalml.pipelines.BinaryClassificationPipeline**

**class** evalml.pipelines.**BinaryClassificationPipeline** (*parameters, random\_seed=0*)

Pipeline subclass for all binary classification pipelines.

**name** = 'Binary Classification Pipeline'

**custom\_name** = None

**problem\_type** = 'binary'

**Instance attributes**

<code>classes_</code>	Gets the class names for the problem.
<code>feature_importance</code>	Return importance associated with each feature.
<code>parameters</code>	Returns parameter dictionary for this pipeline
<code>threshold</code>	Threshold used to make a prediction.

**Methods:**

<code>__init__</code>	Machine learning classification pipeline made out of transformers and a classifier.
<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>describe</code>	Outputs pipeline details including component parameters
<code>fit</code>	Build a classification model. For string and categorical targets, classes are sorted
<code>get_component</code>	Returns component by name
<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>load</code>	Loads pipeline at file path
<code>optimize_threshold</code>	Optimize the pipeline threshold given the objective to use.
<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

## `evalml.pipelines.BinaryClassificationPipeline.__init__`

`BinaryClassificationPipeline.__init__(parameters, random_seed=0)`

Machine learning classification pipeline made out of transformers and a classifier.

**Required Class Variables:** `component_graph` (list): List of components in order. Accepts strings or `ComponentBase` subclasses in the list

### Parameters

- **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.
- **random\_state** (*int*) – Deprecated - use `random_seed` instead.
- **random\_seed** (*int*) – Seed for the random number generator. Defaults to 0.

## `evalml.pipelines.BinaryClassificationPipeline.can_tune_threshold_with_objective`

`BinaryClassificationPipeline.can_tune_threshold_with_objective(objective)`

Determine whether the threshold of a binary classification pipeline can be tuned.

### Parameters

- **pipeline** (`PipelineBase`) – Binary classification pipeline.
- **objective** – Primary `AutoMLSearch` objective.

## `evalml.pipelines.BinaryClassificationPipeline.clone`

`BinaryClassificationPipeline.clone()`

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.

## `evalml.pipelines.BinaryClassificationPipeline.compute_estimator_features`

`BinaryClassificationPipeline.compute_estimator_features(X, y=None)`

Transforms the data by applying all pre-processing components.

**Parameters** **X** (*ww.DataTable*, *pd.DataFrame*) – Input data to the pipeline to transform.

**Returns** New transformed features.

**Return type** *ww.DataTable*

**evalml.pipelines.BinaryClassificationPipeline.create\_objectives**

**static** BinaryClassificationPipeline.**create\_objectives** (*objectives*)

**evalml.pipelines.BinaryClassificationPipeline.describe**

BinaryClassificationPipeline.**describe** (*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

**evalml.pipelines.BinaryClassificationPipeline.fit**

BinaryClassificationPipeline.**fit** (*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** (*ww.DataTable, pd.DataFrame or np.ndarray*) – The input training data of shape [n\_samples, n\_features]
- **y** (*ww.DataColumn, pd.Series, np.ndarray*) – The target training labels of length [n\_samples]

**Returns** self

**evalml.pipelines.BinaryClassificationPipeline.get\_component**

BinaryClassificationPipeline.**get\_component** (*name*)

Returns component by name

**Parameters** **name** (*str*) – Name of component

**Returns** Component to return

**Return type** Component

**evalml.pipelines.BinaryClassificationPipeline.graph**

BinaryClassificationPipeline.**graph** (*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

### `evalml.pipelines.BinaryClassificationPipeline.graph_feature_importance`

`BinaryClassificationPipeline.graph_feature_importance(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

### `evalml.pipelines.BinaryClassificationPipeline.load`

**static** `BinaryClassificationPipeline.load(file_path)`  
Loads pipeline at file path

**Parameters** `file_path` (*str*) – location to load file

**Returns** `PipelineBase` object

### `evalml.pipelines.BinaryClassificationPipeline.optimize_threshold`

`BinaryClassificationPipeline.optimize_threshold(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** (*ww.DataTable*) – Input features
- **y** (*ww.DataColumn*) – Input target values
- **y\_pred\_proba** (*ww.DataColumn*) – The predicted probabilities of the target outputted by the pipeline
- **objective** (*ObjectiveBase*) – The objective to threshold with. Must have a tunable threshold.

### `evalml.pipelines.BinaryClassificationPipeline.predict`

`BinaryClassificationPipeline.predict(X, objective=None)`  
Make predictions using selected features.

#### Parameters

- **X** (*ww.DataTable, 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** `ww.DataColumn`



**evalml.pipelines.BinaryClassificationPipeline.predict\_proba**

BinaryClassificationPipeline.**predict\_proba**(*X*)

Make probability estimates for labels. Assumes that the column at index 1 represents the positive label case.

**Parameters** *X* (*ww.DataTable*, *pd.DataFrame* or *np.ndarray*) – Data of shape [n\_samples, n\_features]

**Returns** Probability estimates

**Return type** *ww.DataTable*

**evalml.pipelines.BinaryClassificationPipeline.save**

BinaryClassificationPipeline.**save**(*file\_path*, *pickle\_protocol=5*)

Saves pipeline at file path

**Parameters**

- **file\_path** (*str*) – location to save file
- **pickle\_protocol** (*int*) – the pickle data stream format.

**Returns** None

**evalml.pipelines.BinaryClassificationPipeline.score**

BinaryClassificationPipeline.**score**(*X*, *y*, *objectives*)

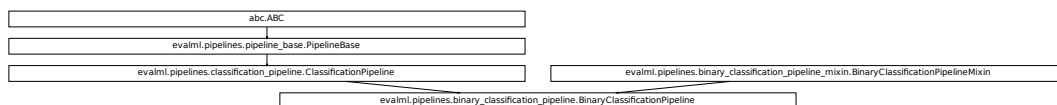
Evaluate model performance on objectives

**Parameters**

- **X** (*ww.DataTable*, *pd.DataFrame* or *np.ndarray*) – Data of shape [n\_samples, n\_features]
- **y** (*ww.DataColumn*, *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

**Class Inheritance**

## evalml.pipelines.MulticlassClassificationPipeline

```
class evalml.pipelines.MulticlassClassificationPipeline(parameters, random_seed=0)
```

Pipeline subclass for all multiclass classification pipelines.

```
name = 'Multiclass Classification Pipeline'
```

```
custom_name = None
```

```
problem_type = 'multiclass'
```

### Instance attributes

<code>classes_</code>	Gets the class names for the problem.
<code>feature_importance</code>	Return importance associated with each feature.
<code>parameters</code>	Returns parameter dictionary for this pipeline

### Methods:

<code>__init__</code>	Machine learning classification pipeline made out of transformers and a classifier.
<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>describe</code>	Outputs pipeline details including component parameters
<code>fit</code>	Build a classification model. For string and categorical targets, classes are sorted
<code>get_component</code>	Returns component by name
<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>load</code>	Loads pipeline at file path
<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

**evalml.pipelines.MulticlassClassificationPipeline.\_\_init\_\_**

`MulticlassClassificationPipeline.__init__(parameters, random_seed=0)`

Machine learning classification pipeline made out of transformers and a classifier.

**Required Class Variables:** `component_graph` (list): List of components in order. Accepts strings or `ComponentBase` subclasses in the list

**Parameters**

- **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.
- **random\_state** (*int*) – Deprecated - use `random_seed` instead.
- **random\_seed** (*int*) – Seed for the random number generator. Defaults to 0.

**evalml.pipelines.MulticlassClassificationPipeline.can\_tune\_threshold\_with\_objective**

`MulticlassClassificationPipeline.can_tune_threshold_with_objective(objective)`

Determine whether the threshold of a binary classification pipeline can be tuned.

**Parameters**

- **pipeline** (`PipelineBase`) – Binary classification pipeline.
- **objective** – Primary `AutoMLSearch` objective.

**evalml.pipelines.MulticlassClassificationPipeline.clone**

`MulticlassClassificationPipeline.clone()`

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.

**evalml.pipelines.MulticlassClassificationPipeline.compute\_estimator\_features**

`MulticlassClassificationPipeline.compute_estimator_features(X, y=None)`

Transforms the data by applying all pre-processing components.

**Parameters** **X** (`ww.DataTable`, `pd.DataFrame`) – Input data to the pipeline to transform.

**Returns** New transformed features.

**Return type** `ww.DataTable`

**evalml.pipelines.MulticlassClassificationPipeline.create\_objectives**

**static** MulticlassClassificationPipeline.**create\_objectives** (*objectives*)

**evalml.pipelines.MulticlassClassificationPipeline.describe**

MulticlassClassificationPipeline.**describe** (*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

**evalml.pipelines.MulticlassClassificationPipeline.fit**

MulticlassClassificationPipeline.**fit** (*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** (*ww.DataTable, pd.DataFrame or np.ndarray*) – The input training data of shape [n\_samples, n\_features]
- **y** (*ww.DataColumn, pd.Series, np.ndarray*) – The target training labels of length [n\_samples]

**Returns** self

**evalml.pipelines.MulticlassClassificationPipeline.get\_component**

MulticlassClassificationPipeline.**get\_component** (*name*)

Returns component by name

**Parameters** **name** (*str*) – Name of component

**Returns** Component to return

**Return type** Component

**evalml.pipelines.MulticlassClassificationPipeline.graph**

MulticlassClassificationPipeline.**graph** (*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

**evalml.pipelines.MulticlassClassificationPipeline.graph\_feature\_importance**

`MulticlassClassificationPipeline.graph_feature_importance` (*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

**evalml.pipelines.MulticlassClassificationPipeline.load**

**static** `MulticlassClassificationPipeline.load` (*file\_path*)

Loads pipeline at file path

**Parameters** `file_path` (*str*) – location to load file

**Returns** `PipelineBase` object

**evalml.pipelines.MulticlassClassificationPipeline.predict**

`MulticlassClassificationPipeline.predict` (*X, objective=None*)

Make predictions using selected features.

**Parameters**

- **X** (*ww.DataTable, 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** `ww.DataColumn`

**evalml.pipelines.MulticlassClassificationPipeline.predict\_proba**

`MulticlassClassificationPipeline.predict_proba` (*X*)

Make probability estimates for labels.

**Parameters** **X** (*ww.DataTable, pd.DataFrame or np.ndarray*) – Data of shape `[n_samples, n_features]`

**Returns** Probability estimates

**Return type** `ww.DataTable`

### `evalml.pipelines.MulticlassClassificationPipeline.save`

`MulticlassClassificationPipeline.save` (*file\_path*, *pickle\_protocol=5*)

Saves pipeline at file path

#### Parameters

- **file\_path** (*str*) – location to save file
- **pickle\_protocol** (*int*) – the pickle data stream format.

**Returns** None

### `evalml.pipelines.MulticlassClassificationPipeline.score`

`MulticlassClassificationPipeline.score` (*X*, *y*, *objectives*)

Evaluate model performance on objectives

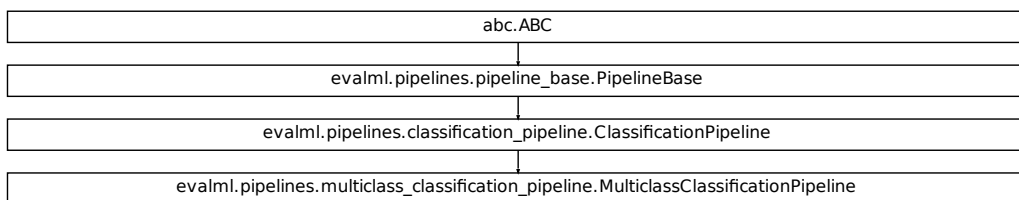
#### Parameters

- **X** (*ww.DataTable*, *pd.DataFrame* or *np.ndarray*) – Data of shape [n\_samples, n\_features]
- **y** (*ww.DataColumn*, *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

## Class Inheritance



**evalml.pipelines.RegressionPipeline**

**class** evalml.pipelines.RegressionPipeline (*parameters*, *random\_seed=0*)

Pipeline subclass for all regression pipelines.

**name** = 'Regression Pipeline'

**custom\_name** = None

**problem\_type** = 'regression'

**Instance attributes**

<code>feature_importance</code>	Return importance associated with each feature.
<code>parameters</code>	Returns parameter dictionary for this pipeline

**Methods:**

<code>__init__</code>	Machine learning pipeline made out of transformers and a estimator.
<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>describe</code>	Outputs pipeline details including component parameters
<code>fit</code>	Build a regression model.
<code>get_component</code>	Returns component by name
<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>load</code>	Loads pipeline at file path
<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

### `evalml.pipelines.RegressionPipeline.__init__`

`RegressionPipeline.__init__(parameters, random_seed=0)`  
Machine learning pipeline made out of transformers and a estimator.

#### Required Class Variables:

**component\_graph (list or dict): List of components in order. Accepts strings or ComponentBase subclasses in the**

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

- **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.
- **random\_seed** (*int*) – Seed for the random number generator. Defaults to 0.

### `evalml.pipelines.RegressionPipeline.can_tune_threshold_with_objective`

`RegressionPipeline.can_tune_threshold_with_objective(objective)`  
Determine whether the threshold of a binary classification pipeline can be tuned.

#### Parameters

- **pipeline** (`PipelineBase`) – Binary classification pipeline.
- **objective** – Primary AutoMLSearch objective.

### `evalml.pipelines.RegressionPipeline.clone`

`RegressionPipeline.clone()`  
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.

### `evalml.pipelines.RegressionPipeline.compute_estimator_features`

`RegressionPipeline.compute_estimator_features(X, y=None)`  
Transforms the data by applying all pre-processing components.

**Parameters** **X** (`ww.DataTable`, `pd.DataFrame`) – Input data to the pipeline to transform.

**Returns** New transformed features.

**Return type** `ww.DataTable`



**evalml.pipelines.RegressionPipeline.create\_objectives**

**static** RegressionPipeline.**create\_objectives** (*objectives*)

**evalml.pipelines.RegressionPipeline.describe**

RegressionPipeline.**describe** (*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

**evalml.pipelines.RegressionPipeline.fit**

RegressionPipeline.**fit** (*X, y*)

Build a regression model.

**Parameters**

- **X** (*ww.DataTable, pd.DataFrame or np.ndarray*) – The input training data of shape [n\_samples, n\_features]
- **y** (*ww.DataColumn, pd.Series, np.ndarray*) – The target training data of length [n\_samples]

**Returns** self

**evalml.pipelines.RegressionPipeline.get\_component**

RegressionPipeline.**get\_component** (*name*)

Returns component by name

**Parameters** **name** (*str*) – Name of component

**Returns** Component to return

**Return type** Component

**evalml.pipelines.RegressionPipeline.graph**

RegressionPipeline.**graph** (*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

### `evalml.pipelines.RegressionPipeline.graph_feature_importance`

`RegressionPipeline.graph_feature_importance` (*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

### `evalml.pipelines.RegressionPipeline.load`

**static** `RegressionPipeline.load` (*file\_path*)

Loads pipeline at file path

**Parameters** `file_path` (*str*) – location to load file

**Returns** `PipelineBase` object

### `evalml.pipelines.RegressionPipeline.predict`

`RegressionPipeline.predict` (*X, objective=None*)

Make predictions using selected features.

**Parameters**

- **X** (*ww.DataTable, 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** `ww.DataColumn`

### `evalml.pipelines.RegressionPipeline.save`

`RegressionPipeline.save` (*file\_path, pickle\_protocol=5*)

Saves pipeline at file path

**Parameters**

- **file\_path** (*str*) – location to save file
- **pickle\_protocol** (*int*) – the pickle data stream format.

**Returns** `None`

## evalml.pipelines.RegressionPipeline.score

RegressionPipeline.**score**(*X*, *y*, *objectives*)

Evaluate model performance on current and additional objectives

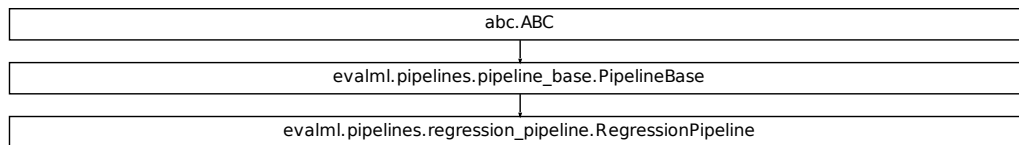
### Parameters

- **X** (*ww.DataTable*, *pd.DataFrame*, or *np.ndarray*) – Data of shape [n\_samples, n\_features]
- **y** (*ww.DataColumn*, *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

## Class Inheritance



## evalml.pipelines.TimeSeriesClassificationPipeline

**class** evalml.pipelines.**TimeSeriesClassificationPipeline**(*parameters*, *random\_seed=0*)

Pipeline base class for time series classification problems.

**name** = 'Time Series Classification Pipeline'

**custom\_name** = None

**problem\_type** = None

### Instance attributes

<code>classes_</code>	Gets the class names for the problem.
<code>feature_importance</code>	Return importance associated with each feature.
<code>parameters</code>	Returns parameter dictionary for this pipeline

**Methods:**

<code>__init__</code>	Machine learning pipeline for time series classification problems made out of transformers and a classifier.
<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>describe</code>	Outputs pipeline details including component parameters
<code>fit</code>	Fit a time series classification pipeline.
<code>get_component</code>	Returns component by name
<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>load</code>	Loads pipeline at file path
<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.

**evalml.pipelines.TimeSeriesClassificationPipeline.\_\_init\_\_**

`TimeSeriesClassificationPipeline.__init__(parameters, random_seed=0)`

Machine learning pipeline for time series classification problems made out of transformers and a classifier.

**Required Class Variables:** `component_graph` (list): List of components in order. Accepts strings or `ComponentBase` subclasses in the list

**Parameters**

- **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 `gap` and `max_delay` must be specified with the "pipeline" key. For example: `Pipeline(parameters={"pipeline": {"max_delay": 4, "gap": 2}})`.
- **random\_seed** (*int*) – Seed for the random number generator. Defaults to 0.

**evalml.pipelines.TimeSeriesClassificationPipeline.can\_tune\_threshold\_with\_objective**

`TimeSeriesClassificationPipeline.can_tune_threshold_with_objective(objective)`

Determine whether the threshold of a binary classification pipeline can be tuned.

**Parameters**

- **pipeline** (`PipelineBase`) – Binary classification pipeline.
- **objective** – Primary AutoMLSearch objective.

**evalml.pipelines.TimeSeriesClassificationPipeline.clone**

`TimeSeriesClassificationPipeline.clone()`

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.

**evalml.pipelines.TimeSeriesClassificationPipeline.compute\_estimator\_features**

`TimeSeriesClassificationPipeline.compute_estimator_features(X, y=None)`

Transforms the data by applying all pre-processing components.

**Parameters** **X** (`ww.DataTable`, `pd.DataFrame`) – Input data to the pipeline to transform.

**Returns** New transformed features.

**Return type** `ww.DataTable`

**evalml.pipelines.TimeSeriesClassificationPipeline.create\_objectives**

**static** `TimeSeriesClassificationPipeline.create_objectives(objectives)`

**evalml.pipelines.TimeSeriesClassificationPipeline.describe**

`TimeSeriesClassificationPipeline.describe(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`

### `evalml.pipelines.TimeSeriesClassificationPipeline.fit`

`TimeSeriesClassificationPipeline.fit(X, y)`

Fit a time series classification pipeline.

#### Parameters

- **X** (*ww.DataTable*, *pd.DataFrame* or *np.ndarray*) – The input training data of shape [n\_samples, n\_features]
- **y** (*ww.DataColumn*, *pd.Series*, *np.ndarray*) – The target training targets of length [n\_samples]

**Returns** self

### `evalml.pipelines.TimeSeriesClassificationPipeline.get_component`

`TimeSeriesClassificationPipeline.get_component(name)`

Returns component by name

**Parameters** **name** (*str*) – Name of component

**Returns** Component to return

**Return type** Component

### `evalml.pipelines.TimeSeriesClassificationPipeline.graph`

`TimeSeriesClassificationPipeline.graph(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

### `evalml.pipelines.TimeSeriesClassificationPipeline.graph_feature_importance`

`TimeSeriesClassificationPipeline.graph_feature_importance(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

**evalml.pipelines.TimeSeriesClassificationPipeline.load**

**static** TimeSeriesClassificationPipeline.**load**(*file\_path*)

Loads pipeline at file path

**Parameters** **file\_path** (*str*) – location to load file

**Returns** PipelineBase object

**evalml.pipelines.TimeSeriesClassificationPipeline.predict**

TimeSeriesClassificationPipeline.**predict**(*X*, *y=None*, *objective=None*)

Make predictions using selected features.

**Parameters**

- **X** (*ww.DataTable*, *pd.DataFrame*, or *np.ndarray*) – Data of shape [n\_samples, n\_features]
- **y** (*ww.DataColumn*, *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** ww.DataColumn

**evalml.pipelines.TimeSeriesClassificationPipeline.predict\_proba**

TimeSeriesClassificationPipeline.**predict\_proba**(*X*, *y=None*)

Make probability estimates for labels.

**Parameters** **X** (*ww.DataTable*, *pd.DataFrame* or *np.ndarray*) – Data of shape [n\_samples, n\_features]

**Returns** Probability estimates

**Return type** ww.DataTable

**evalml.pipelines.TimeSeriesClassificationPipeline.save**

TimeSeriesClassificationPipeline.**save**(*file\_path*, *pickle\_protocol=5*)

Saves pipeline at file path

**Parameters**

- **file\_path** (*str*) – location to save file
- **pickle\_protocol** (*int*) – the pickle data stream format.

**Returns** None

### evalml.pipelines.TimeSeriesClassificationPipeline.score

TimeSeriesClassificationPipeline.**score**(*X*, *y*, *objectives*)

Evaluate model performance on current and additional objectives.

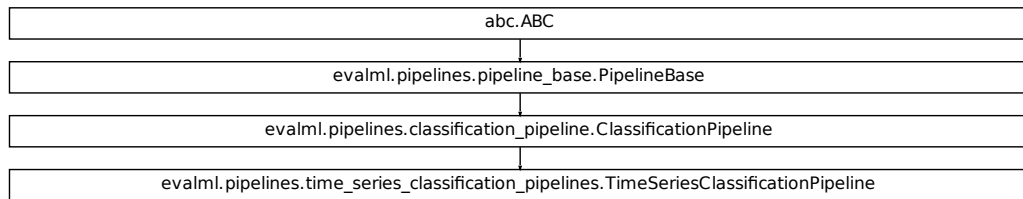
#### Parameters

- **X** (*ww.DataTable*, *pd.DataFrame* or *np.ndarray*) – Data of shape [n\_samples, n\_features]
- **y** (*ww.DataColumn*, *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

### Class Inheritance



### evalml.pipelines.TimeSeriesBinaryClassificationPipeline

```
class evalml.pipelines.TimeSeriesBinaryClassificationPipeline(parameters, random_seed=0)
```

```
name = 'Time Series Binary Classification Pipeline'
```

```
custom_name = None
```

```
problem_type = 'time series binary'
```

#### Instance attributes

classes_	Gets the class names for the problem.
feature_importance	Return importance associated with each feature.
parameters	Returns parameter dictionary for this pipeline
threshold	Threshold used to make a prediction.



**Methods:**

<code>__init__</code>	Machine learning pipeline for time series classification problems made out of transformers and a classifier.
<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>describe</code>	Outputs pipeline details including component parameters
<code>fit</code>	Fit a time series classification pipeline.
<code>get_component</code>	Returns component by name
<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>load</code>	Loads pipeline at file path
<code>optimize_threshold</code>	Optimize the pipeline threshold given the objective to use.
<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.

**evalml.pipelines.TimeSeriesBinaryClassificationPipeline.\_\_init\_\_**

`TimeSeriesBinaryClassificationPipeline.__init__(parameters, random_seed=0)`

Machine learning pipeline for time series classification problems made out of transformers and a classifier.

**Required Class Variables:** `component_graph` (list): List of components in order. Accepts strings or `ComponentBase` subclasses in the list

**Parameters**

- **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 `gap` and `max_delay` must be specified with the "pipeline" key. For example: `Pipeline(parameters={"pipeline": {"max_delay": 4, "gap": 2}})`.
- **random\_seed** (*int*) – Seed for the random number generator. Defaults to 0.

**evalml.pipelines.TimeSeriesBinaryClassificationPipeline.can\_tune\_threshold\_with\_objective**

`TimeSeriesBinaryClassificationPipeline.can_tune_threshold_with_objective(objective)`  
Determine whether the threshold of a binary classification pipeline can be tuned.

**Parameters**

- **pipeline** (`PipelineBase`) – Binary classification pipeline.
- **objective** – Primary AutoMLSearch objective.

**evalml.pipelines.TimeSeriesBinaryClassificationPipeline.clone**

`TimeSeriesBinaryClassificationPipeline.clone()`  
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.

**evalml.pipelines.TimeSeriesBinaryClassificationPipeline.compute\_estimator\_features**

`TimeSeriesBinaryClassificationPipeline.compute_estimator_features(X, y=None)`  
Transforms the data by applying all pre-processing components.

**Parameters** **X** (`ww.DataTable`, `pd.DataFrame`) – Input data to the pipeline to transform.

**Returns** New transformed features.

**Return type** `ww.DataTable`

**evalml.pipelines.TimeSeriesBinaryClassificationPipeline.create\_objectives**

**static** `TimeSeriesBinaryClassificationPipeline.create_objectives(objectives)`

**evalml.pipelines.TimeSeriesBinaryClassificationPipeline.describe**

`TimeSeriesBinaryClassificationPipeline.describe(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`

**evalml.pipelines.TimeSeriesBinaryClassificationPipeline.fit**

`TimeSeriesBinaryClassificationPipeline.fit(X, y)`

Fit a time series classification pipeline.

**Parameters**

- **X** (*ww.DataTable*, *pd.DataFrame* or *np.ndarray*) – The input training data of shape [n\_samples, n\_features]
- **y** (*ww.DataColumn*, *pd.Series*, *np.ndarray*) – The target training targets of length [n\_samples]

**Returns** self

**evalml.pipelines.TimeSeriesBinaryClassificationPipeline.get\_component**

`TimeSeriesBinaryClassificationPipeline.get_component(name)`

Returns component by name

**Parameters** **name** (*str*) – Name of component

**Returns** Component to return

**Return type** Component

**evalml.pipelines.TimeSeriesBinaryClassificationPipeline.graph**

`TimeSeriesBinaryClassificationPipeline.graph(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

**evalml.pipelines.TimeSeriesBinaryClassificationPipeline.graph\_feature\_importance**

`TimeSeriesBinaryClassificationPipeline.graph_feature_importance(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

**evalml.pipelines.TimeSeriesBinaryClassificationPipeline.load**

**static** TimeSeriesBinaryClassificationPipeline.load(*file\_path*)

Loads pipeline at file path

**Parameters** *file\_path* (*str*) – location to load file

**Returns** PipelineBase object

**evalml.pipelines.TimeSeriesBinaryClassificationPipeline.optimize\_threshold**

TimeSeriesBinaryClassificationPipeline.optimize\_threshold(*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** (*ww.DataTable*) – Input features
- **y** (*ww.DataColumn*) – Input target values
- **y\_pred\_proba** (*ww.DataColumn*) – The predicted probabilities of the target outputted by the pipeline
- **objective** (*ObjectiveBase*) – The objective to threshold with. Must have a tunable threshold.

**evalml.pipelines.TimeSeriesBinaryClassificationPipeline.predict**

TimeSeriesBinaryClassificationPipeline.predict(*X*, *y=None*, *objective=None*)

Make predictions using selected features.

**Parameters**

- **X** (*ww.DataTable*, *pd.DataFrame*, or *np.ndarray*) – Data of shape [n\_samples, n\_features]
- **y** (*ww.DataColumn*, *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** ww.DataColumn

**evalml.pipelines.TimeSeriesBinaryClassificationPipeline.predict\_proba**

`TimeSeriesBinaryClassificationPipeline.predict_proba(X, y=None)`

Make probability estimates for labels.

**Parameters** *X* (*ww.DataTable*, *pd.DataFrame* or *np.ndarray*) – Data of shape [n\_samples, n\_features]

**Returns** Probability estimates

**Return type** *ww.DataTable*

**evalml.pipelines.TimeSeriesBinaryClassificationPipeline.save**

`TimeSeriesBinaryClassificationPipeline.save(file_path, pickle_protocol=5)`

Saves pipeline at file path

**Parameters**

- **file\_path** (*str*) – location to save file
- **pickle\_protocol** (*int*) – the pickle data stream format.

**Returns** None

**evalml.pipelines.TimeSeriesBinaryClassificationPipeline.score**

`TimeSeriesBinaryClassificationPipeline.score(X, y, objectives)`

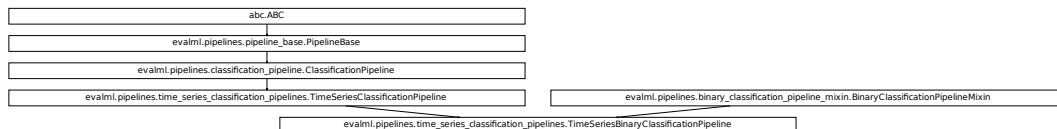
Evaluate model performance on current and additional objectives.

**Parameters**

- **X** (*ww.DataTable*, *pd.DataFrame* or *np.ndarray*) – Data of shape [n\_samples, n\_features]
- **y** (*ww.DataColumn*, *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

**Class Inheritance**

## evalml.pipelines.TimeSeriesMulticlassClassificationPipeline

```
class evalml.pipelines.TimeSeriesMulticlassClassificationPipeline(parameters,  
                                                                ran-  
                                                                dom_seed=0)  
  
    name = 'Time Series Multiclass Classification Pipeline'  
    custom_name = None  
    problem_type = 'time series multiclass'
```

### Instance attributes

<code>classes_</code>	Gets the class names for the problem.
<code>feature_importance</code>	Return importance associated with each feature.
<code>parameters</code>	Returns parameter dictionary for this pipeline

### Methods:

<code>__init__</code>	Machine learning pipeline for time series classification problems made out of transformers and a classifier.
<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>describe</code>	Outputs pipeline details including component parameters
<code>fit</code>	Fit a time series classification pipeline.
<code>get_component</code>	Returns component by name
<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>load</code>	Loads pipeline at file path
<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.

**evalml.pipelines.TimeSeriesMulticlassClassificationPipeline.\_\_init\_\_**

`TimeSeriesMulticlassClassificationPipeline.__init__(parameters, random_seed=0)`

Machine learning pipeline for time series classification problems made out of transformers and a classifier.

**Required Class Variables:** `component_graph` (list): List of components in order. Accepts strings or `ComponentBase` subclasses in the list

**Parameters**

- **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 `gap` and `max_delay` must be specified with the “pipeline” key. For example: `Pipeline(parameters={"pipeline": {"max_delay": 4, "gap": 2}})`.
- **random\_seed** (*int*) – Seed for the random number generator. Defaults to 0.

**evalml.pipelines.TimeSeriesMulticlassClassificationPipeline.can\_tune\_threshold\_with\_objective**

`TimeSeriesMulticlassClassificationPipeline.can_tune_threshold_with_objective(objective)`  
Determine whether the threshold of a binary classification pipeline can be tuned.

**Parameters**

- **pipeline** (`PipelineBase`) – Binary classification pipeline.
- **objective** – Primary AutoMLSearch objective.

**evalml.pipelines.TimeSeriesMulticlassClassificationPipeline.clone**

`TimeSeriesMulticlassClassificationPipeline.clone()`

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.

**evalml.pipelines.TimeSeriesMulticlassClassificationPipeline.compute\_estimator\_features**

`TimeSeriesMulticlassClassificationPipeline.compute_estimator_features(X, y=None)`

Transforms the data by applying all pre-processing components.

**Parameters** **X** (*ww.DataTable*, *pd.DataFrame*) – Input data to the pipeline to transform.

**Returns** New transformed features.

**Return type** *ww.DataTable*

**evalml.pipelines.TimeSeriesMulticlassClassificationPipeline.create\_objectives**

**static** TimeSeriesMulticlassClassificationPipeline.**create\_objectives** (*objectives*)

**evalml.pipelines.TimeSeriesMulticlassClassificationPipeline.describe**

TimeSeriesMulticlassClassificationPipeline.**describe** (*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

**evalml.pipelines.TimeSeriesMulticlassClassificationPipeline.fit**

TimeSeriesMulticlassClassificationPipeline.**fit** (*X, y*)

Fit a time series classification pipeline.

**Parameters**

- **X** (*ww.DataTable, pd.DataFrame or np.ndarray*) – The input training data of shape [n\_samples, n\_features]
- **y** (*ww.DataColumn, pd.Series, np.ndarray*) – The target training targets of length [n\_samples]

**Returns** self

**evalml.pipelines.TimeSeriesMulticlassClassificationPipeline.get\_component**

TimeSeriesMulticlassClassificationPipeline.**get\_component** (*name*)

Returns component by name

**Parameters** **name** (*str*) – Name of component

**Returns** Component to return

**Return type** Component

**evalml.pipelines.TimeSeriesMulticlassClassificationPipeline.graph**

TimeSeriesMulticlassClassificationPipeline.**graph** (*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



**evalml.pipelines.TimeSeriesMulticlassClassificationPipeline.graph\_feature\_importance**

`TimeSeriesMulticlassClassificationPipeline.graph_feature_importance` (*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

**evalml.pipelines.TimeSeriesMulticlassClassificationPipeline.load**

**static** `TimeSeriesMulticlassClassificationPipeline.load` (*file\_path*)

Loads pipeline at file path

**Parameters** `file_path` (*str*) – location to load file

**Returns** `PipelineBase` object

**evalml.pipelines.TimeSeriesMulticlassClassificationPipeline.predict**

`TimeSeriesMulticlassClassificationPipeline.predict` (*X*, *y=None*, *objective=None*)

Make predictions using selected features.

**Parameters**

- **X** (*ww.DataTable, pd.DataFrame, or np.ndarray*) – Data of shape `[n_samples, n_features]`
- **y** (*ww.DataColumn, 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** `ww.DataColumn`

**evalml.pipelines.TimeSeriesMulticlassClassificationPipeline.predict\_proba**

`TimeSeriesMulticlassClassificationPipeline.predict_proba` (*X*, *y=None*)

Make probability estimates for labels.

**Parameters** **X** (*ww.DataTable, pd.DataFrame or np.ndarray*) – Data of shape `[n_samples, n_features]`

**Returns** Probability estimates

**Return type** `ww.DataTable`

**evalml.pipelines.TimeSeriesMulticlassClassificationPipeline.save**

`TimeSeriesMulticlassClassificationPipeline.save(file_path, pickle_protocol=5)`

Saves pipeline at file path

**Parameters**

- **file\_path** (*str*) – location to save file
- **pickle\_protocol** (*int*) – the pickle data stream format.

**Returns** None

**evalml.pipelines.TimeSeriesMulticlassClassificationPipeline.score**

`TimeSeriesMulticlassClassificationPipeline.score(X, y, objectives)`

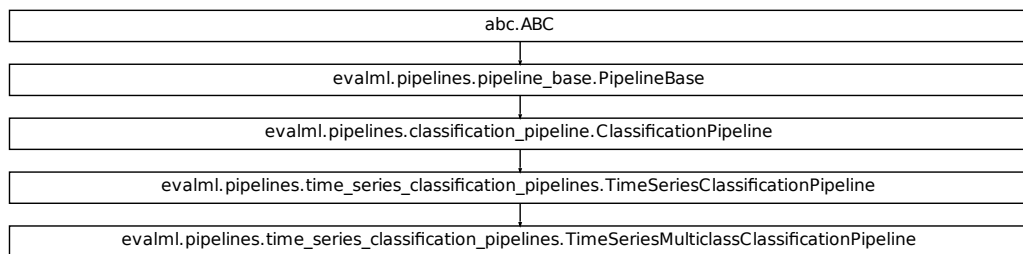
Evaluate model performance on current and additional objectives.

**Parameters**

- **X** (*ww.DataTable, pd.DataFrame or np.ndarray*) – Data of shape [n\_samples, n\_features]
- **y** (*ww.DataColumn, 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

**Class Inheritance**

**evalml.pipelines.TimeSeriesRegressionPipeline**

**class** evalml.pipelines.**TimeSeriesRegressionPipeline** (*parameters*, *random\_seed=0*)

Pipeline base class for time series regression problems.

**name** = 'Time Series Regression Pipeline'

**custom\_name** = None

**problem\_type** = 'time series regression'

**Instance attributes**

<code>feature_importance</code>	Return importance associated with each feature.
<code>parameters</code>	Returns parameter dictionary for this pipeline

**Methods:**

<code>__init__</code>	Machine learning pipeline for time series regression problems made out of transformers and a classifier.
<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>describe</code>	Outputs pipeline details including component parameters
<code>fit</code>	Fit a time series regression pipeline.
<code>get_component</code>	Returns component by name
<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>load</code>	Loads pipeline at file path
<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.

### `evalml.pipelines.TimeSeriesRegressionPipeline.__init__`

`TimeSeriesRegressionPipeline.__init__(parameters, random_seed=0)`

Machine learning pipeline for time series regression problems made out of transformers and a classifier.

**Required Class Variables:** `component_graph` (list): List of components in order. Accepts strings or `ComponentBase` subclasses in the list

#### Parameters

- **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 `gap` and `max_delay` must be specified with the “pipeline” key. For example: `Pipeline(parameters={"pipeline": {"max_delay": 4, "gap": 2}})`.
- **random\_seed** (*int*) – Seed for the random number generator. Defaults to 0.

### `evalml.pipelines.TimeSeriesRegressionPipeline.can_tune_threshold_with_objective`

`TimeSeriesRegressionPipeline.can_tune_threshold_with_objective(objective)`

Determine whether the threshold of a binary classification pipeline can be tuned.

#### Parameters

- **pipeline** (`PipelineBase`) – Binary classification pipeline.
- **objective** – Primary `AutoMLSearch` objective.

### `evalml.pipelines.TimeSeriesRegressionPipeline.clone`

`TimeSeriesRegressionPipeline.clone()`

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.

### `evalml.pipelines.TimeSeriesRegressionPipeline.compute_estimator_features`

`TimeSeriesRegressionPipeline.compute_estimator_features(X, y=None)`

Transforms the data by applying all pre-processing components.

**Parameters** **X** (*ww.DataTable*, *pd.DataFrame*) – Input data to the pipeline to transform.

**Returns** New transformed features.

**Return type** `ww.DataTable`

**evalml.pipelines.TimeSeriesRegressionPipeline.create\_objectives**

**static** TimeSeriesRegressionPipeline.**create\_objectives** (*objectives*)

**evalml.pipelines.TimeSeriesRegressionPipeline.describe**

TimeSeriesRegressionPipeline.**describe** (*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

**evalml.pipelines.TimeSeriesRegressionPipeline.fit**

TimeSeriesRegressionPipeline.**fit** (*X, y*)

Fit a time series regression pipeline.

**Parameters**

- **X** (*ww.DataTable, pd.DataFrame or np.ndarray*) – The input training data of shape [n\_samples, n\_features]
- **y** (*ww.DataColumn, pd.Series, np.ndarray*) – The target training targets of length [n\_samples]

**Returns** self

**evalml.pipelines.TimeSeriesRegressionPipeline.get\_component**

TimeSeriesRegressionPipeline.**get\_component** (*name*)

Returns component by name

**Parameters** **name** (*str*) – Name of component

**Returns** Component to return

**Return type** Component

**evalml.pipelines.TimeSeriesRegressionPipeline.graph**

TimeSeriesRegressionPipeline.**graph** (*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

**evalml.pipelines.TimeSeriesRegressionPipeline.graph\_feature\_importance**

`TimeSeriesRegressionPipeline.graph_feature_importance(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

**evalml.pipelines.TimeSeriesRegressionPipeline.load**

**static** `TimeSeriesRegressionPipeline.load(file_path)`  
Loads pipeline at file path

**Parameters** `file_path` (*str*) – location to load file

**Returns** `PipelineBase` object

**evalml.pipelines.TimeSeriesRegressionPipeline.predict**

`TimeSeriesRegressionPipeline.predict(X, y=None, objective=None)`  
Make predictions using selected features.

**Parameters**

- **X** (*ww.DataTable, pd.DataFrame, or np.ndarray*) – Data of shape `[n_samples, n_features]`
- **y** (*ww.DataColumn, 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** `ww.DataColumn`

**evalml.pipelines.TimeSeriesRegressionPipeline.save**

`TimeSeriesRegressionPipeline.save(file_path, pickle_protocol=5)`  
Saves pipeline at file path

**Parameters**

- **file\_path** (*str*) – location to save file
- **pickle\_protocol** (*int*) – the pickle data stream format.

**Returns** `None`

## evalml.pipelines.TimeSeriesRegressionPipeline.score

`TimeSeriesRegressionPipeline.score` (*X*, *y*, *objectives*)  
 Evaluate model performance on current and additional objectives.

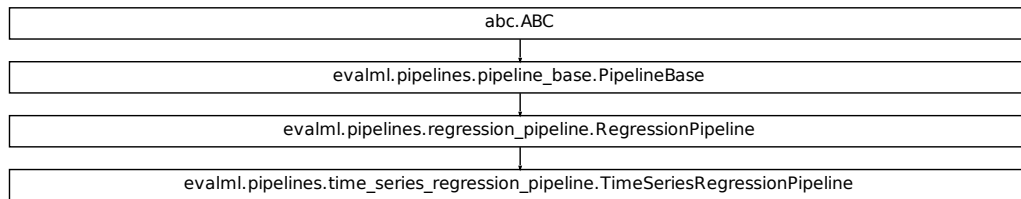
### Parameters

- **X** (*ww.DataTable*, *pd.DataFrame* or *np.ndarray*) – Data of shape [n\_samples, n\_features]
- **y** (*pd.Series*, *ww.DataColumn*) – 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

## Class Inheritance



## 5.5.2 Classification Pipelines

<a href="#"><i>BaselineBinaryPipeline</i></a>	Baseline Pipeline for binary classification.
<a href="#"><i>BaselineMulticlassPipeline</i></a>	Baseline Pipeline for multiclass classification.
<a href="#"><i>ModeBaselineBinaryPipeline</i></a>	Mode Baseline Pipeline for binary classification.
<a href="#"><i>ModeBaselineMulticlassPipeline</i></a>	Mode Baseline Pipeline for multiclass classification.

## evalml.pipelines.BaselineBinaryPipeline

```

class evalml.pipelines.BaselineBinaryPipeline(parameters, random_seed=0)
    Baseline Pipeline for binary classification.

    name = 'Baseline Classification Pipeline'
    custom_name = 'Baseline Classification Pipeline'
    summary = 'Baseline Classifier'
    component_graph = ['Baseline Classifier']
    problem_type = 'binary'
  
```

```
model_family = 'baseline'
hyperparameters = {'Baseline Classifier': {}}
custom_hyperparameters = None
default_parameters = {'Baseline Classifier': {'strategy': 'mode'}}
```

### Instance attributes

<code>classes_</code>	Gets the class names for the problem.
<code>feature_importance</code>	Return importance associated with each feature.
<code>linearized_component_graph</code>	
<code>parameters</code>	Returns parameter dictionary for this pipeline
<code>threshold</code>	Threshold used to make a prediction.

### Methods:

<code>__init__</code>	Machine learning classification pipeline made out of transformers and a classifier.
<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>describe</code>	Outputs pipeline details including component parameters
<code>fit</code>	Build a classification model. For string and categorical targets, classes are sorted
<code>get_component</code>	Returns component by name
<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>load</code>	Loads pipeline at file path
<code>optimize_threshold</code>	Optimize the pipeline threshold given the objective to use.
<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



## evalml.pipelines.BaselineBinaryPipeline.\_\_init\_\_

BaselineBinaryPipeline.\_\_init\_\_(parameters, random\_seed=0)

Machine learning classification pipeline made out of transformers and a classifier.

**Required Class Variables:** component\_graph (list): List of components in order. Accepts strings or ComponentBase subclasses in the list

### Parameters

- **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.
- **random\_state** (*int*) – Deprecated - use random\_seed instead.
- **random\_seed** (*int*) – Seed for the random number generator. Defaults to 0.

## evalml.pipelines.BaselineBinaryPipeline.can\_tune\_threshold\_with\_objective

BaselineBinaryPipeline.can\_tune\_threshold\_with\_objective(objective)

Determine whether the threshold of a binary classification pipeline can be tuned.

### Parameters

- **pipeline** (*PipelineBase*) – Binary classification pipeline.
- **objective** – Primary AutoMLSearch objective.

## evalml.pipelines.BaselineBinaryPipeline.clone

BaselineBinaryPipeline.clone()

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.

## evalml.pipelines.BaselineBinaryPipeline.compute\_estimator\_features

BaselineBinaryPipeline.compute\_estimator\_features(X, y=None)

Transforms the data by applying all pre-processing components.

**Parameters** **X** (*ww.DataTable*, *pd.DataFrame*) – Input data to the pipeline to transform.

**Returns** New transformed features.

**Return type** *ww.DataTable*

### `evalml.pipelines.BaselineBinaryPipeline.create_objectives`

**static** `BaselineBinaryPipeline.create_objectives(objectives)`

### `evalml.pipelines.BaselineBinaryPipeline.describe`

`BaselineBinaryPipeline.describe(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

### `evalml.pipelines.BaselineBinaryPipeline.fit`

`BaselineBinaryPipeline.fit(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** (*ww.DataTable, pd.DataFrame or np.ndarray*) – The input training data of shape `[n_samples, n_features]`
- **y** (*ww.DataColumn, pd.Series, np.ndarray*) – The target training labels of length `[n_samples]`

**Returns** self

### `evalml.pipelines.BaselineBinaryPipeline.get_component`

`BaselineBinaryPipeline.get_component(name)`

Returns component by name

**Parameters** `name` (*str*) – Name of component

**Returns** Component to return

**Return type** Component

### `evalml.pipelines.BaselineBinaryPipeline.graph`

`BaselineBinaryPipeline.graph(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

**evalml.pipelines.BaselineBinaryPipeline.graph\_feature\_importance**

`BaselineBinaryPipeline.graph_feature_importance(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

**evalml.pipelines.BaselineBinaryPipeline.load**

**static** `BaselineBinaryPipeline.load(file_path)`

Loads pipeline at file path

**Parameters** `file_path` (*str*) – location to load file

**Returns** `PipelineBase` object

**evalml.pipelines.BaselineBinaryPipeline.optimize\_threshold**

`BaselineBinaryPipeline.optimize_threshold(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` (*ww.DataTable*) – Input features
- `y` (*ww.DataColumn*) – Input target values
- `y_pred_proba` (*ww.DataColumn*) – The predicted probabilities of the target outputted by the pipeline
- `objective` (*ObjectiveBase*) – The objective to threshold with. Must have a tunable threshold.

**evalml.pipelines.BaselineBinaryPipeline.predict**

`BaselineBinaryPipeline.predict(X, objective=None)`

Make predictions using selected features.

**Parameters**

- `X` (*ww.DataTable, 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** `ww.DataColumn`

### evalml.pipelines.BaselineBinaryPipeline.predict\_proba

BaselineBinaryPipeline.**predict\_proba**(*X*)

Make probability estimates for labels. Assumes that the column at index 1 represents the positive label case.

**Parameters** *X* (*ww.DataTable*, *pd.DataFrame* or *np.ndarray*) – Data of shape [n\_samples, n\_features]

**Returns** Probability estimates

**Return type** *ww.DataTable*

### evalml.pipelines.BaselineBinaryPipeline.save

BaselineBinaryPipeline.**save**(*file\_path*, *pickle\_protocol=5*)

Saves pipeline at file path

**Parameters**

- **file\_path** (*str*) – location to save file
- **pickle\_protocol** (*int*) – the pickle data stream format.

**Returns** None

### evalml.pipelines.BaselineBinaryPipeline.score

BaselineBinaryPipeline.**score**(*X*, *y*, *objectives*)

Evaluate model performance on objectives

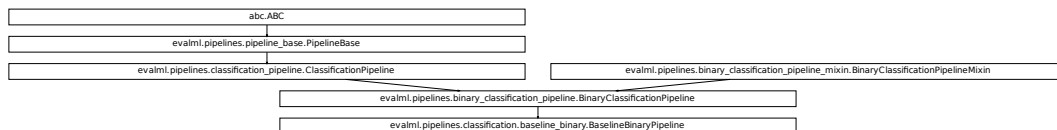
**Parameters**

- **X** (*ww.DataTable*, *pd.DataFrame* or *np.ndarray*) – Data of shape [n\_samples, n\_features]
- **y** (*ww.DataColumn*, *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

## Class Inheritance



**evalml.pipelines.BaselineMulticlassPipeline**

```
class evalml.pipelines.BaselineMulticlassPipeline(parameters, random_seed=0)
    Baseline Pipeline for multiclass classification.

    name = 'Baseline Multiclass Classification Pipeline'
    custom_name = 'Baseline Multiclass Classification Pipeline'
    summary = 'Baseline Classifier'
    component_graph = ['Baseline Classifier']
    problem_type = 'multiclass'
    model_family = 'baseline'
    hyperparameters = {'Baseline Classifier': {}}
    custom_hyperparameters = None
    default_parameters = {'Baseline Classifier': {'strategy': 'mode'}}
```

**Instance attributes**

<code>classes_</code>	Gets the class names for the problem.
<code>feature_importance</code>	Return importance associated with each feature.
<code>linearized_component_graph</code>	
<code>parameters</code>	Returns parameter dictionary for this pipeline

**Methods:**

<code>__init__</code>	Machine learning classification pipeline made out of transformers and a classifier.
<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>describe</code>	Outputs pipeline details including component parameters
<code>fit</code>	Build a classification model. For string and categorical targets, classes are sorted
<code>get_component</code>	Returns component by name
<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>load</code>	Loads pipeline at file path
<code>predict</code>	Make predictions using selected features.
<code>predict_proba</code>	Make probability estimates for labels.

continues on next page

Table 48 – continued from previous page

<code>save</code>	Saves pipeline at file path
<code>score</code>	Evaluate model performance on objectives

### `evalml.pipelines.BaselineMulticlassPipeline.__init__`

`BaselineMulticlassPipeline.__init__` (*parameters*, *random\_seed=0*)

Machine learning classification pipeline made out of transformers and a classifier.

**Required Class Variables:** `component_graph` (list): List of components in order. Accepts strings or `ComponentBase` subclasses in the list

#### Parameters

- **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.
- **random\_state** (*int*) – Deprecated - use `random_seed` instead.
- **random\_seed** (*int*) – Seed for the random number generator. Defaults to 0.

### `evalml.pipelines.BaselineMulticlassPipeline.can_tune_threshold_with_objective`

`BaselineMulticlassPipeline.can_tune_threshold_with_objective` (*objective*)

Determine whether the threshold of a binary classification pipeline can be tuned.

#### Parameters

- **pipeline** (`PipelineBase`) – Binary classification pipeline.
- **objective** – Primary `AutoMLSearch` objective.

### `evalml.pipelines.BaselineMulticlassPipeline.clone`

`BaselineMulticlassPipeline.clone` ()

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.

### `evalml.pipelines.BaselineMulticlassPipeline.compute_estimator_features`

`BaselineMulticlassPipeline.compute_estimator_features` (*X*, *y=None*)

Transforms the data by applying all pre-processing components.

**Parameters** *X* (`ww.DataTable`, `pd.DataFrame`) – Input data to the pipeline to transform.

**Returns** New transformed features.

**Return type** `ww.DataTable`

**evalml.pipelines.BaselineMulticlassPipeline.create\_objectives**

**static** BaselineMulticlassPipeline.**create\_objectives** (*objectives*)

**evalml.pipelines.BaselineMulticlassPipeline.describe**

BaselineMulticlassPipeline.**describe** (*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

**evalml.pipelines.BaselineMulticlassPipeline.fit**

BaselineMulticlassPipeline.**fit** (*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** (*ww.DataTable, pd.DataFrame or np.ndarray*) – The input training data of shape [n\_samples, n\_features]
- **y** (*ww.DataColumn, pd.Series, np.ndarray*) – The target training labels of length [n\_samples]

**Returns** self

**evalml.pipelines.BaselineMulticlassPipeline.get\_component**

BaselineMulticlassPipeline.**get\_component** (*name*)

Returns component by name

**Parameters** **name** (*str*) – Name of component

**Returns** Component to return

**Return type** Component

**evalml.pipelines.BaselineMulticlassPipeline.graph**

BaselineMulticlassPipeline.**graph** (*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

### `evalml.pipelines.BaselineMulticlassPipeline.graph_feature_importance`

`BaselineMulticlassPipeline.graph_feature_importance(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

### `evalml.pipelines.BaselineMulticlassPipeline.load`

**static** `BaselineMulticlassPipeline.load(file_path)`

Loads pipeline at file path

**Parameters** `file_path` (*str*) – location to load file

**Returns** `PipelineBase` object

### `evalml.pipelines.BaselineMulticlassPipeline.predict`

`BaselineMulticlassPipeline.predict(X, objective=None)`

Make predictions using selected features.

**Parameters**

- **X** (*ww.DataTable, 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** `ww.DataColumn`

### `evalml.pipelines.BaselineMulticlassPipeline.predict_proba`

`BaselineMulticlassPipeline.predict_proba(X)`

Make probability estimates for labels.

**Parameters** **X** (*ww.DataTable, pd.DataFrame or np.ndarray*) – Data of shape `[n_samples, n_features]`

**Returns** Probability estimates

**Return type** `ww.DataTable`



**evalml.pipelines.BaselineMulticlassPipeline.save**

`BaselineMulticlassPipeline.save(file_path, pickle_protocol=5)`

Saves pipeline at file path

**Parameters**

- **file\_path** (*str*) – location to save file
- **pickle\_protocol** (*int*) – the pickle data stream format.

**Returns** None

**evalml.pipelines.BaselineMulticlassPipeline.score**

`BaselineMulticlassPipeline.score(X, y, objectives)`

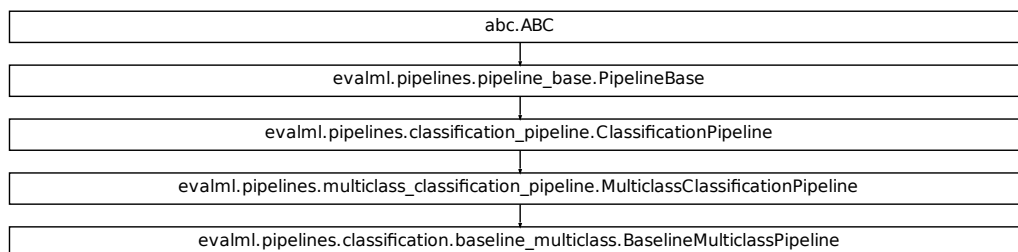
Evaluate model performance on objectives

**Parameters**

- **X** (*ww.DataTable, pd.DataFrame or np.ndarray*) – Data of shape [n\_samples, n\_features]
- **y** (*ww.DataColumn, 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

**Class Inheritance**

**evalml.pipelines.ModeBaselineBinaryPipeline**

```

class evalml.pipelines.ModeBaselineBinaryPipeline (parameters, random_seed=0)
    Mode Baseline Pipeline for binary classification.

    name = 'Mode Baseline Binary Classification Pipeline'
    custom_name = 'Mode Baseline Binary Classification Pipeline'
    summary = 'Baseline Classifier'
    component_graph = ['Baseline Classifier']
    problem_type = 'binary'
    model_family = 'baseline'
    hyperparameters = {'Baseline Classifier': {}}
    custom_hyperparameters = {'strategy': ['mode']}
    default_parameters = {'Baseline Classifier': {'strategy': 'mode'}}

```

**Instance attributes**

<code>classes_</code>	Gets the class names for the problem.
<code>feature_importance</code>	Return importance associated with each feature.
<code>linearized_component_graph</code>	
<code>parameters</code>	Returns parameter dictionary for this pipeline
<code>threshold</code>	Threshold used to make a prediction.

**Methods:**

<code>__init__</code>	Machine learning classification pipeline made out of transformers and a classifier.
<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>describe</code>	Outputs pipeline details including component parameters
<code>fit</code>	Build a classification model. For string and categorical targets, classes are sorted
<code>get_component</code>	Returns component by name
<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>load</code>	Loads pipeline at file path

continues on next page

Table 50 – continued from previous page

<code>optimize_threshold</code>	Optimize the pipeline threshold given the objective to use.
<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

### `evalml.pipelines.ModeBaselineBinaryPipeline.__init__`

`ModeBaselineBinaryPipeline.__init__(parameters, random_seed=0)`

Machine learning classification pipeline made out of transformers and a classifier.

**Required Class Variables:** `component_graph` (list): List of components in order. Accepts strings or `ComponentBase` subclasses in the list

#### Parameters

- **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.
- **random\_state** (*int*) – Deprecated - use `random_seed` instead.
- **random\_seed** (*int*) – Seed for the random number generator. Defaults to 0.

### `evalml.pipelines.ModeBaselineBinaryPipeline.can_tune_threshold_with_objective`

`ModeBaselineBinaryPipeline.can_tune_threshold_with_objective(objective)`

Determine whether the threshold of a binary classification pipeline can be tuned.

#### Parameters

- **pipeline** (`PipelineBase`) – Binary classification pipeline.
- **objective** – Primary `AutoMLSearch` objective.

### `evalml.pipelines.ModeBaselineBinaryPipeline.clone`

`ModeBaselineBinaryPipeline.clone()`

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.

**evalml.pipelines.ModeBaselineBinaryPipeline.compute\_estimator\_features**

ModeBaselineBinaryPipeline.**compute\_estimator\_features**(*X*, *y=None*)

Transforms the data by applying all pre-processing components.

**Parameters** *X* (*ww.DataTable*, *pd.DataFrame*) – Input data to the pipeline to transform.

**Returns** New transformed features.

**Return type** *ww.DataTable*

**evalml.pipelines.ModeBaselineBinaryPipeline.create\_objectives**

**static** ModeBaselineBinaryPipeline.**create\_objectives**(*objectives*)

**evalml.pipelines.ModeBaselineBinaryPipeline.describe**

ModeBaselineBinaryPipeline.**describe**(*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*

**evalml.pipelines.ModeBaselineBinaryPipeline.fit**

ModeBaselineBinaryPipeline.**fit**(*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** (*ww.DataTable*, *pd.DataFrame* or *np.ndarray*) – The input training data of shape [*n\_samples*, *n\_features*]
- **y** (*ww.DataColumn*, *pd.Series*, *np.ndarray*) – The target training labels of length [*n\_samples*]

**Returns** *self*

**evalml.pipelines.ModeBaselineBinaryPipeline.get\_component**

ModeBaselineBinaryPipeline.**get\_component**(*name*)

Returns component by name

**Parameters** **name** (*str*) – Name of component

**Returns** Component to return

**Return type** *Component*

**evalml.pipelines.ModeBaselineBinaryPipeline.graph**

ModeBaselineBinaryPipeline.**graph** (*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

**evalml.pipelines.ModeBaselineBinaryPipeline.graph\_feature\_importance**

ModeBaselineBinaryPipeline.**graph\_feature\_importance** (*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

**evalml.pipelines.ModeBaselineBinaryPipeline.load**

**static** ModeBaselineBinaryPipeline.**load** (*file\_path*)

Loads pipeline at file path

**Parameters** **file\_path** (*str*) – location to load file

**Returns** PipelineBase object

**evalml.pipelines.ModeBaselineBinaryPipeline.optimize\_threshold**

ModeBaselineBinaryPipeline.**optimize\_threshold** (*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** (*ww.DataTable*) – Input features
- **y** (*ww.DataColumn*) – Input target values
- **y\_pred\_proba** (*ww.DataColumn*) – The predicted probabilities of the target outputted by the pipeline
- **objective** (*ObjectiveBase*) – The objective to threshold with. Must have a tunable threshold.

### `evalml.pipelines.ModeBaselineBinaryPipeline.predict`

`ModeBaselineBinaryPipeline.predict` (*X*, *objective=None*)

Make predictions using selected features.

#### Parameters

- **X** (*ww.DataTable*, *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** *ww.DataColumn*

### `evalml.pipelines.ModeBaselineBinaryPipeline.predict_proba`

`ModeBaselineBinaryPipeline.predict_proba` (*X*)

Make probability estimates for labels. Assumes that the column at index 1 represents the positive label case.

**Parameters** **X** (*ww.DataTable*, *pd.DataFrame* or *np.ndarray*) – Data of shape [n\_samples, n\_features]

**Returns** Probability estimates

**Return type** *ww.DataTable*

### `evalml.pipelines.ModeBaselineBinaryPipeline.save`

`ModeBaselineBinaryPipeline.save` (*file\_path*, *pickle\_protocol=5*)

Saves pipeline at file path

#### Parameters

- **file\_path** (*str*) – location to save file
- **pickle\_protocol** (*int*) – the pickle data stream format.

**Returns** None

### `evalml.pipelines.ModeBaselineBinaryPipeline.score`

`ModeBaselineBinaryPipeline.score` (*X*, *y*, *objectives*)

Evaluate model performance on objectives

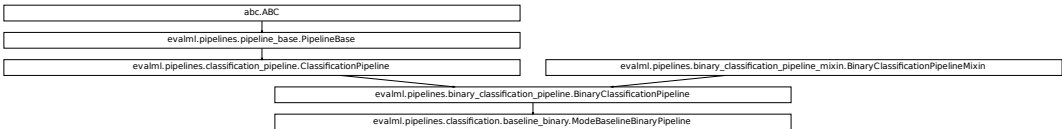
#### Parameters

- **X** (*ww.DataTable*, *pd.DataFrame* or *np.ndarray*) – Data of shape [n\_samples, n\_features]
- **y** (*ww.DataColumn*, *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

Class Inheritance



evalml.pipelines.ModeBaselineMulticlassPipeline

```
class evalml.pipelines.ModeBaselineMulticlassPipeline(parameters, random_seed=0)
    Mode Baseline Pipeline for multiclass classification.

    name = 'Mode Baseline Multiclass Classification Pipeline'
    custom_name = 'Mode Baseline Multiclass Classification Pipeline'
    summary = 'Baseline Classifier'
    component_graph = ['Baseline Classifier']
    problem_type = 'multiclass'
    model_family = 'baseline'
    hyperparameters = {'Baseline Classifier': {}}
    custom_hyperparameters = {'strategy': ['mode']}
    default_parameters = {'Baseline Classifier': {'strategy': 'mode'}}
```

Instance attributes

classes_	Gets the class names for the problem.
feature_importance	Return importance associated with each feature.
linearized_component_graph	
parameters	Returns parameter dictionary for this pipeline

Methods:

<code>__init__</code>	Machine learning classification pipeline made out of transformers and a classifier.
<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.

continues on next page

Table 52 – continued from previous page

<code>compute_estimator_features</code>	Transforms the data by applying all pre-processing components.
<code>create_objectives</code>	
<code>describe</code>	Outputs pipeline details including component parameters
<code>fit</code>	Build a classification model. For string and categorical targets, classes are sorted
<code>get_component</code>	Returns component by name
<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>load</code>	Loads pipeline at file path
<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

### `evalml.pipelines.ModeBaselineMulticlassPipeline.__init__`

`ModeBaselineMulticlassPipeline.__init__(parameters, random_seed=0)`

Machine learning classification pipeline made out of transformers and a classifier.

**Required Class Variables:** `component_graph` (list): List of components in order. Accepts strings or `ComponentBase` subclasses in the list

#### Parameters

- **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.
- **random\_state** (*int*) – Deprecated - use `random_seed` instead.
- **random\_seed** (*int*) – Seed for the random number generator. Defaults to 0.

### `evalml.pipelines.ModeBaselineMulticlassPipeline.can_tune_threshold_with_objective`

`ModeBaselineMulticlassPipeline.can_tune_threshold_with_objective(objective)`

Determine whether the threshold of a binary classification pipeline can be tuned.

#### Parameters

- **pipeline** (`PipelineBase`) – Binary classification pipeline.
- **objective** – Primary `AutoMLSearch` objective.



**evalml.pipelines.ModeBaselineMulticlassPipeline.clone**

ModeBaselineMulticlassPipeline.**clone**()

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.

**evalml.pipelines.ModeBaselineMulticlassPipeline.compute\_estimator\_features**

ModeBaselineMulticlassPipeline.**compute\_estimator\_features**(X, y=None)

Transforms the data by applying all pre-processing components.

**Parameters** **X** (*ww.DataTable*, *pd.DataFrame*) – Input data to the pipeline to transform.

**Returns** New transformed features.

**Return type** *ww.DataTable*

**evalml.pipelines.ModeBaselineMulticlassPipeline.create\_objectives**

**static** ModeBaselineMulticlassPipeline.**create\_objectives**(objectives)

**evalml.pipelines.ModeBaselineMulticlassPipeline.describe**

ModeBaselineMulticlassPipeline.**describe**(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*

**evalml.pipelines.ModeBaselineMulticlassPipeline.fit**

ModeBaselineMulticlassPipeline.**fit**(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** (*ww.DataTable*, *pd.DataFrame* or *np.ndarray*) – The input training data of shape `[n_samples, n_features]`
- **y** (*ww.DataColumn*, *pd.Series*, *np.ndarray*) – The target training labels of length `[n_samples]`

**Returns** *self*

**evalml.pipelines.ModeBaselineMulticlassPipeline.get\_component**

ModeBaselineMulticlassPipeline.**get\_component** (*name*)

Returns component by name

**Parameters** **name** (*str*) – Name of component

**Returns** Component to return

**Return type** Component

**evalml.pipelines.ModeBaselineMulticlassPipeline.graph**

ModeBaselineMulticlassPipeline.**graph** (*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

**evalml.pipelines.ModeBaselineMulticlassPipeline.graph\_feature\_importance**

ModeBaselineMulticlassPipeline.**graph\_feature\_importance** (*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

**evalml.pipelines.ModeBaselineMulticlassPipeline.load**

**static** ModeBaselineMulticlassPipeline.**load** (*file\_path*)

Loads pipeline at file path

**Parameters** **file\_path** (*str*) – location to load file

**Returns** PipelineBase object

**evalml.pipelines.ModeBaselineMulticlassPipeline.predict**

ModeBaselineMulticlassPipeline.**predict** (*X, objective=None*)

Make predictions using selected features.

**Parameters**

- **X** (*ww.DataTable, 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** ww.DataColumn

### evalml.pipelines.ModeBaselineMulticlassPipeline.predict\_proba

ModeBaselineMulticlassPipeline.**predict\_proba**(X)

Make probability estimates for labels.

**Parameters** **X** (ww.DataTable, pd.DataFrame or np.ndarray) – Data of shape [n\_samples, n\_features]

**Returns** Probability estimates

**Return type** ww.DataTable

### evalml.pipelines.ModeBaselineMulticlassPipeline.save

ModeBaselineMulticlassPipeline.**save**(file\_path, pickle\_protocol=5)

Saves pipeline at file path

**Parameters**

- **file\_path** (str) – location to save file
- **pickle\_protocol** (int) – the pickle data stream format.

**Returns** None

### evalml.pipelines.ModeBaselineMulticlassPipeline.score

ModeBaselineMulticlassPipeline.**score**(X, y, objectives)

Evaluate model performance on objectives

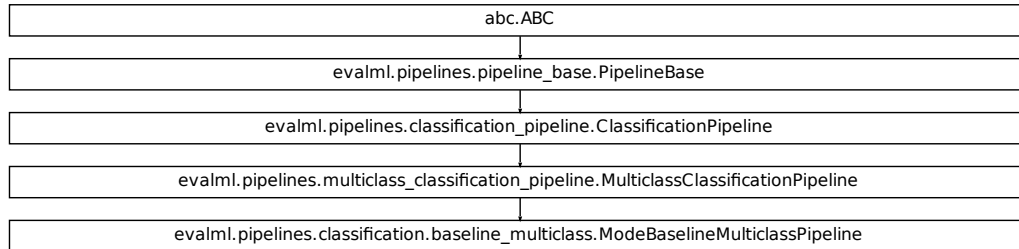
**Parameters**

- **X** (ww.DataTable, pd.DataFrame or np.ndarray) – Data of shape [n\_samples, n\_features]
- **y** (ww.DataColumn, 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

## Class Inheritance



### 5.5.3 Regression Pipelines

<i>BaselineRegressionPipeline</i>	Baseline Pipeline for regression problems.
<i>MeanBaselineRegressionPipeline</i>	Baseline Pipeline for regression problems.
<i>TimeSeriesBaselineRegressionPipeline</i>	Baseline Pipeline for time series regression problems.

#### evalml.pipelines.BaselineRegressionPipeline

```
class evalml.pipelines.BaselineRegressionPipeline (parameters, random_seed=0)
    Baseline Pipeline for regression problems.

    name = 'Baseline Regression Pipeline'
    custom_name = None
    summary = 'Baseline Regressor'
    component_graph = ['Baseline Regressor']
    problem_type = 'regression'
    model_family = 'baseline'
    hyperparameters = {'Baseline Regressor': {}}
    custom_hyperparameters = None
    default_parameters = {'Baseline Regressor': {'strategy': 'mean'}}
```

**Instance attributes**

<code>feature_importance</code>	Return importance associated with each feature.
<code>linearized_component_graph</code>	
<code>parameters</code>	Returns parameter dictionary for this pipeline

**Methods:**

<code>__init__</code>	Machine learning pipeline made out of transformers and a estimator.
<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>describe</code>	Outputs pipeline details including component parameters
<code>fit</code>	Build a regression model.
<code>get_component</code>	Returns component by name
<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>load</code>	Loads pipeline at file path
<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

**evalml.pipelines.BaselineRegressionPipeline.\_\_init\_\_**

`BaselineRegressionPipeline.__init__(parameters, random_seed=0)`

Machine learning pipeline made out of transformers and a estimator.

**Required Class Variables:**

**component\_graph (list or dict): List of components in order. Accepts strings or ComponentBase subclasses in the**

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**

- **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.

- **random\_seed** (*int*) – Seed for the random number generator. Defaults to 0.

### **evalml.pipelines.BaselineRegressionPipeline.can\_tune\_threshold\_with\_objective**

`BaselineRegressionPipeline.can_tune_threshold_with_objective(objective)`

Determine whether the threshold of a binary classification pipeline can be tuned.

#### **Parameters**

- **pipeline** (`PipelineBase`) – Binary classification pipeline.
- **objective** – Primary AutoMLSearch objective.

### **evalml.pipelines.BaselineRegressionPipeline.clone**

`BaselineRegressionPipeline.clone()`

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.

### **evalml.pipelines.BaselineRegressionPipeline.compute\_estimator\_features**

`BaselineRegressionPipeline.compute_estimator_features(X, y=None)`

Transforms the data by applying all pre-processing components.

**Parameters** **X** (`ww.DataTable`, `pd.DataFrame`) – Input data to the pipeline to transform.

**Returns** New transformed features.

**Return type** `ww.DataTable`

### **evalml.pipelines.BaselineRegressionPipeline.create\_objectives**

**static** `BaselineRegressionPipeline.create_objectives(objectives)`

### **evalml.pipelines.BaselineRegressionPipeline.describe**

`BaselineRegressionPipeline.describe(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`

**evalml.pipelines.BaselineRegressionPipeline.fit**

`BaselineRegressionPipeline.fit(X, y)`

Build a regression model.

**Parameters**

- **X** (*ww.DataTable*, *pd.DataFrame* or *np.ndarray*) – The input training data of shape [n\_samples, n\_features]
- **y** (*ww.DataColumn*, *pd.Series*, *np.ndarray*) – The target training data of length [n\_samples]

**Returns** self

**evalml.pipelines.BaselineRegressionPipeline.get\_component**

`BaselineRegressionPipeline.get_component(name)`

Returns component by name

**Parameters** **name** (*str*) – Name of component

**Returns** Component to return

**Return type** Component

**evalml.pipelines.BaselineRegressionPipeline.graph**

`BaselineRegressionPipeline.graph(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

**evalml.pipelines.BaselineRegressionPipeline.graph\_feature\_importance**

`BaselineRegressionPipeline.graph_feature_importance(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

### `evalml.pipelines.BaselineRegressionPipeline.load`

**static** `BaselineRegressionPipeline.load(file_path)`

Loads pipeline at file path

**Parameters** `file_path` (*str*) – location to load file

**Returns** PipelineBase object

### `evalml.pipelines.BaselineRegressionPipeline.predict`

`BaselineRegressionPipeline.predict(X, objective=None)`

Make predictions using selected features.

**Parameters**

- **X** (*ww.DataTable*, *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** *ww.DataColumn*

### `evalml.pipelines.BaselineRegressionPipeline.save`

`BaselineRegressionPipeline.save(file_path, pickle_protocol=5)`

Saves pipeline at file path

**Parameters**

- **file\_path** (*str*) – location to save file
- **pickle\_protocol** (*int*) – the pickle data stream format.

**Returns** None

### `evalml.pipelines.BaselineRegressionPipeline.score`

`BaselineRegressionPipeline.score(X, y, objectives)`

Evaluate model performance on current and additional objectives

**Parameters**

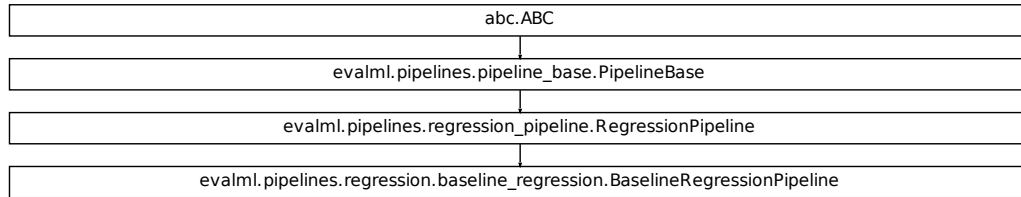
- **X** (*ww.DataTable*, *pd.DataFrame*, or *np.ndarray*) – Data of shape [n\_samples, n\_features]
- **y** (*ww.DataColumn*, *pd.Series*, or *np.ndarray*) – True values of length [n\_samples]
- **objectives** (*list*) – Non-empty list of objectives to score on

**Returns** Ordered dictionary of objective scores

**Return type** dict



## Class Inheritance



### evalml.pipelines.MeanBaselineRegressionPipeline

```

class evalml.pipelines.MeanBaselineRegressionPipeline(parameters, random_seed=0)
    Baseline Pipeline for regression problems.
    name = 'Mean Baseline Regression Pipeline'
    custom_name = None
    summary = 'Baseline Regressor'
    component_graph = ['Baseline Regressor']
    problem_type = 'regression'
    model_family = 'baseline'
    hyperparameters = {'Baseline Regressor': {}}
    custom_hyperparameters = {'strategy': ['mean']}
    default_parameters = {'Baseline Regressor': {'strategy': 'mean'}}
  
```

#### Instance attributes

<code>feature_importance</code>	Return importance associated with each feature.
<code>linearized_component_graph</code>	
<code>parameters</code>	Returns parameter dictionary for this pipeline

## Methods:

<code>__init__</code>	Machine learning pipeline made out of transformers and a estimator.
<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>describe</code>	Outputs pipeline details including component parameters
<code>fit</code>	Build a regression model.
<code>get_component</code>	Returns component by name
<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>load</code>	Loads pipeline at file path
<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

## `evalml.pipelines.MeanBaselineRegressionPipeline.__init__`

`MeanBaselineRegressionPipeline.__init__(parameters, random_seed=0)`

Machine learning pipeline made out of transformers and a estimator.

### Required Class Variables:

**component\_graph (list or dict): List of components in order. Accepts strings or `ComponentBase` subclasses in the**

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

- **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.
- **random\_seed** (*int*) – Seed for the random number generator. Defaults to 0.

**evalml.pipelines.MeanBaselineRegressionPipeline.can\_tune\_threshold\_with\_objective**

MeanBaselineRegressionPipeline.**can\_tune\_threshold\_with\_objective** (*objective*)

Determine whether the threshold of a binary classification pipeline can be tuned.

**Parameters**

- **pipeline** (*PipelineBase*) – Binary classification pipeline.
- **objective** – Primary AutoMLSearch objective.

**evalml.pipelines.MeanBaselineRegressionPipeline.clone**

MeanBaselineRegressionPipeline.**clone** ()

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.

**evalml.pipelines.MeanBaselineRegressionPipeline.compute\_estimator\_features**

MeanBaselineRegressionPipeline.**compute\_estimator\_features** (*X*, *y=None*)

Transforms the data by applying all pre-processing components.

**Parameters** **X** (*ww.DataTable*, *pd.DataFrame*) – Input data to the pipeline to transform.

**Returns** New transformed features.

**Return type** *ww.DataTable*

**evalml.pipelines.MeanBaselineRegressionPipeline.create\_objectives**

**static** MeanBaselineRegressionPipeline.**create\_objectives** (*objectives*)

**evalml.pipelines.MeanBaselineRegressionPipeline.describe**

MeanBaselineRegressionPipeline.**describe** (*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*

### **evalml.pipelines.MeanBaselineRegressionPipeline.fit**

`MeanBaselineRegressionPipeline.fit(X, y)`

Build a regression model.

#### **Parameters**

- **X** (*ww.DataTable*, *pd.DataFrame* or *np.ndarray*) – The input training data of shape [n\_samples, n\_features]
- **y** (*ww.DataColumn*, *pd.Series*, *np.ndarray*) – The target training data of length [n\_samples]

**Returns** self

### **evalml.pipelines.MeanBaselineRegressionPipeline.get\_component**

`MeanBaselineRegressionPipeline.get_component(name)`

Returns component by name

**Parameters** **name** (*str*) – Name of component

**Returns** Component to return

**Return type** Component

### **evalml.pipelines.MeanBaselineRegressionPipeline.graph**

`MeanBaselineRegressionPipeline.graph(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

### **evalml.pipelines.MeanBaselineRegressionPipeline.graph\_feature\_importance**

`MeanBaselineRegressionPipeline.graph_feature_importance(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

**evalml.pipelines.MeanBaselineRegressionPipeline.load****static** MeanBaselineRegressionPipeline.**load**(*file\_path*)

Loads pipeline at file path

**Parameters** **file\_path** (*str*) – location to load file**Returns** PipelineBase object**evalml.pipelines.MeanBaselineRegressionPipeline.predict**MeanBaselineRegressionPipeline.**predict**(*X*, *objective=None*)

Make predictions using selected features.

**Parameters**

- **X** (*ww.DataTable*, *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** ww.DataColumn**evalml.pipelines.MeanBaselineRegressionPipeline.save**MeanBaselineRegressionPipeline.**save**(*file\_path*, *pickle\_protocol=5*)

Saves pipeline at file path

**Parameters**

- **file\_path** (*str*) – location to save file
- **pickle\_protocol** (*int*) – the pickle data stream format.

**Returns** None**evalml.pipelines.MeanBaselineRegressionPipeline.score**MeanBaselineRegressionPipeline.**score**(*X*, *y*, *objectives*)

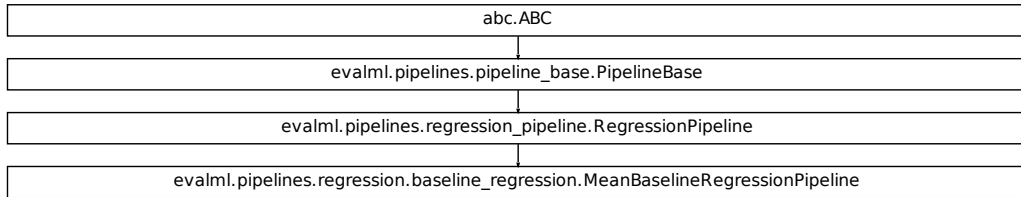
Evaluate model performance on current and additional objectives

**Parameters**

- **X** (*ww.DataTable*, *pd.DataFrame*, or *np.ndarray*) – Data of shape [n\_samples, n\_features]
- **y** (*ww.DataColumn*, *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

## Class Inheritance



### evalml.pipelines.TimeSeriesBaselineRegressionPipeline

```
class evalml.pipelines.TimeSeriesBaselineRegressionPipeline(parameters, random_seed=0)
    Baseline Pipeline for time series regression problems.
    name = 'Time Series Baseline Regression Pipeline'
    custom_name = None
    summary = 'Time Series Baseline Estimator'
    component_graph = ['Time Series Baseline Estimator']
    problem_type = 'time series regression'
    model_family = 'baseline'
    hyperparameters = {'Time Series Baseline Estimator': {}}
    custom_hyperparameters = None
    default_parameters = {'Time Series Baseline Estimator': {'gap': 1}}
```

#### Instance attributes

<code>feature_importance</code>	Return importance associated with each feature.
<code>linearized_component_graph</code>	
<code>parameters</code>	Returns parameter dictionary for this pipeline

**Methods:**

<code>__init__</code>	Machine learning pipeline for time series regression problems made out of transformers and a classifier.
<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>describe</code>	Outputs pipeline details including component parameters
<code>fit</code>	Fit a time series regression pipeline.
<code>get_component</code>	Returns component by name
<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>load</code>	Loads pipeline at file path
<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.

**evalml.pipelines.TimeSeriesBaselineRegressionPipeline.\_\_init\_\_**

`TimeSeriesBaselineRegressionPipeline.__init__(parameters, random_seed=0)`

Machine learning pipeline for time series regression problems made out of transformers and a classifier.

**Required Class Variables:** `component_graph` (list): List of components in order. Accepts strings or `ComponentBase` subclasses in the list

**Parameters**

- **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 `gap` and `max_delay` must be specified with the "pipeline" key. For example: `Pipeline(parameters={"pipeline": {"max_delay": 4, "gap": 2}})`.
- **random\_seed** (*int*) – Seed for the random number generator. Defaults to 0.

**evalml.pipelines.TimeSeriesBaselineRegressionPipeline.can\_tune\_threshold\_with\_objective**

`TimeSeriesBaselineRegressionPipeline.can_tune_threshold_with_objective(objective)`

Determine whether the threshold of a binary classification pipeline can be tuned.

**Parameters**

- **pipeline** (`PipelineBase`) – Binary classification pipeline.
- **objective** – Primary AutoMLSearch objective.

**evalml.pipelines.TimeSeriesBaselineRegressionPipeline.clone**

`TimeSeriesBaselineRegressionPipeline.clone()`

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.

**evalml.pipelines.TimeSeriesBaselineRegressionPipeline.compute\_estimator\_features**

`TimeSeriesBaselineRegressionPipeline.compute_estimator_features(X, y=None)`

Transforms the data by applying all pre-processing components.

**Parameters** **X** (`ww.DataTable`, `pd.DataFrame`) – Input data to the pipeline to transform.

**Returns** New transformed features.

**Return type** `ww.DataTable`

**evalml.pipelines.TimeSeriesBaselineRegressionPipeline.create\_objectives**

**static** `TimeSeriesBaselineRegressionPipeline.create_objectives(objectives)`

**evalml.pipelines.TimeSeriesBaselineRegressionPipeline.describe**

`TimeSeriesBaselineRegressionPipeline.describe(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`



**evalml.pipelines.TimeSeriesBaselineRegressionPipeline.fit**

`TimeSeriesBaselineRegressionPipeline.fit(X, y)`

Fit a time series regression pipeline.

**Parameters**

- **X** (*ww.DataTable*, *pd.DataFrame* or *np.ndarray*) – The input training data of shape [n\_samples, n\_features]
- **y** (*ww.DataColumn*, *pd.Series*, *np.ndarray*) – The target training targets of length [n\_samples]

**Returns** self

**evalml.pipelines.TimeSeriesBaselineRegressionPipeline.get\_component**

`TimeSeriesBaselineRegressionPipeline.get_component(name)`

Returns component by name

**Parameters** **name** (*str*) – Name of component

**Returns** Component to return

**Return type** Component

**evalml.pipelines.TimeSeriesBaselineRegressionPipeline.graph**

`TimeSeriesBaselineRegressionPipeline.graph(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

**evalml.pipelines.TimeSeriesBaselineRegressionPipeline.graph\_feature\_importance**

`TimeSeriesBaselineRegressionPipeline.graph_feature_importance(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

### `evalml.pipelines.TimeSeriesBaselineRegressionPipeline.load`

**static** `TimeSeriesBaselineRegressionPipeline.load(file_path)`

Loads pipeline at file path

**Parameters** `file_path` (*str*) – location to load file

**Returns** PipelineBase object

### `evalml.pipelines.TimeSeriesBaselineRegressionPipeline.predict`

`TimeSeriesBaselineRegressionPipeline.predict(X, y=None, objective=None)`

Make predictions using selected features.

#### **Parameters**

- **X** (*ww.DataTable*, *pd.DataFrame*, or *np.ndarray*) – Data of shape [n\_samples, n\_features]
- **y** (*ww.DataColumn*, *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** *ww.DataColumn*

### `evalml.pipelines.TimeSeriesBaselineRegressionPipeline.save`

`TimeSeriesBaselineRegressionPipeline.save(file_path, pickle_protocol=5)`

Saves pipeline at file path

#### **Parameters**

- **file\_path** (*str*) – location to save file
- **pickle\_protocol** (*int*) – the pickle data stream format.

**Returns** None

### `evalml.pipelines.TimeSeriesBaselineRegressionPipeline.score`

`TimeSeriesBaselineRegressionPipeline.score(X, y, objectives)`

Evaluate model performance on current and additional objectives.

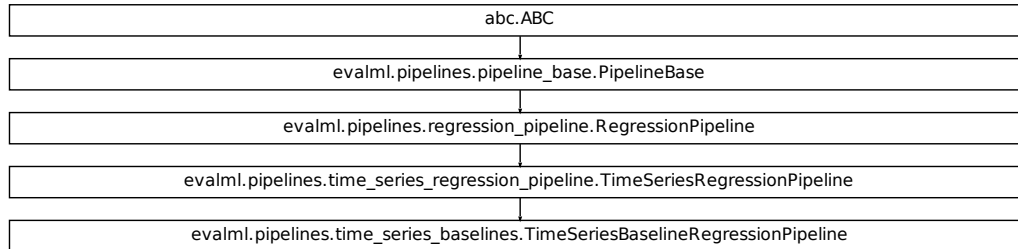
#### **Parameters**

- **X** (*ww.DataTable*, *pd.DataFrame* or *np.ndarray*) – Data of shape [n\_samples, n\_features]
- **y** (*pd.Series*, *ww.DataColumn*) – 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

## Class Inheritance



### 5.5.4 Pipeline Utils

<code>make_pipeline</code>	Given input data, target data, an estimator class and the problem type,
<code>make_pipeline_from_components</code>	Given a list of component instances and the problem type, an pipeline instance is generated with the component instances.
<code>generate_pipeline_code</code>	Creates and returns a string that contains the Python imports and code required for running the EvalML pipeline.

#### evalml.pipelines.utils.make\_pipeline

`evalml.pipelines.utils.make_pipeline(X, y, estimator, problem_type, custom_hyperparameters=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`, `ww.DataTable`) – The input data of shape [n\_samples, n\_features]
- **y** (`pd.Series`, `ww.DataColumn`) – The target data of length [n\_samples]
- **estimator** (`Estimator`) – Estimator for pipeline
- **problem\_type** (`ProblemTypes` or `str`) – Problem type for pipeline to generate
- **custom\_hyperparameters** – Dictionary of custom hyperparameters, with component name as key and dictionary of parameters as the value

## evalml.pipelines.utils.make\_pipeline\_from\_components

```
evalml.pipelines.utils.make_pipeline_from_components(component_instances, problem_type, custom_name=None,  
                                                    random_seed=0)
```

Given a list of component instances and the problem type, an pipeline instance is generated with the component instances.

The pipeline will be a subclass of the appropriate pipeline base class for the specified `problem_type`. The pipeline will be untrained, even if the input components are already trained. A custom name for the pipeline can optionally be specified; otherwise the default pipeline name will be ‘Templated Pipeline’.

### Parameters

- **component\_instances** (*list*) – a list of all of the components to include in the pipeline
- **problem\_type** (*str* or `ProblemTypes`) – problem type for the pipeline to generate
- **custom\_name** (*string*) – a name for the new pipeline
- **random\_seed** – Random seed used to initialize the pipeline.

## evalml.pipelines.utils.generate\_pipeline\_code

```
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.

## 5.6 Components

### 5.6.1 Component Base Classes

Components represent a step in a pipeline.

<code>ComponentBase</code>	Base class for all components.
<code>Transformer</code>	A component that may or may not need fitting that transforms data.
<code>Estimator</code>	A component that fits and predicts given data.

**evalml.pipelines.components.ComponentBase**

```
class evalml.pipelines.components.ComponentBase (parameters=None, component_obj=None, random_seed=0, **kwargs)
```

Base class for all components.

**Methods**

<code>__init__</code>	Initialize self.
<code>clone</code>	Constructs a new component with the same parameters and random state.
<code>describe</code>	Describe a component and its parameters
<code>fit</code>	Fits component to data
<code>load</code>	Loads component at file path
<code>save</code>	Saves component at file path

**evalml.pipelines.components.ComponentBase.\_\_init\_\_**

```
ComponentBase.__init__ (parameters=None, component_obj=None, random_seed=0, **kwargs)
```

Initialize self. See help(type(self)) for accurate signature.

**evalml.pipelines.components.ComponentBase.clone**

```
ComponentBase.clone ()
```

Constructs a new component with the same parameters and random state.

**Returns** A new instance of this component with identical parameters and random state.

**evalml.pipelines.components.ComponentBase.describe**

```
ComponentBase.describe (print_name=False, return_dict=False)
```

Describe a component and its parameters

**Parameters**

- **print\_name** (*bool, optional*) – whether to print name of component
- **return\_dict** (*bool, optional*) – whether to return description as dictionary in the format {"name": name, "parameters": parameters}

**Returns** prints and returns dictionary

**Return type** None or dict

### evalml.pipelines.components.ComponentBase.fit

`ComponentBase.fit` (*X*, *y=None*)

Fits component to data

#### Parameters

- **X** (*list*, *ww.DataTable*, *pd.DataFrame* or *np.ndarray*) – The input training data of shape [n\_samples, n\_features]
- **y** (*list*, *ww.DataColumn*, *pd.Series*, *np.ndarray*, *optional*) – The target training data of length [n\_samples]

**Returns** `self`

### evalml.pipelines.components.ComponentBase.load

**static** `ComponentBase.load` (*file\_path*)

Loads component at file path

**Parameters** **file\_path** (*str*) – Location to load file

**Returns** `ComponentBase` object

### evalml.pipelines.components.ComponentBase.save

`ComponentBase.save` (*file\_path*, *pickle\_protocol=5*)

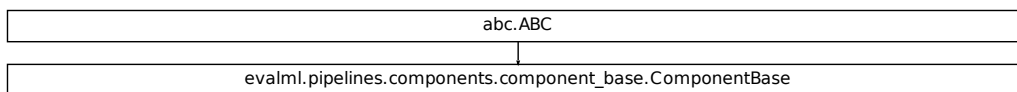
Saves component at file path

#### Parameters

- **file\_path** (*str*) – Location to save file
- **pickle\_protocol** (*int*) – The pickle data stream format.

**Returns** `None`

## Class Inheritance



## evalml.pipelines.components.Transformer

```
class evalml.pipelines.components.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.

### Methods

<code>__init__</code>	Initialize self.
<code>clone</code>	Constructs a new component with the same parameters and random state.
<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>save</code>	Saves component at file path
<code>transform</code>	Transforms data X.

### evalml.pipelines.components.Transformer.\_\_init\_\_

```
Transformer.__init__(parameters=None, component_obj=None, random_seed=0, **kwargs)
```

Initialize self. See `help(type(self))` for accurate signature.

### evalml.pipelines.components.Transformer.clone

```
Transformer.clone()
```

Constructs a new component with the same parameters and random state.

**Returns** A new instance of this component with identical parameters and random state.

### evalml.pipelines.components.Transformer.describe

```
Transformer.describe(print_name=False, return_dict=False)
```

Describe a component and its parameters

#### Parameters

- **print\_name** (*bool, optional*) – whether to print name of component
- **return\_dict** (*bool, optional*) – whether to return description as dictionary in the format {"name": name, "parameters": parameters}

**Returns** prints and returns dictionary

**Return type** None or dict

### **evalml.pipelines.components.Transformer.fit**

`Transformer.fit(X, y=None)`

Fits component to data

#### **Parameters**

- **X** (*list*, *ww.DataTable*, *pd.DataFrame* or *np.ndarray*) – The input training data of shape [n\_samples, n\_features]
- **y** (*list*, *ww.DataColumn*, *pd.Series*, *np.ndarray*, *optional*) – The target training data of length [n\_samples]

**Returns** self

### **evalml.pipelines.components.Transformer.fit\_transform**

`Transformer.fit_transform(X, y=None)`

Fits on X and transforms X

#### **Parameters**

- **X** (*ww.DataTable*, *pd.DataFrame*) – Data to fit and transform
- **y** (*ww.DataColumn*, *pd.Series*) – Target data

**Returns** Transformed X

**Return type** ww.DataTable

### **evalml.pipelines.components.Transformer.load**

**static** `Transformer.load(file_path)`

Loads component at file path

**Parameters** **file\_path** (*str*) – Location to load file

**Returns** ComponentBase object

### **evalml.pipelines.components.Transformer.save**

`Transformer.save(file_path, pickle_protocol=5)`

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.Transformer.transform

`Transformer.transform(X, y=None)`

Transforms data X.

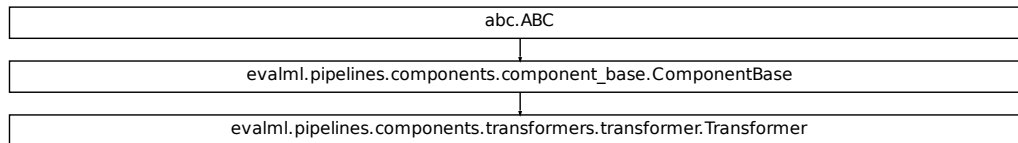
### Parameters

- **X** (`ww.DataTable`, `pd.DataFrame`) – Data to transform.
- **y** (`ww.DataColumn`, `pd.Series`, *optional*) – Target data.

**Returns** Transformed X

**Return type** `ww.DataTable`

## Class Inheritance



## evalml.pipelines.components.Estimator

**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.

### Methods

<code>__init__</code>	Initialize self.
<code>clone</code>	Constructs a new component with the same parameters and random state.
<code>describe</code>	Describe a component and its parameters
<code>fit</code>	Fits component to data
<code>load</code>	Loads component at file path
<code>predict</code>	Make predictions using selected features.

continues on next page

Table 64 – continued from previous page

<code>predict_proba</code>	Make probability estimates for labels.
<code>save</code>	Saves component at file path

### `evalml.pipelines.components.Estimator.__init__`

`Estimator.__init__` (*parameters=None, component\_obj=None, random\_seed=0, \*\*kwargs*)  
Initialize self. See help(type(self)) for accurate signature.

### `evalml.pipelines.components.Estimator.clone`

`Estimator.clone` ()  
Constructs a new component with the same parameters and random state.

**Returns** A new instance of this component with identical parameters and random state.

### `evalml.pipelines.components.Estimator.describe`

`Estimator.describe` (*print\_name=False, return\_dict=False*)  
Describe a component and its parameters

#### Parameters

- **print\_name** (*bool, optional*) – whether to print name of component
- **return\_dict** (*bool, optional*) – whether to return description as dictionary in the format {"name": name, "parameters": parameters}

**Returns** prints and returns dictionary

**Return type** None or dict

### `evalml.pipelines.components.Estimator.fit`

`Estimator.fit` (*X, y=None*)  
Fits component to data

#### Parameters

- **X** (*list, ww.DataTable, pd.DataFrame or np.ndarray*) – The input training data of shape [n\_samples, n\_features]
- **y** (*list, ww.DataColumn, pd.Series, np.ndarray, optional*) – The target training data of length [n\_samples]

**Returns** self

**evalml.pipelines.components.Estimator.load****static** Estimator.load(*file\_path*)

Loads component at file path

**Parameters** *file\_path* (*str*) – Location to load file**Returns** ComponentBase object**evalml.pipelines.components.Estimator.predict**Estimator.predict(*X*)

Make predictions using selected features.

**Parameters** *X* (*ww.DataTable*, *pd.DataFrame*, or *np.ndarray*) – Data of shape [n\_samples, n\_features]**Returns** Predicted values**Return type** ww.DataColumn**evalml.pipelines.components.Estimator.predict\_proba**Estimator.predict\_proba(*X*)

Make probability estimates for labels.

**Parameters** *X* (*ww.DataTable*, *pd.DataFrame*, or *np.ndarray*) – Features**Returns** Probability estimates**Return type** ww.DataTable**evalml.pipelines.components.Estimator.save**Estimator.save(*file\_path*, *pickle\_protocol=5*)

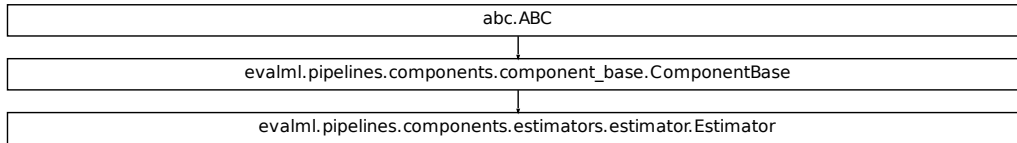
Saves component at file path

**Parameters**

- *file\_path* (*str*) – Location to save file
- *pickle\_protocol* (*int*) – The pickle data stream format.

**Returns** None

## Class Inheritance



### 5.6.2 Component Utils

<code>allowed_model_families</code>	List the model types allowed for a particular problem type.
<code>get_estimators</code>	Returns the estimators 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.

#### `evalml.pipelines.components.utils.allowed_model_families`

`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.get_estimators`

`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.generate\_component\_code

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.

### 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>OneHotEncoder</i>	One-hot encoder to encode non-numeric data.
<i>TargetEncoder</i>	Target encoder to encode categorical data
<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>	Standardize features: removes mean and scales 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 featurize DateTime columns.
<i>TextFeaturizer</i>	Transformer that can automatically featurize text 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 ww.DataTables and pd.DataFrames
<i>PolynomialDetrender</i>	Removes trends from time series by fitting a polynomial to the data.
<i>Undersampler</i>	Random undersampler component.

## evalml.pipelines.components.DropColumns

```
class evalml.pipelines.components.DropColumns (columns=None, random_seed=0,  
                                              **kwargs)
```

Drops specified columns in input data.

```
name = 'Drop Columns Transformer'
```

```
model_family = 'none'
```

```
hyperparameter_ranges = {}
```

```
default_parameters = {'columns': None}
```

### Instance attributes

---

`needs_fitting`

---

`parameters`

Returns the parameters which were used to initialize the component

---

### Methods:

---

`__init__`

Initializes an transformer that drops specified columns in input data.

---

`clone`

Constructs a new component with the same parameters and random state.

---

`describe`

Describe a component and its parameters

---

`fit`

Fits the transformer by checking if column names are present in the dataset.

---

`fit_transform`

Fits on X and transforms X

---

`load`

Loads component at file path

---

`save`

Saves component at file path

---

`transform`

Transforms data X by dropping columns.

---

### evalml.pipelines.components.DropColumns.\_\_init\_\_

```
DropColumns.__init__ (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 drop.

**evalml.pipelines.components.DropColumns.clone**`DropColumns.clone()`

Constructs a new component with the same parameters and random state.

**Returns** A new instance of this component with identical parameters and random state.

**evalml.pipelines.components.DropColumns.describe**`DropColumns.describe(print_name=False, return_dict=False)`

Describe a component and its parameters

**Parameters**

- **print\_name** (*bool, optional*) – whether to print name of component
- **return\_dict** (*bool, optional*) – whether to return description as dictionary in the format {"name": name, "parameters": parameters}

**Returns** prints and returns dictionary

**Return type** None or dict

**evalml.pipelines.components.DropColumns.fit**`DropColumns.fit(X, y=None)`

Fits the transformer by checking if column names are present in the dataset.

**Parameters**

- **X** (*ww.DataTable, pd.DataFrame*) – Data to check.
- **y** (*ww.DataColumn, pd.Series, optional*) – Targets.

**Returns** self

**evalml.pipelines.components.DropColumns.fit\_transform**`DropColumns.fit_transform(X, y=None)`

Fits on X and transforms X

**Parameters**

- **X** (*ww.DataTable, pd.DataFrame*) – Data to fit and transform
- **y** (*ww.DataColumn, pd.Series*) – Target data

**Returns** Transformed X

**Return type** ww.DataTable

### evalml.pipelines.components.DropColumns.load

**static** DropColumns.load(*file\_path*)

Loads component at file path

**Parameters** *file\_path* (*str*) – Location to load file

**Returns** ComponentBase object

### evalml.pipelines.components.DropColumns.save

DropColumns.save(*file\_path*, *pickle\_protocol*=5)

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.DropColumns.transform

DropColumns.transform(*X*, *y=None*)

Transforms data X by dropping columns.

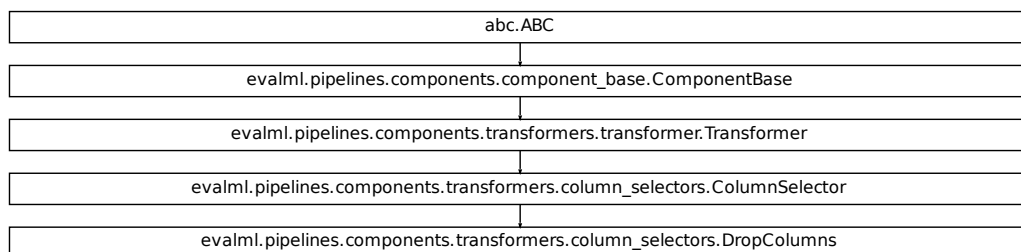
**Parameters**

- *X* (*ww.DataTable*, *pd.DataFrame*) – Data to transform.
- *y* (*ww.DataColumn*, *pd.Series*, *optional*) – Targets.

**Returns** Transformed X.

**Return type** ww.DataTable

## Class Inheritance





**evalml.pipelines.components.SelectColumns**

```
class evalml.pipelines.components.SelectColumns (columns=None, random_seed=0,  
                                              **kwargs)
```

Selects specified columns in input data.

```
name = 'Select Columns Transformer'
```

```
model_family = 'none'
```

```
hyperparameter_ranges = {}
```

```
default_parameters = {'columns': None}
```

**Instance attributes**


---

`needs_fitting`

---

<code>parameters</code>	Returns the parameters which were used to initialize the component
-------------------------	--

---

**Methods:**

<code>__init__</code>	Initializes an transformer that drops specified columns in input data.
<code>clone</code>	Constructs a new component with the same parameters and random state.
<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>save</code>	Saves component at file path
<code>transform</code>	Transforms data X by selecting columns.

**evalml.pipelines.components.SelectColumns.\_\_init\_\_**

```
SelectColumns.__init__ (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 drop.

### `evalml.pipelines.components.SelectColumns.clone`

`SelectColumns.clone()`

Constructs a new component with the same parameters and random state.

**Returns** A new instance of this component with identical parameters and random state.

### `evalml.pipelines.components.SelectColumns.describe`

`SelectColumns.describe(print_name=False, return_dict=False)`

Describe a component and its parameters

#### Parameters

- **print\_name** (*bool, optional*) – whether to print name of component
- **return\_dict** (*bool, optional*) – whether to return description as dictionary in the format {"name": name, "parameters": parameters}

**Returns** prints and returns dictionary

**Return type** None or dict

### `evalml.pipelines.components.SelectColumns.fit`

`SelectColumns.fit(X, y=None)`

Fits the transformer by checking if column names are present in the dataset.

#### Parameters

- **X** (*ww.DataTable, pd.DataFrame*) – Data to check.
- **y** (*ww.DataColumn, pd.Series, optional*) – Targets.

**Returns** self

### `evalml.pipelines.components.SelectColumns.fit_transform`

`SelectColumns.fit_transform(X, y=None)`

Fits on X and transforms X

#### Parameters

- **X** (*ww.DataTable, pd.DataFrame*) – Data to fit and transform
- **y** (*ww.DataColumn, pd.Series*) – Target data

**Returns** Transformed X

**Return type** ww.DataTable

**evalml.pipelines.components.SelectColumns.load**

**static** `SelectColumns.load(file_path)`

Loads component at file path

**Parameters** `file_path` (*str*) – Location to load file

**Returns** ComponentBase object

**evalml.pipelines.components.SelectColumns.save**

`SelectColumns.save(file_path, pickle_protocol=5)`

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.SelectColumns.transform**

`SelectColumns.transform(X, y=None)`

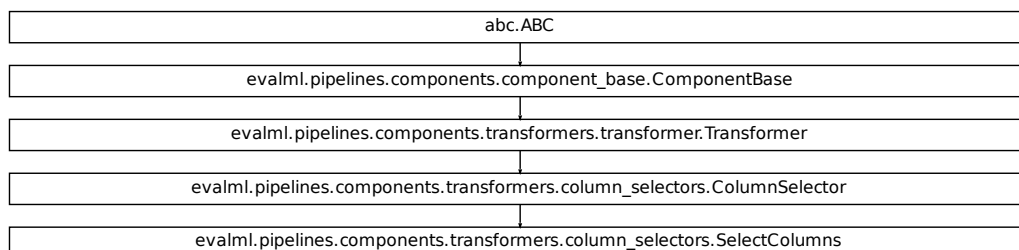
Transforms data X by selecting columns.

**Parameters**

- `X` (*ww.DataTable*, *pd.DataFrame*) – Data to transform.
- `y` (*ww.DataColumn*, *pd.Series*, *optional*) – Targets.

**Returns** Transformed X.

**Return type** ww.DataTable

**Class Inheritance**

**evalml.pipelines.components.OneHotEncoder**

```
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)
```

One-hot encoder to encode non-numeric data.

**name** = 'One Hot Encoder'

**model\_family** = 'none'

**hyperparameter\_ranges** = {}

**default\_parameters** = {'categories': None, 'drop': 'if\_binary', 'features\_to\_encode': None}

**Instance attributes**


---

`needs_fitting`

---

<code>parameters</code>	Returns the parameters which were used to initialize the component
-------------------------	--

---

**Methods:**

<code>__init__</code>	Initializes an transformer that encodes categorical features in a one-hot numeric array.”
<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>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>save</code>	Saves component at file path
<code>transform</code>	One-hot encode the input data.

**evalml.pipelines.components.OneHotEncoder.\_\_init\_\_**

```
OneHotEncoder.__init__ (top_n=10, features_to_encode=None, categories=None,  

drop='if_binary', handle_unknown='ignore', handle_missing='error',  

random_seed=0, **kwargs)
```

Initializes an 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.

### evalml.pipelines.components.OneHotEncoder.categories

`OneHotEncoder.categories(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`

### evalml.pipelines.components.OneHotEncoder.clone

`OneHotEncoder.clone()`

Constructs a new component with the same parameters and random state.

**Returns** A new instance of this component with identical parameters and random state.

### evalml.pipelines.components.OneHotEncoder.describe

`OneHotEncoder.describe(print_name=False, return_dict=False)`

Describe a component and its parameters

**Parameters**

- **print\_name** (*bool, optional*) – whether to print name of component
- **return\_dict** (*bool, optional*) – whether to return description as dictionary in the format {“*name*”: *name*, “*parameters*”: *parameters*}

**Returns** prints and returns dictionary

**Return type** None or dict

### `evalml.pipelines.components.OneHotEncoder.fit`

`OneHotEncoder.fit(X, y=None)`

Fits component to data

**Parameters**

- **X** (*list, ww.DataTable, pd.DataFrame or np.ndarray*) – The input training data of shape [n\_samples, n\_features]
- **y** (*list, ww.DataColumn, pd.Series, np.ndarray, optional*) – The target training data of length [n\_samples]

**Returns** self

### `evalml.pipelines.components.OneHotEncoder.fit_transform`

`OneHotEncoder.fit_transform(X, y=None)`

Fits on X and transforms X

**Parameters**

- **X** (*ww.DataTable, pd.DataFrame*) – Data to fit and transform
- **y** (*ww.DataColumn, pd.Series*) – Target data

**Returns** Transformed X

**Return type** ww.DataTable

### `evalml.pipelines.components.OneHotEncoder.get_feature_names`

`OneHotEncoder.get_feature_names()`

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

**evalml.pipelines.components.OneHotEncoder.load**

**static** `OneHotEncoder.load(file_path)`

Loads component at file path

**Parameters** `file_path` (*str*) – Location to load file

**Returns** ComponentBase object

**evalml.pipelines.components.OneHotEncoder.save**

`OneHotEncoder.save(file_path, pickle_protocol=5)`

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.OneHotEncoder.transform**

`OneHotEncoder.transform(X, y=None)`

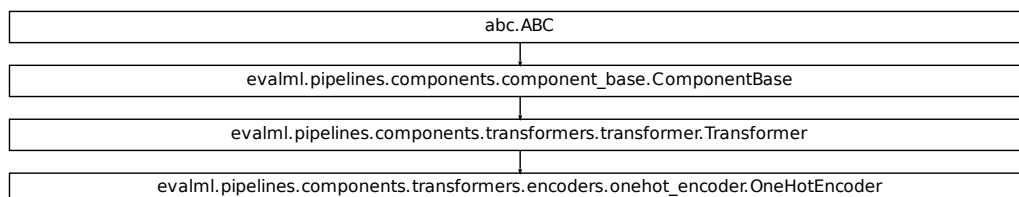
One-hot encode the input data.

**Parameters**

- `X` (*ww.DataTable*, *pd.DataFrame*) – Features to one-hot encode.
- `y` (*ww.DataColumn*, *pd.Series*) – Ignored.

**Returns** Transformed data, where each categorical feature has been encoded into numerical columns using one-hot encoding.

**Return type** *ww.DataTable*

**Class Inheritance**

**evalml.pipelines.components.TargetEncoder**

```
class evalml.pipelines.components.TargetEncoder(cols=None, smoothing=1.0,  
                                                handle_unknown='value', handle_  
                                                missing='value', random_seed=0,  
                                                **kwargs)
```

Target encoder to encode categorical data

**name** = 'Target Encoder'

**model\_family** = 'none'

**hyperparameter\_ranges** = {}

**default\_parameters** = {'cols': None, 'handle\_missing': 'value', 'handle\_unknown': 'value', 'random\_seed': 0, \*\*kwargs}

**Instance attributes**

<code>needs_fitting</code>	
<code>parameters</code>	Returns the parameters which were used to initialize the component

**Methods:**

<code>__init__</code>	Initializes a transformer that encodes categorical features into target encodings.
<code>clone</code>	Constructs a new component with the same parameters and random state.
<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.
<code>load</code>	Loads component at file path
<code>save</code>	Saves component at file path
<code>transform</code>	Transforms data X.

**evalml.pipelines.components.TargetEncoder.\_\_init\_\_**

```
TargetEncoder.__init__(cols=None, smoothing=1.0, handle_unknown='value', handle_  
                       missing='value', random_seed=0, **kwargs)
```

Initializes 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.

### evalml.pipelines.components.TargetEncoder.clone

`TargetEncoder.clone()`

Constructs a new component with the same parameters and random state.

**Returns** A new instance of this component with identical parameters and random state.

### evalml.pipelines.components.TargetEncoder.describe

`TargetEncoder.describe(print_name=False, return_dict=False)`

Describe a component and its parameters

#### Parameters

- **print\_name** (*bool, optional*) – whether to print name of component
- **return\_dict** (*bool, optional*) – whether to return description as dictionary in the format {“name”: name, “parameters”: parameters}

**Returns** prints and returns dictionary

**Return type** None or dict

### evalml.pipelines.components.TargetEncoder.fit

`TargetEncoder.fit(X, y)`

Fits component to data

#### Parameters

- **X** (*list, ww.DataTable, pd.DataFrame or np.ndarray*) – The input training data of shape [n\_samples, n\_features]
- **y** (*list, ww.DataColumn, pd.Series, np.ndarray, optional*) – The target training data of length [n\_samples]

**Returns** self

**evalml.pipelines.components.TargetEncoder.fit\_transform**

`TargetEncoder.fit_transform(X, y)`

Fits on X and transforms X

**Parameters**

- **X** (*ww.DataTable*, *pd.DataFrame*) – Data to fit and transform
- **y** (*ww.DataColumn*, *pd.Series*) – Target data

**Returns** Transformed X

**Return type** *ww.DataTable*

**evalml.pipelines.components.TargetEncoder.get\_feature\_names**

`TargetEncoder.get_feature_names()`

Return feature names for the input features after fitting.

**Returns** The feature names after encoding

**Return type** *np.array*

**evalml.pipelines.components.TargetEncoder.load**

**static** `TargetEncoder.load(file_path)`

Loads component at file path

**Parameters** **file\_path** (*str*) – Location to load file

**Returns** *ComponentBase* object

**evalml.pipelines.components.TargetEncoder.save**

`TargetEncoder.save(file_path, pickle_protocol=5)`

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.TargetEncoder.transform**

`TargetEncoder.transform(X, y=None)`

Transforms data X.

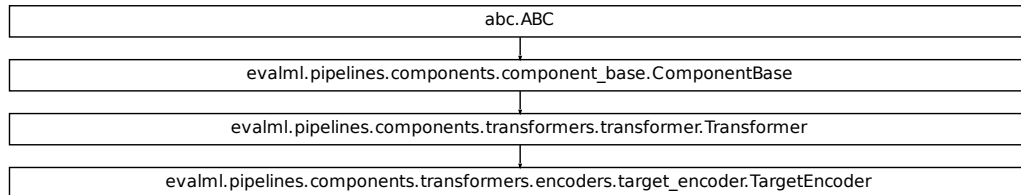
**Parameters**

- **X** (*ww.DataTable*, *pd.DataFrame*) – Data to transform.
- **y** (*ww.DataColumn*, *pd.Series*, *optional*) – Target data.

**Returns** Transformed X

**Return type** ww.DataTable

## Class Inheritance



## evalml.pipelines.components.PerColumnImputer

```
class evalml.pipelines.components.PerColumnImputer(impute_strategies=None, de-  
                                                    fault_impute_strategy='most_frequent',  
                                                    random_seed=0, **kwargs)
```

Imputes missing data according to a specified imputation strategy per column

**name** = 'Per Column Imputer'

**model\_family** = 'none'

**hyperparameter\_ranges** = {}

**default\_parameters** = {'default\_impute\_strategy': 'most\_frequent', 'impute\_strategies':

### Instance attributes

<code>needs_fitting</code>	
<code>parameters</code>	Returns the parameters which were used to initialize the component

### Methods:

<code>__init__</code>	Initializes a transformer that imputes missing data according to the specified imputation strategy per column."
<code>clone</code>	Constructs a new component with the same parameters and random state.
<code>describe</code>	Describe a component and its parameters
<code>fit</code>	Fits imputers on input data

continues on next page

Table 76 – continued from previous page

<code>fit_transform</code>	Fits on X and transforms X
<code>load</code>	Loads component at file path
<code>save</code>	Saves component at file path
<code>transform</code>	Transforms input data by imputing missing values.

### `evalml.pipelines.components.PerColumnImputer.__init__`

`PerColumnImputer.__init__` (*impute\_strategies=None*, *default\_impute\_strategy='most\_frequent'*, *random\_seed=0*, *\*\*kwargs*)

Initializes a transformer that imputes missing data according to the 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 “most\_frequent” for all columns.

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.

- **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.

### `evalml.pipelines.components.PerColumnImputer.clone`

`PerColumnImputer.clone` ()

Constructs a new component with the same parameters and random state.

**Returns** A new instance of this component with identical parameters and random state.

### `evalml.pipelines.components.PerColumnImputer.describe`

`PerColumnImputer.describe` (*print\_name=False*, *return\_dict=False*)

Describe a component and its parameters

#### Parameters

- **print\_name** (*bool*, *optional*) – whether to print name of component
- **return\_dict** (*bool*, *optional*) – whether to return description as dictionary in the format {“name”: name, “parameters”: parameters}

**Returns** prints and returns dictionary

**Return type** None or dict

**evalml.pipelines.components.PerColumnImputer.fit**`PerColumnImputer.fit(X, y=None)`

Fits imputers on input data

**Parameters**

- **X** (*ww.DataTable*, *pd.DataFrame* or *np.ndarray*) – The input training data of shape [n\_samples, n\_features] to fit.
- **y** (*ww.DataColumn*, *pd.Series*, *optional*) – The target training data of length [n\_samples]. Ignored.

**Returns** self**evalml.pipelines.components.PerColumnImputer.fit\_transform**`PerColumnImputer.fit_transform(X, y=None)`

Fits on X and transforms X

**Parameters**

- **X** (*ww.DataTable*, *pd.DataFrame*) – Data to fit and transform
- **y** (*ww.DataColumn*, *pd.Series*) – Target data

**Returns** Transformed X**Return type** ww.DataTable**evalml.pipelines.components.PerColumnImputer.load**`static PerColumnImputer.load(file_path)`

Loads component at file path

**Parameters** **file\_path** (*str*) – Location to load file**Returns** ComponentBase object**evalml.pipelines.components.PerColumnImputer.save**`PerColumnImputer.save(file_path, pickle_protocol=5)`

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.PerColumnImputer.transform

`PerColumnImputer.transform(X, y=None)`  
Transforms input data by imputing missing values.

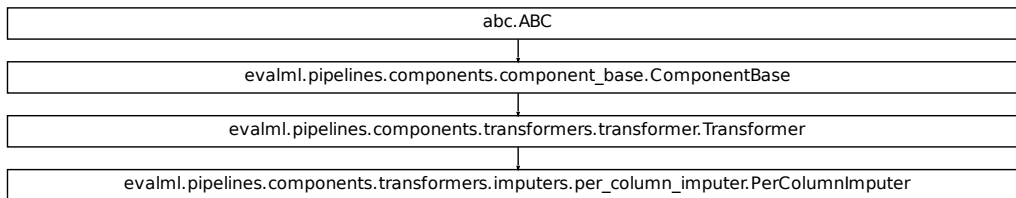
#### Parameters

- **X** (*ww.DataTable*, *pd.DataFrame* or *np.ndarray*) – The input training data of shape `[n_samples, n_features]` to transform.
- **y** (*ww.DataColumn*, *pd.Series*, *optional*) – The target training data of length `[n_samples]`. Ignored.

**Returns** Transformed X

**Return type** *ww.DataTable*

### Class Inheritance



### evalml.pipelines.components.Imputer

```
class evalml.pipelines.components.Imputer (categorical_impute_strategy='most_frequent',  
                                           categorical_fill_value=None,           nu-  
                                           meric_impute_strategy='mean',         nu-  
                                           meric_fill_value=None,           random_seed=0,  
                                           **kwargs)
```

Imputes missing data according to a specified imputation strategy.

```
name = 'Imputer'
```

```
model_family = 'none'
```

```
hyperparameter_ranges = {'categorical_impute_strategy': ['most_frequent'], 'numeric_impute_strategy': ['most_frequent']}
```

```
default_parameters = {'categorical_fill_value': None, 'categorical_impute_strategy': 'most_frequent', 'numeric_fill_value': None, 'numeric_impute_strategy': 'mean'}
```

### Instance attributes

<code>needs_fitting</code>	
<code>parameters</code>	Returns the parameters which were used to initialize the component

### Methods:

<code>__init__</code>	Initializes an transformer that imputes missing data according to the specified imputation strategy.”
<code>clone</code>	Constructs a new component with the same parameters and random state.
<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>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

### `evalml.pipelines.components.Imputer.__init__`

`Imputer.__init__` (*categorical\_impute\_strategy*='most\_frequent', *categorical\_fill\_value*=None, *numeric\_impute\_strategy*='mean', *numeric\_fill\_value*=None, *random\_seed*=0, *\*\*kwargs*)

Initializes an transformer that imputes missing data according to the 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.

### `evalml.pipelines.components.Imputer.clone`

`Imputer.clone()`

Constructs a new component with the same parameters and random state.

**Returns** A new instance of this component with identical parameters and random state.

### `evalml.pipelines.components.Imputer.describe`

`Imputer.describe(print_name=False, return_dict=False)`

Describe a component and its parameters

#### Parameters

- **print\_name** (*bool, optional*) – whether to print name of component
- **return\_dict** (*bool, optional*) – whether to return description as dictionary in the format {"name": name, "parameters": parameters}

**Returns** prints and returns dictionary

**Return type** None or dict

### `evalml.pipelines.components.Imputer.fit`

`Imputer.fit(X, y=None)`

**Fits imputer to data.** ‘None’ values are converted to `np.nan` before imputation and are treated as the same.

#### Parameters

- **X** (*ww.DataTable, pd.DataFrame or np.ndarray*) – The input training data of shape [n\_samples, n\_features]
- **y** (*ww.DataColumn, pd.Series, optional*) – The target training data of length [n\_samples]

**Returns** self

### `evalml.pipelines.components.Imputer.fit_transform`

`Imputer.fit_transform(X, y=None)`

Fits on X and transforms X

#### Parameters

- **X** (*ww.DataTable, pd.DataFrame*) – Data to fit and transform
- **y** (*ww.DataColumn, pd.Series*) – Target data

**Returns** Transformed X

**Return type** ww.DataTable



**evalml.pipelines.components.Imputer.load****static** `Imputer.load(file_path)`

Loads component at file path

**Parameters** `file_path` (*str*) – Location to load file**Returns** ComponentBase object**evalml.pipelines.components.Imputer.save**`Imputer.save(file_path, pickle_protocol=5)`

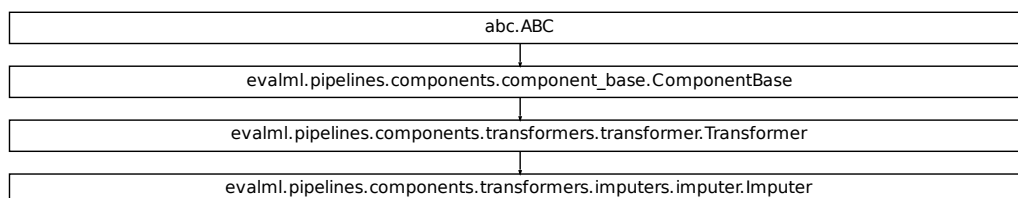
Saves component at file path

**Parameters**

- `file_path` (*str*) – Location to save file
- `pickle_protocol` (*int*) – The pickle data stream format.

**Returns** None**evalml.pipelines.components.Imputer.transform**`Imputer.transform(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` (*ww.DataTable*, *pd.DataFrame*) – Data to transform
- `y` (*ww.DataColumn*, *pd.Series*, *optional*) – Ignored.

**Returns** Transformed X**Return type** ww.DataTable**Class Inheritance**

**evalml.pipelines.components.SimpleImputer**

```
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.

    name = 'Simple Imputer'
    model_family = 'none'
    hyperparameter_ranges = {'impute_strategy': ['mean', 'median', 'most_frequent']}
    default_parameters = {'fill_value': None, 'impute_strategy': 'most_frequent'}
```

**Instance attributes**

<code>needs_fitting</code>	
<code>parameters</code>	Returns the parameters which were used to initialize the component

**Methods:**

<code>__init__</code>	Initializes an transformer that imputes missing data according to the specified imputation strategy.”
<code>clone</code>	Constructs a new component with the same parameters and random state.
<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>save</code>	Saves component at file path
<code>transform</code>	Transforms input by imputing missing values.

**evalml.pipelines.components.SimpleImputer.\_\_init\_\_**

```
SimpleImputer.__init__(impute_strategy='most_frequent', fill_value=None, random_seed=0,
                       **kwargs)
```

Initializes an transformer that imputes missing data according to the 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.

### evalml.pipelines.components.SimpleImputer.clone

`SimpleImputer.clone()`

Constructs a new component with the same parameters and random state.

**Returns** A new instance of this component with identical parameters and random state.

### evalml.pipelines.components.SimpleImputer.describe

`SimpleImputer.describe(print_name=False, return_dict=False)`

Describe a component and its parameters

#### Parameters

- **print\_name** (*bool, optional*) – whether to print name of component
- **return\_dict** (*bool, optional*) – whether to return description as dictionary in the format {"name": name, "parameters": parameters}

**Returns** prints and returns dictionary

**Return type** None or dict

### evalml.pipelines.components.SimpleImputer.fit

`SimpleImputer.fit(X, y=None)`

**Fits imputer to data.** 'None' values are converted to np.nan before imputation and are treated as the same.

#### Parameters

- **X** (*ww.DataTable, pd.DataFrame or np.ndarray*) – the input training data of shape [n\_samples, n\_features]
- **y** (*ww.DataColumn, pd.Series, optional*) – the target training data of length [n\_samples]

**Returns** self

### evalml.pipelines.components.SimpleImputer.fit\_transform

`SimpleImputer.fit_transform(X, y=None)`

Fits on X and transforms X

#### Parameters

- **X** (*ww.DataTable, pd.DataFrame*) – Data to fit and transform
- **y** (*ww.DataColumn, pd.Series, optional*) – Target data.

**Returns** Transformed X

**Return type** ww.DataTable

**evalml.pipelines.components.SimpleImputer.load****static** SimpleImputer.load(*file\_path*)

Loads component at file path

**Parameters** **file\_path** (*str*) – Location to load file**Returns** ComponentBase object**evalml.pipelines.components.SimpleImputer.save**SimpleImputer.save(*file\_path*, *pickle\_protocol=5*)

Saves component at file path

**Parameters**

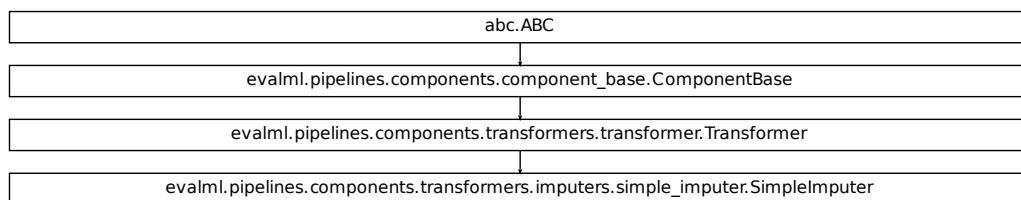
- **file\_path** (*str*) – Location to save file
- **pickle\_protocol** (*int*) – The pickle data stream format.

**Returns** None**evalml.pipelines.components.SimpleImputer.transform**SimpleImputer.transform(*X*, *y=None*)

Transforms input by imputing missing values. 'None' and np.nan values are treated as the same.

**Parameters**

- **X** (*ww.DataTable*, *pd.DataFrame*) – Data to transform
- **y** (*ww.DataColumn*, *pd.Series*, *optional*) – Ignored.

**Returns** Transformed X**Return type** ww.DataTable**Class Inheritance**

**evalml.pipelines.components.StandardScaler**

**class** evalml.pipelines.components.**StandardScaler** (*random\_seed=0, \*\*kwargs*)

Standardize features: removes mean and scales to unit variance.

**name** = 'Standard Scaler'

**model\_family** = 'none'

**hyperparameter\_ranges** = {}

**default\_parameters** = {}

**Instance attributes**


---

`needs_fitting`

---

`parameters`

Returns the parameters which were used to initialize the component

---

**Methods:**


---

`__init__`

Initialize self.

---

`clone`

Constructs a new component with the same parameters and random state.

---

`describe`

Describe a component and its parameters

---

`fit`

Fits component to data

---

`fit_transform`

Fits on X and transforms X

---

`load`

Loads component at file path

---

`save`

Saves component at file path

---

`transform`

Transforms data X.

---

**evalml.pipelines.components.StandardScaler.\_\_init\_\_**

**StandardScaler.\_\_init\_\_** (*random\_seed=0, \*\*kwargs*)

Initialize self. See help(type(self)) for accurate signature.

### `evalml.pipelines.components.StandardScaler.clone`

`StandardScaler.clone()`

Constructs a new component with the same parameters and random state.

**Returns** A new instance of this component with identical parameters and random state.

### `evalml.pipelines.components.StandardScaler.describe`

`StandardScaler.describe(print_name=False, return_dict=False)`

Describe a component and its parameters

#### Parameters

- **print\_name** (*bool, optional*) – whether to print name of component
- **return\_dict** (*bool, optional*) – whether to return description as dictionary in the format {"name": name, "parameters": parameters}

**Returns** prints and returns dictionary

**Return type** None or dict

### `evalml.pipelines.components.StandardScaler.fit`

`StandardScaler.fit(X, y=None)`

Fits component to data

#### Parameters

- **X** (*list, ww.DataTable, pd.DataFrame or np.ndarray*) – The input training data of shape [n\_samples, n\_features]
- **y** (*list, ww.DataColumn, pd.Series, np.ndarray, optional*) – The target training data of length [n\_samples]

**Returns** self

### `evalml.pipelines.components.StandardScaler.fit_transform`

`StandardScaler.fit_transform(X, y=None)`

Fits on X and transforms X

#### Parameters

- **X** (*ww.DataTable, pd.DataFrame*) – Data to fit and transform
- **y** (*ww.DataColumn, pd.Series*) – Target data

**Returns** Transformed X

**Return type** ww.DataTable

**evalml.pipelines.components.StandardScaler.load**

**static** `StandardScaler.load(file_path)`

Loads component at file path

**Parameters** `file_path` (*str*) – Location to load file

**Returns** ComponentBase object

**evalml.pipelines.components.StandardScaler.save**

`StandardScaler.save(file_path, pickle_protocol=5)`

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.StandardScaler.transform**

`StandardScaler.transform(X, y=None)`

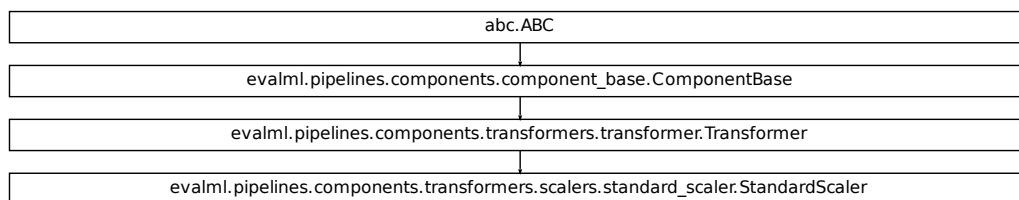
Transforms data X.

**Parameters**

- `X` (*ww.DataTable*, *pd.DataFrame*) – Data to transform.
- `y` (*ww.DataColumn*, *pd.Series*, *optional*) – Target data.

**Returns** Transformed X

**Return type** *ww.DataTable*

**Class Inheritance**

**evalml.pipelines.components.RFRegressorSelectFromModel**

```

class evalml.pipelines.components.RFRegressorSelectFromModel (number_features=None,
                                                              n_estimators=10,
                                                              max_depth=None,
                                                              per-
cent_features=0.5,
                                                              threshold=-    inf,
                                                              n_jobs=- 1,  ran-
dom_seed=0,
                                                              **kwargs)

```

Selects top features based on importance weights using a Random Forest regressor.

**name** = 'RF Regressor Select From Model'

**model\_family** = 'none'

**hyperparameter\_ranges** = {'percent\_features': Real(low=0.01, high=1, prior='uniform', t

**default\_parameters** = {'max\_depth': None, 'n\_estimators': 10, 'n\_jobs': -1, 'number\_fea

**Instance attributes**

<code>needs_fitting</code>	
<code>parameters</code>	Returns the parameters which were used to initialize the component

**Methods:**

<code>__init__</code>	Initialize self.
<code>clone</code>	Constructs a new component with the same parameters and random state.
<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>save</code>	Saves component at file path
<code>transform</code>	Transforms input data by selecting features.



**evalml.pipelines.components.RFRegressorSelectFromModel.\_\_init\_\_**

```
RFRegressorSelectFromModel.__init__(number_features=None, n_estimators=10,
                                     max_depth=None, percent_features=0.5,
                                     threshold=-inf, n_jobs=-1, random_seed=0,
                                     **kwargs)
```

Initialize self. See help(type(self)) for accurate signature.

**evalml.pipelines.components.RFRegressorSelectFromModel.clone**

```
RFRegressorSelectFromModel.clone()
```

Constructs a new component with the same parameters and random state.

**Returns** A new instance of this component with identical parameters and random state.

**evalml.pipelines.components.RFRegressorSelectFromModel.describe**

```
RFRegressorSelectFromModel.describe(print_name=False, return_dict=False)
```

Describe a component and its parameters

**Parameters**

- **print\_name** (*bool, optional*) – whether to print name of component
- **return\_dict** (*bool, optional*) – whether to return description as dictionary in the format {"name": name, "parameters": parameters}

**Returns** prints and returns dictionary

**Return type** None or dict

**evalml.pipelines.components.RFRegressorSelectFromModel.fit**

```
RFRegressorSelectFromModel.fit(X, y=None)
```

Fits component to data

**Parameters**

- **X** (*list, ww.DataTable, pd.DataFrame or np.ndarray*) – The input training data of shape [n\_samples, n\_features]
- **y** (*list, ww.DataColumn, pd.Series, np.ndarray, optional*) – The target training data of length [n\_samples]

**Returns** self

**evalml.pipelines.components.RFRegressorSelectFromModel.fit\_transform**

`RFRegressorSelectFromModel.fit_transform(X, y=None)`

Fits on X and transforms X

**Parameters**

- **X** (*ww.DataTable*, *pd.DataFrame*) – Data to fit and transform
- **y** (*ww.DataColumn*, *pd.Series*) – Target data

**Returns** Transformed X

**Return type** *ww.DataTable*

**evalml.pipelines.components.RFRegressorSelectFromModel.get\_names**

`RFRegressorSelectFromModel.get_names()`

Get names of selected features.

**Returns** List of the names of features selected

**Return type** *list[str]*

**evalml.pipelines.components.RFRegressorSelectFromModel.load**

**static** `RFRegressorSelectFromModel.load(file_path)`

Loads component at file path

**Parameters** **file\_path** (*str*) – Location to load file

**Returns** *ComponentBase* object

**evalml.pipelines.components.RFRegressorSelectFromModel.save**

`RFRegressorSelectFromModel.save(file_path, pickle_protocol=5)`

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.RFRegressorSelectFromModel.transform**

`RFRegressorSelectFromModel.transform(X, y=None)`

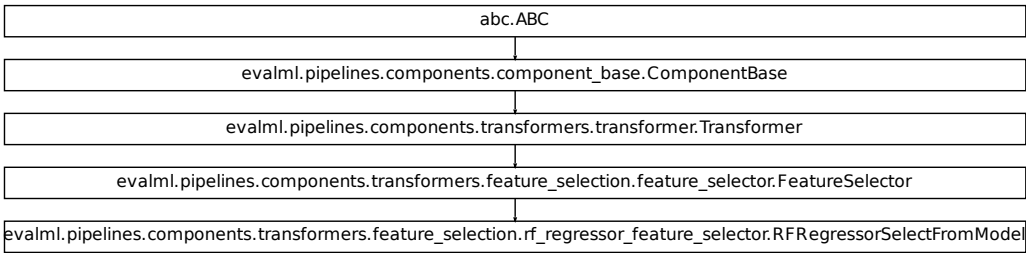
Transforms input data by selecting features. If the component\_obj does not have a transform method, will raise an *MethodPropertyNotFoundError* exception.

**Parameters**

- **X** (*ww.DataTable*, *pd.DataFrame*) – Data to transform.
- **y** (*ww.DataColumn*, *pd.Series*, *optional*) – Target data. Ignored.

**Returns** Transformed X  
**Return type** ww.DataTable

Class Inheritance



evalml.pipelines.components.RFClassifierSelectFromModel

```
class evalml.pipelines.components.RFClassifierSelectFromModel (number_features=None,
                                                                n_estimators=10,
                                                                max_depth=None,
                                                                per-
                                                                cent_features=0.5,
                                                                threshold=- inf,
                                                                n_jobs=- 1, ran-
                                                                dom_seed=0,
                                                                **kwargs)

Selects top features based on importance weights using a Random Forest classifier.

name = 'RF Classifier Select From Model'
model_family = 'none'
hyperparameter_ranges = {'percent_features': Real(low=0.01, high=1, prior='uniform', t
default_parameters = {'max_depth': None, 'n_estimators': 10, 'n_jobs': -1, 'number_fea
```

Instance attributes

needs_fitting	
parameters	Returns the parameters which were used to initialize the component

**Methods:**

<code>__init__</code>	Initialize self.
<code>clone</code>	Constructs a new component with the same parameters and random state.
<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>save</code>	Saves component at file path
<code>transform</code>	Transforms input data by selecting features.

**evalml.pipelines.components.RFClassifierSelectFromModel.\_\_init\_\_**

```
RFClassifierSelectFromModel.__init__(number_features=None, n_estimators=10,
                                     max_depth=None, percent_features=0.5,
                                     threshold=-inf, n_jobs=-1, random_seed=0,
                                     **kwargs)
```

Initialize self. See help(type(self)) for accurate signature.

**evalml.pipelines.components.RFClassifierSelectFromModel.clone**

```
RFClassifierSelectFromModel.clone()
```

Constructs a new component with the same parameters and random state.

**Returns** A new instance of this component with identical parameters and random state.

**evalml.pipelines.components.RFClassifierSelectFromModel.describe**

```
RFClassifierSelectFromModel.describe(print_name=False, return_dict=False)
```

Describe a component and its parameters

**Parameters**

- **print\_name** (*bool, optional*) – whether to print name of component
- **return\_dict** (*bool, optional*) – whether to return description as dictionary in the format {"name": name, "parameters": parameters}

**Returns** prints and returns dictionary

**Return type** None or dict

**evalml.pipelines.components.RFClassifierSelectFromModel.fit**

`RFClassifierSelectFromModel.fit(X, y=None)`

Fits component to data

**Parameters**

- **X** (*list*, *ww.DataTable*, *pd.DataFrame* or *np.ndarray*) – The input training data of shape [n\_samples, n\_features]
- **y** (*list*, *ww.DataColumn*, *pd.Series*, *np.ndarray*, *optional*) – The target training data of length [n\_samples]

**Returns** self

**evalml.pipelines.components.RFClassifierSelectFromModel.fit\_transform**

`RFClassifierSelectFromModel.fit_transform(X, y=None)`

Fits on X and transforms X

**Parameters**

- **X** (*ww.DataTable*, *pd.DataFrame*) – Data to fit and transform
- **y** (*ww.DataColumn*, *pd.Series*) – Target data

**Returns** Transformed X

**Return type** *ww.DataTable*

**evalml.pipelines.components.RFClassifierSelectFromModel.get\_names**

`RFClassifierSelectFromModel.get_names()`

Get names of selected features.

**Returns** List of the names of features selected

**Return type** *list[str]*

**evalml.pipelines.components.RFClassifierSelectFromModel.load**

**static** `RFClassifierSelectFromModel.load(file_path)`

Loads component at file path

**Parameters** **file\_path** (*str*) – Location to load file

**Returns** *ComponentBase* object

**evalml.pipelines.components.RFClassifierSelectFromModel.save**

`RFClassifierSelectFromModel.save(file_path, pickle_protocol=5)`

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.RFClassifierSelectFromModel.transform**

`RFClassifierSelectFromModel.transform(X, y=None)`

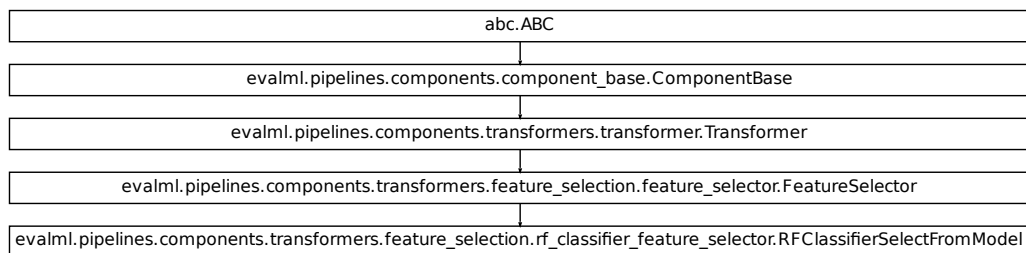
Transforms input data by selecting features. If the `component_obj` does not have a transform method, will raise an `MethodPropertyNotFoundError` exception.

**Parameters**

- **X** (*ww.DataTable*, *pd.DataFrame*) – Data to transform.
- **y** (*ww.DataColumn*, *pd.Series*, *optional*) – Target data. Ignored.

**Returns** Transformed X

**Return type** `ww.DataTable`

**Class Inheritance**

**evalml.pipelines.components.DropNullColumns**

```
class evalml.pipelines.components.DropNullColumns (pct_null_threshold=1.0,          ran-
                                                    dom_seed=0, **kwargs)
    Transformer to drop features whose percentage of NaN values exceeds a specified threshold

    name = 'Drop Null Columns Transformer'
    model_family = 'none'
    hyperparameter_ranges = {}
    default_parameters = {'pct_null_threshold': 1.0}
```

**Instance attributes**

<code>needs_fitting</code>	
<code>parameters</code>	Returns the parameters which were used to initialize the component

**Methods:**

<code>__init__</code>	Initializes an transformer to drop features whose percentage of NaN values exceeds a specified threshold.
<code>clone</code>	Constructs a new component with the same parameters and random state.
<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>save</code>	Saves component at file path
<code>transform</code>	Transforms data X by dropping columns that exceed the threshold of null values.

**evalml.pipelines.components.DropNullColumns.\_\_init\_\_**

`DropNullColumns.__init__` (*pct\_null\_threshold=1.0, random\_seed=0, \*\*kwargs*)  
 Initializes an 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.

**evalml.pipelines.components.DropNullColumns.clone**

`DropNullColumns.clone()`

Constructs a new component with the same parameters and random state.

**Returns** A new instance of this component with identical parameters and random state.

**evalml.pipelines.components.DropNullColumns.describe**

`DropNullColumns.describe(print_name=False, return_dict=False)`

Describe a component and its parameters

**Parameters**

- **print\_name** (*bool, optional*) – whether to print name of component
- **return\_dict** (*bool, optional*) – whether to return description as dictionary in the format {"name": name, "parameters": parameters}

**Returns** prints and returns dictionary

**Return type** None or dict

**evalml.pipelines.components.DropNullColumns.fit**

`DropNullColumns.fit(X, y=None)`

Fits component to data

**Parameters**

- **X** (*list, ww.DataTable, pd.DataFrame or np.ndarray*) – The input training data of shape [n\_samples, n\_features]
- **y** (*list, ww.DataColumn, pd.Series, np.ndarray, optional*) – The target training data of length [n\_samples]

**Returns** self

**evalml.pipelines.components.DropNullColumns.fit\_transform**

`DropNullColumns.fit_transform(X, y=None)`

Fits on X and transforms X

**Parameters**

- **X** (*ww.DataTable, pd.DataFrame*) – Data to fit and transform
- **y** (*ww.DataColumn, pd.Series*) – Target data

**Returns** Transformed X

**Return type** ww.DataTable



**evalml.pipelines.components.DropNullColumns.load**

**static** DropNullColumns.**load**(*file\_path*)

Loads component at file path

**Parameters** **file\_path** (*str*) – Location to load file

**Returns** ComponentBase object

**evalml.pipelines.components.DropNullColumns.save**

DropNullColumns.**save**(*file\_path*, *pickle\_protocol*=5)

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.DropNullColumns.transform**

DropNullColumns.**transform**(*X*, *y=None*)

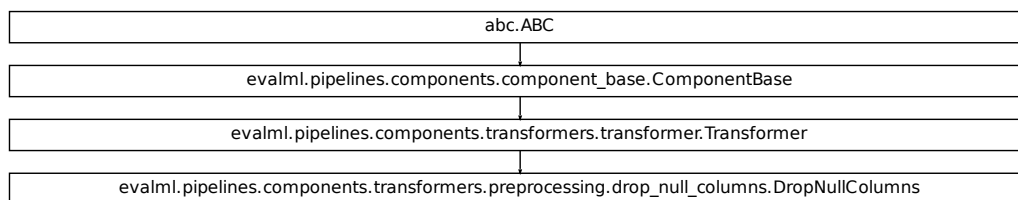
Transforms data X by dropping columns that exceed the threshold of null values.

**Parameters**

- **X** (*ww.DataTable*, *pd.DataFrame*) – Data to transform
- **y** (*ww.DataColumn*, *pd.Series*, *optional*) – Ignored.

**Returns** Transformed X

**Return type** ww.DataTable

**Class Inheritance**

## evalml.pipelines.components.DateTimeFeaturizer

```
class evalml.pipelines.components.DateTimeFeaturizer(features_to_extract=None,  
encode_as_categories=False,  
random_seed=0, **kwargs)
```

Transformer that can automatically featurize DateTime columns.

```
name = 'DateTime Featurization Component'
```

```
model_family = 'none'
```

```
hyperparameter_ranges = {}
```

```
default_parameters = {'encode_as_categories': False, 'features_to_extract': ['year', 'month', 'day_of_week', 'hour']}
```

### Instance attributes

<code>needs_fitting</code>	
<code>parameters</code>	Returns the parameters which were used to initialize the component

### Methods:

<code>__init__</code>	Extracts features from DateTime columns
<code>clone</code>	Constructs a new component with the same parameters and random state.
<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>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

## evalml.pipelines.components.DateTimeFeaturizer.\_\_init\_\_

```
DateTimeFeaturizer.__init__(features_to_extract=None, encode_as_categories=False, random_seed=0, **kwargs)
```

Extracts features from DateTime columns

### Parameters

- **features\_to\_extract** (*list*) – List of features to extract. Valid options include “year”, “month”, “day\_of\_week”, “hour”.
- **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.
- **random\_seed** (*int*) – Seed for the random number generator. Defaults to 0.

**evalml.pipelines.components.DateTimeFeaturizer.clone**`DateTimeFeaturizer.clone()`

Constructs a new component with the same parameters and random state.

**Returns** A new instance of this component with identical parameters and random state.

**evalml.pipelines.components.DateTimeFeaturizer.describe**`DateTimeFeaturizer.describe(print_name=False, return_dict=False)`

Describe a component and its parameters

**Parameters**

- **print\_name** (*bool, optional*) – whether to print name of component
- **return\_dict** (*bool, optional*) – whether to return description as dictionary in the format {"name": name, "parameters": parameters}

**Returns** prints and returns dictionary

**Return type** None or dict

**evalml.pipelines.components.DateTimeFeaturizer.fit**`DateTimeFeaturizer.fit(X, y=None)`

Fits component to data

**Parameters**

- **X** (*list, ww.DataTable, pd.DataFrame or np.ndarray*) – The input training data of shape [n\_samples, n\_features]
- **y** (*list, ww.DataColumn, pd.Series, np.ndarray, optional*) – The target training data of length [n\_samples]

**Returns** self

**evalml.pipelines.components.DateTimeFeaturizer.fit\_transform**`DateTimeFeaturizer.fit_transform(X, y=None)`

Fits on X and transforms X

**Parameters**

- **X** (*ww.DataTable, pd.DataFrame*) – Data to fit and transform
- **y** (*ww.DataColumn, pd.Series*) – Target data

**Returns** Transformed X

**Return type** ww.DataTable

**evalml.pipelines.components.DateTimeFeaturizer.get\_feature\_names**

`DateTimeFeaturizer.get_feature_names()`

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.

**evalml.pipelines.components.DateTimeFeaturizer.load**

**static** `DateTimeFeaturizer.load(file_path)`

Loads component at file path

**Parameters** `file_path` (*str*) – Location to load file

**Returns** ComponentBase object

**evalml.pipelines.components.DateTimeFeaturizer.save**

`DateTimeFeaturizer.save(file_path, pickle_protocol=5)`

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.DateTimeFeaturizer.transform**

`DateTimeFeaturizer.transform(X, y=None)`

Transforms data X by creating new features using existing DateTime columns, and then dropping those DateTime columns

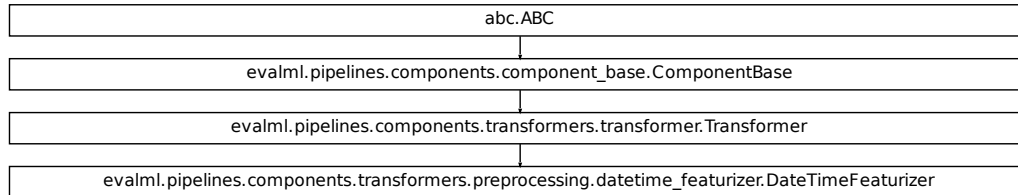
**Parameters**

- `X` (*ww.DataTable*, *pd.DataFrame*) – Data to transform
- `y` (*ww.DataColumn*, *pd.Series*, *optional*) – Ignored.

**Returns** Transformed X

**Return type** *ww.DataTable*

## Class Inheritance



### evalml.pipelines.components.TextFeaturizer

**class** evalml.pipelines.components.**TextFeaturizer** (*random\_seed=0, \*\*kwargs*)

Transformer that can automatically featurize text columns.

**name** = 'Text Featurization Component'

**model\_family** = 'none'

**hyperparameter\_ranges** = {}

**default\_parameters** = {}

#### Instance attributes

<code>needs_fitting</code>	
<code>parameters</code>	Returns the parameters which were used to initialize the component

#### Methods:

<code>__init__</code>	Extracts features from text columns using feature-tools' <code>nlp_primitives</code>
<code>clone</code>	Constructs a new component with the same parameters and random state.
<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>save</code>	Saves component at file path
<code>transform</code>	Transforms data X by creating new features using existing text columns

**evalml.pipelines.components.TextFeaturizer.\_\_init\_\_**

`TextFeaturizer.__init__(random_seed=0, **kwargs)`

Extracts features from text columns using featurer tools' nlp\_primitives

**Parameters** `random_seed` (*int*) – Seed for the random number generator. Defaults to 0.

**evalml.pipelines.components.TextFeaturizer.clone**

`TextFeaturizer.clone()`

Constructs a new component with the same parameters and random state.

**Returns** A new instance of this component with identical parameters and random state.

**evalml.pipelines.components.TextFeaturizer.describe**

`TextFeaturizer.describe(print_name=False, return_dict=False)`

Describe a component and its parameters

**Parameters**

- **print\_name** (*bool*, *optional*) – whether to print name of component
- **return\_dict** (*bool*, *optional*) – whether to return description as dictionary in the format {"name": name, "parameters": parameters}

**Returns** prints and returns dictionary

**Return type** None or dict

**evalml.pipelines.components.TextFeaturizer.fit**

`TextFeaturizer.fit(X, y=None)`

Fits component to data

**Parameters**

- **X** (*ww.DataTable*, *pd.DataFrame* or *np.ndarray*) – The input training data of shape [n\_samples, n\_features]
- **y** (*ww.DataColumn*, *pd.Series*, *np.ndarray*, *optional*) – The target training data of length [n\_samples]

**Returns** self

**evalml.pipelines.components.TextFeaturizer.fit\_transform**

`TextFeaturizer.fit_transform(X, y=None)`

Fits on X and transforms X

**Parameters**

- **X** (*ww.DataTable*, *pd.DataFrame*) – Data to fit and transform
- **y** (*ww.DataColumn*, *pd.Series*) – Target data

**Returns** Transformed X

**Return type** ww.DataTable

### evalml.pipelines.components.TextFeaturizer.load

**static** TextFeaturizer.load(*file\_path*)

Loads component at file path

**Parameters** *file\_path* (*str*) – Location to load file

**Returns** ComponentBase object

### evalml.pipelines.components.TextFeaturizer.save

TextFeaturizer.save(*file\_path*, *pickle\_protocol*=5)

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.TextFeaturizer.transform

TextFeaturizer.transform(*X*, *y*=None)

Transforms data X by creating new features using existing text columns

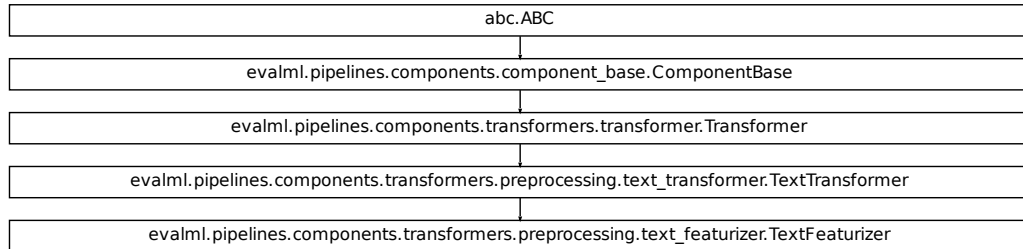
**Parameters**

- *X* (ww.DataTable, pd.DataFrame) – The data to transform.
- *y* (ww.DataColumn, pd.Series, optional) – Ignored.

**Returns** Transformed X

**Return type** ww.DataTable

## Class Inheritance



### evalml.pipelines.components.DelayedFeatureTransformer

```

class evalml.pipelines.components.DelayedFeatureTransformer(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.

```
name = 'Delayed Feature Transformer'
```

```
model_family = 'none'
```

```
hyperparameter_ranges = {}
```

```
default_parameters = {'delay_features': True, 'delay_target': True, 'gap': 1, 'max_del.
```

#### Instance attributes

needs_fitting	
parameters	Returns the parameters which were used to initialize the component



**Methods:**

<code>__init__</code>	Creates a DelayedFeatureTransformer.
<code>clone</code>	Constructs a new component with the same parameters and random state.
<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>save</code>	Saves component at file path
<code>transform</code>	Computes the delayed features for all features in X and y.

**evalml.pipelines.components.DelayedFeatureTransformer.\_\_init\_\_**

`DelayedFeatureTransformer.__init__(max_delay=2, delay_features=True, delay_target=True, gap=1, random_seed=0, **kwargs)`

Creates a DelayedFeatureTransformer.

**Parameters**

- **max\_delay** (*int*) – Maximum number of time units to delay each feature.
- **delay\_features** (*bool*) – Whether to delay the input features.
- **delay\_target** (*bool*) – Whether to delay the target.
- **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.
- **random\_seed** (*int*) – Seed for the random number generator. This transformer performs the same regardless of the random seed provided.

**evalml.pipelines.components.DelayedFeatureTransformer.clone**

`DelayedFeatureTransformer.clone()`

Constructs a new component with the same parameters and random state.

**Returns** A new instance of this component with identical parameters and random state.

**evalml.pipelines.components.DelayedFeatureTransformer.describe**

`DelayedFeatureTransformer.describe(print_name=False, return_dict=False)`

Describe a component and its parameters

**Parameters**

- **print\_name** (*bool, optional*) – whether to print name of component
- **return\_dict** (*bool, optional*) – whether to return description as dictionary in the format {"name": name, "parameters": parameters}

**Returns** prints and returns dictionary

**Return type** None or dict

### **evalml.pipelines.components.DelayedFeatureTransformer.fit**

`DelayedFeatureTransformer.fit(X, y=None)`

Fits the DelayFeatureTransformer.

#### **Parameters**

- **X** (*ww.DataTable*, *pd.DataFrame* or *np.ndarray*) – The input training data of shape [n\_samples, n\_features]
- **y** (*ww.DataColumn*, *pd.Series*, *optional*) – The target training data of length [n\_samples]

**Returns** self

### **evalml.pipelines.components.DelayedFeatureTransformer.fit\_transform**

`DelayedFeatureTransformer.fit_transform(X, y)`

Fits on X and transforms X

#### **Parameters**

- **X** (*ww.DataTable*, *pd.DataFrame*) – Data to fit and transform
- **y** (*ww.DataColumn*, *pd.Series*) – Target data

**Returns** Transformed X

**Return type** ww.DataTable

### **evalml.pipelines.components.DelayedFeatureTransformer.load**

**static** `DelayedFeatureTransformer.load(file_path)`

Loads component at file path

**Parameters** **file\_path** (*str*) – Location to load file

**Returns** ComponentBase object

### **evalml.pipelines.components.DelayedFeatureTransformer.save**

`DelayedFeatureTransformer.save(file_path, pickle_protocol=5)`

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.DelayedFeatureTransformer.transform**

`DelayedFeatureTransformer.transform(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

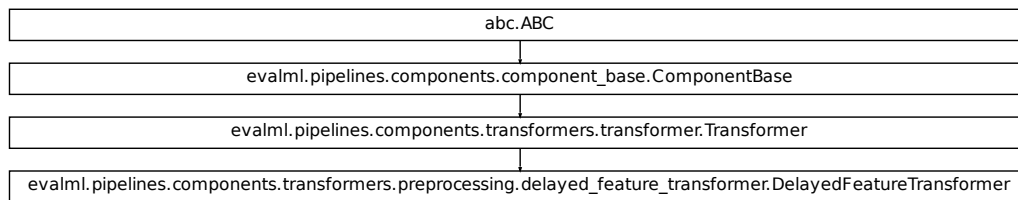
If y is not None, it will also compute the delayed values for the target variable.

**Parameters**

- **X** (*ww.DataTable*, *pd.DataFrame* or *None*) – Data to transform. None is expected when only the target variable is being used.
- **y** (*ww.DataColumn*, *pd.Series*, or *None*) – Target.

**Returns** Transformed X.

**Return type** *ww.DataTable*

**Class Inheritance****evalml.pipelines.components.DFSTransformer**

```
class evalml.pipelines.components.DFSTransformer (index='index',      random_seed=0,
                                                **kwargs)
```

Featuretools DFS component that generates features for *ww.DataTables* and *pd.DataFrames*

```
name = 'DFS Transformer'
```

```
model_family = 'none'
```

```
hyperparameter_ranges = {}
```

```
default_parameters = {'index': 'index'}
```

### Instance attributes

<code>needs_fitting</code>	
<code>parameters</code>	Returns the parameters which were used to initialize the component

### Methods:

<code>__init__</code>	Allows for featuretools to be used in EvalML.
<code>clone</code>	Constructs a new component with the same parameters and random state.
<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>save</code>	Saves component at file path
<code>transform</code>	Computes the feature matrix for the input X using featuretools' dfs algorithm.

### `evalml.pipelines.components.DFSTransformer.__init__`

`DFSTransformer.__init__(index='index', random_seed=0, **kwargs)`

Allows for featuretools to be used in EvalML.

#### 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

### `evalml.pipelines.components.DFSTransformer.clone`

`DFSTransformer.clone()`

Constructs a new component with the same parameters and random state.

**Returns** A new instance of this component with identical parameters and random state.

### `evalml.pipelines.components.DFSTransformer.describe`

`DFSTransformer.describe(print_name=False, return_dict=False)`

Describe a component and its parameters

#### Parameters

- **print\_name** (*bool, optional*) – whether to print name of component
- **return\_dict** (*bool, optional*) – whether to return description as dictionary in the format {"name": name, "parameters": parameters}

**Returns** prints and returns dictionary

**Return type** None or dict

### `evalml.pipelines.components.DFSTransformer.fit`

`DFSTransformer.fit(X, y=None)`

Fits the DFSTransformer Transformer component.

#### **Parameters**

- **X** (*ww.DataTable*, *pd.DataFrame*, *np.array*) – The input data to transform, of shape [n\_samples, n\_features]
- **y** (*ww.DataColumn*, *pd.Series*, *np.ndarray*, *optional*) – The target training data of length [n\_samples]

**Returns** self

### `evalml.pipelines.components.DFSTransformer.fit_transform`

`DFSTransformer.fit_transform(X, y=None)`

Fits on X and transforms X

#### **Parameters**

- **X** (*ww.DataTable*, *pd.DataFrame*) – Data to fit and transform
- **y** (*ww.DataColumn*, *pd.Series*) – Target data

**Returns** Transformed X

**Return type** *ww.DataTable*

### `evalml.pipelines.components.DFSTransformer.load`

**static** `DFSTransformer.load(file_path)`

Loads component at file path

**Parameters** **file\_path** (*str*) – Location to load file

**Returns** ComponentBase object

### `evalml.pipelines.components.DFSTransformer.save`

`DFSTransformer.save(file_path, pickle_protocol=5)`

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.DFSTransformer.transform

`DFSTransformer.transform(X, y=None)`

Computes the feature matrix for the input X using featuretools' dfs algorithm.

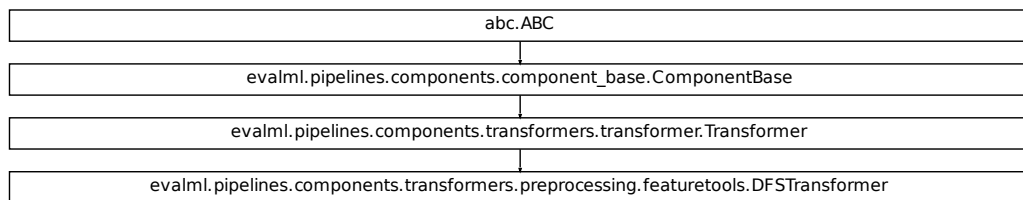
#### Parameters

- **X** (*ww.DataTable*, *pd.DataFrame* or *np.ndarray*) – The input training data to transform. Has shape [n\_samples, n\_features]
- **y** (*ww.DataColumn*, *pd.Series*, *optional*) – Ignored.

**Returns** Feature matrix

**Return type** *ww.DataTable*

### Class Inheritance



### evalml.pipelines.components.PolynomialDetrender

**class** `evalml.pipelines.components.PolynomialDetrender` (*degree=1*, *random\_seed=0*,  
\*\**kwargs*)

Removes trends from time series by fitting a polynomial to the data.

**name** = 'Polynomial Detrender'

**model\_family** = 'none'

**hyperparameter\_ranges** = {'degree': Integer(low=1, high=3, prior='uniform', transform='')

**default\_parameters** = {'degree': 1}

## Instance attributes

<code>needs_fitting</code>	
<code>parameters</code>	Returns the parameters which were used to initialize the component

## Methods:

<code>__init__</code>	Initialize the PolynomialDetrender.
<code>clone</code>	Constructs a new component with the same parameters and random state.
<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>save</code>	Saves component at file path
<code>transform</code>	Removes fitted trend from target variable.

## `evalml.pipelines.components.PolynomialDetrender.__init__`

`PolynomialDetrender.__init__(degree=1, random_seed=0, **kwargs)`  
Initialize the PolynomialDetrender.

### Parameters

- **degree** (*int*) – Degree for the polynomial. If 1, linear model is fit to the data. If 2, quadratic model is fit, etc. Default of 1.
- **random\_seed** (*int*) – Seed for the random number generator. Defaults to 0.

## `evalml.pipelines.components.PolynomialDetrender.clone`

`PolynomialDetrender.clone()`  
Constructs a new component with the same parameters and random state.

**Returns** A new instance of this component with identical parameters and random state.

## `evalml.pipelines.components.PolynomialDetrender.describe`

`PolynomialDetrender.describe(print_name=False, return_dict=False)`  
Describe a component and its parameters

### Parameters

- **print\_name** (*bool, optional*) – whether to print name of component
- **return\_dict** (*bool, optional*) – whether to return description as dictionary in the format {"name": name, "parameters": parameters}

**Returns** prints and returns dictionary

**Return type** None or dict

### `evalml.pipelines.components.PolynomialDetrender.fit`

`PolynomialDetrender.fit(X, y=None)`

Fits the PolynomialDetrender.

#### **Parameters**

- **X** (`ww.DataTable`, `pd.DataFrame`, *optional*) – Ignored.
- **y** (`ww.DataColumn`, `pd.Series`) – Target variable to detrend.

**Returns** self

### `evalml.pipelines.components.PolynomialDetrender.fit_transform`

`PolynomialDetrender.fit_transform(X, y=None)`

Removes fitted trend from target variable.

#### **Parameters**

- **X** (`ww.DataTable`, `pd.DataFrame`, *optional*) – Ignored.
- **y** (`ww.DataColumn`, `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 `ww.DataTable`, `ww.DataColumn`

### `evalml.pipelines.components.PolynomialDetrender.inverse_transform`

`PolynomialDetrender.inverse_transform(X, y)`

Adds back fitted trend to target variable.

#### **Parameters**

- **X** (`ww.DataTable`, `pd.DataFrame`, *optional*) – Ignored.
- **y** (`ww.DataColumn`, `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 `ww.DataTable`, `ww.DataColumn`



**evalml.pipelines.components.PolynomialDetrender.load**

**static** `PolynomialDetrender.load(file_path)`

Loads component at file path

**Parameters** `file_path` (*str*) – Location to load file

**Returns** ComponentBase object

**evalml.pipelines.components.PolynomialDetrender.save**

`PolynomialDetrender.save(file_path, pickle_protocol=5)`

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.PolynomialDetrender.transform**

`PolynomialDetrender.transform(X, y=None)`

Removes fitted trend from target variable.

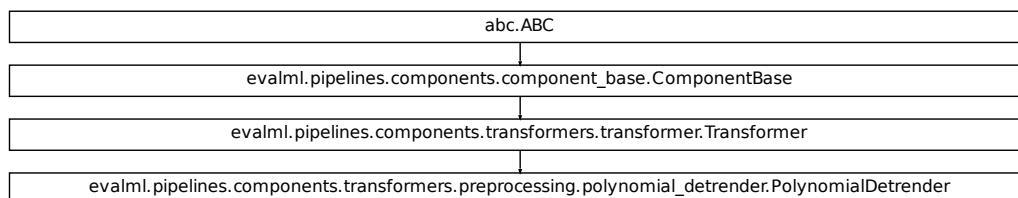
**Parameters**

- `X` (*ww.DataTable*, *pd.DataFrame*, *optional*) – Ignored.
- `y` (*ww.DataColumn*, *pd.Series*) – Target variable to detrend.

**Returns**

The input features are returned without modification. The target variable `y` is detrended

**Return type** tuple of *ww.DataTable*, *ww.DataColumn*

**Class Inheritance**

## evalml.pipelines.components.Undersampler

```
class evalml.pipelines.components.Undersampler(sampling_ratio=0.25,
                                                min_samples=100,
                                                min_percentage=0.1, random_seed=0,
                                                **kwargs)

Random undersampler component. This component is only run during training and not during predict.

name = 'Undersampler'
model_family = 'none'
hyperparameter_ranges = {}
default_parameters = {'min_percentage': 0.1, 'min_samples': 100, 'sampling_ratio': 0.2}
```

### Instance attributes

<code>needs_fitting</code>	
<code>parameters</code>	Returns the parameters which were used to initialize the component

### Methods:

<code>__init__</code>	Initializes an undersampling transformer to down-sample the majority classes in the dataset.
<code>clone</code>	Constructs a new component with the same parameters and random state.
<code>describe</code>	Describe a component and its parameters
<code>fit</code>	Resample the data using the sampler.
<code>fit_transform</code>	Fit and transform the data using the undersampler.
<code>load</code>	Loads component at file path
<code>save</code>	Saves component at file path
<code>transform</code>	No transformation needs to be done here.

## evalml.pipelines.components.Undersampler.\_\_init\_\_

```
Undersampler.__init__(sampling_ratio=0.25, min_samples=100, min_percentage=0.1, random_seed=0, **kwargs)

Initializes an undersampling transformer to downsample the majority classes in the dataset.
```

### 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.
- **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.

### evalml.pipelines.components.Undersampler.clone

Undersampler.**clone**()

Constructs a new component with the same parameters and random state.

**Returns** A new instance of this component with identical parameters and random state.

### evalml.pipelines.components.Undersampler.describe

Undersampler.**describe**(*print\_name=False, return\_dict=False*)

Describe a component and its parameters

#### Parameters

- **print\_name** (*bool, optional*) – whether to print name of component
- **return\_dict** (*bool, optional*) – whether to return description as dictionary in the format {"name": name, "parameters": parameters}

**Returns** prints and returns dictionary

**Return type** None or dict

### evalml.pipelines.components.Undersampler.fit

Undersampler.**fit**(*X, y*)

Resample the data using the sampler. Since our sampler doesn't need to be fit, we do nothing here.

#### Parameters

- **X** (*ww.DataFrame*) – Training features
- **y** (*ww.DataColumn*) – Target features

**Returns** self

### evalml.pipelines.components.Undersampler.fit\_transform

Undersampler.**fit\_transform**(*X, y*)

Fit and transform the data using the undersampler. Used during training of the pipeline

#### Parameters

- **X** (*ww.DataFrame*) – Training features
- **y** – Target features

**evalml.pipelines.components.Undersampler.load****static** Undersampler.load(*file\_path*)

Loads component at file path

**Parameters** **file\_path** (*str*) – Location to load file**Returns** ComponentBase object**evalml.pipelines.components.Undersampler.save**Undersampler.save(*file\_path*, *pickle\_protocol=5*)

Saves component at file path

**Parameters**

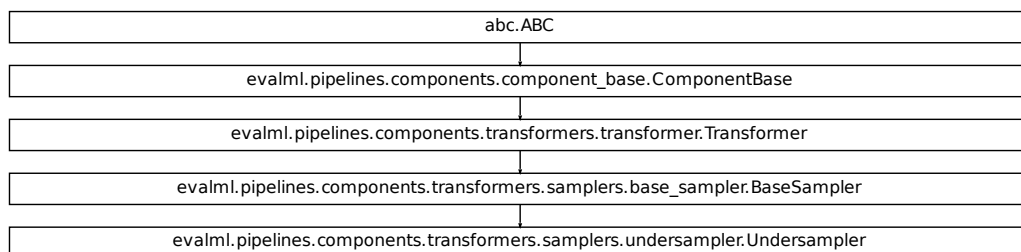
- **file\_path** (*str*) – Location to save file
- **pickle\_protocol** (*int*) – The pickle data stream format.

**Returns** None**evalml.pipelines.components.Undersampler.transform**Undersampler.transform(*X*, *y=None*)

No transformation needs to be done here.

**Parameters**

- **X** (*ww.DataFrame*) – Training features. Ignored.
- **y** (*ww.DataColumn*) – Target features. Ignored.

**Returns** X and y data that was passed in.**Return type** ww.DataTable, ww.DataColumn**Class Inheritance**

## 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.
<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>StackedEnsembleClassifier</i>	Stacked Ensemble Classifier.
<i>DecisionTreeClassifier</i>	Decision Tree Classifier.
<i>KNeighborsClassifier</i>	K-Nearest Neighbors Classifier.
<i>SVMClassifier</i>	Support Vector Machine Classifier.

### evalml.pipelines.components.CatBoostClassifier

```
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, **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/>

```
name = 'CatBoost Classifier'
```

```
model_family = 'catboost'
```

```
supported_problem_types = [<ProblemTypes.BINARY: 'binary'>, <ProblemTypes.MULTICLASS: 'multiclass'>]
```

```
hyperparameter_ranges = {'eta': Real(low=1e-06, high=1, prior='uniform', transform='id')}
```

```
default_parameters = {'allow_writing_files': False, 'bootstrap_type': None, 'eta': 0.03}
```

```
predict_uses_y = False
```

#### Instance attributes

<code>feature_importance</code>	Return an attribute of instance, which is of type owner.
<code>needs_fitting</code>	
<code>parameters</code>	Returns the parameters which were used to initialize the component

**Methods:**

<code>__init__</code>	Initialize self.
<code>clone</code>	Constructs a new component with the same parameters and random state.
<code>describe</code>	Describe a component and its parameters
<code>fit</code>	Fits component to data
<code>load</code>	Loads component at file path
<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

**evalml.pipelines.components.CatBoostClassifier.\_\_init\_\_**

`CatBoostClassifier.__init__(n_estimators=10, eta=0.03, max_depth=6, bootstrap_type=None, silent=True, allow_writing_files=False, random_seed=0, **kwargs)`  
Initialize self. See help(type(self)) for accurate signature.

**evalml.pipelines.components.CatBoostClassifier.clone**

`CatBoostClassifier.clone()`  
Constructs a new component with the same parameters and random state.  
**Returns** A new instance of this component with identical parameters and random state.

**evalml.pipelines.components.CatBoostClassifier.describe**

`CatBoostClassifier.describe(print_name=False, return_dict=False)`  
Describe a component and its parameters

**Parameters**

- **print\_name** (*bool, optional*) – whether to print name of component
- **return\_dict** (*bool, optional*) – whether to return description as dictionary in the format {"name": name, "parameters": parameters}

**Returns** prints and returns dictionary

**Return type** None or dict

**evalml.pipelines.components.CatBoostClassifier.fit**

`CatBoostClassifier.fit(X, y=None)`  
Fits component to data

**Parameters**

- **X** (*list, ww.DataTable, pd.DataFrame or np.ndarray*) – The input training data of shape [n\_samples, n\_features]

- **y** (*list, ww.DataColumn, pd.Series, np.ndarray, optional*) – The target training data of length [n\_samples]

**Returns** self

### evalml.pipelines.components.CatBoostClassifier.load

**static** CatBoostClassifier.**load** (*file\_path*)

Loads component at file path

**Parameters** **file\_path** (*str*) – Location to load file

**Returns** ComponentBase object

### evalml.pipelines.components.CatBoostClassifier.predict

CatBoostClassifier.**predict** (*X*)

Make predictions using selected features.

**Parameters** **X** (*ww.DataTable, pd.DataFrame, or np.ndarray*) – Data of shape [n\_samples, n\_features]

**Returns** Predicted values

**Return type** ww.DataColumn

### evalml.pipelines.components.CatBoostClassifier.predict\_proba

CatBoostClassifier.**predict\_proba** (*X*)

Make probability estimates for labels.

**Parameters** **X** (*ww.DataTable, pd.DataFrame, or np.ndarray*) – Features

**Returns** Probability estimates

**Return type** ww.DataTable

### evalml.pipelines.components.CatBoostClassifier.save

CatBoostClassifier.**save** (*file\_path, pickle\_protocol=5*)

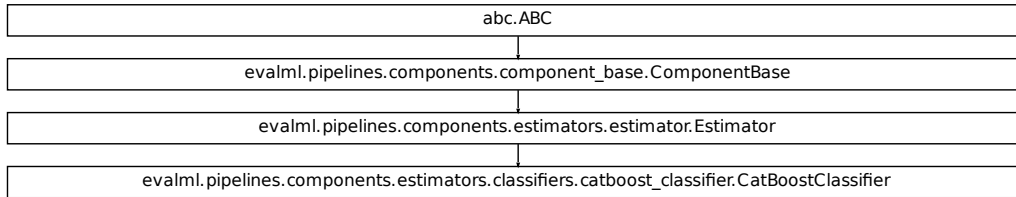
Saves component at file path

**Parameters**

- **file\_path** (*str*) – Location to save file
- **pickle\_protocol** (*int*) – The pickle data stream format.

**Returns** None

## Class Inheritance



### evalml.pipelines.components.ElasticNetClassifier

```
class evalml.pipelines.components.ElasticNetClassifier (alpha=0.5, l1_ratio=0.5,  
n_jobs=- 1, max_iter=1000,  
random_seed=0,  
penalty='elasticnet',  
**kwargs)  
  
    Elastic Net Classifier.  
  
    name = 'Elastic Net Classifier'  
    model_family = 'linear_model'  
    supported_problem_types = [<ProblemTypes.BINARY: 'binary'>, <ProblemTypes.MULTICLASS:  
    hyperparameter_ranges = {'alpha': Real(low=0, high=1, prior='uniform', transform='iden  
    default_parameters = {'alpha': 0.5, 'l1_ratio': 0.5, 'loss': 'log', 'max_iter': 1000,  
    predict_uses_y = False
```

#### Instance attributes

feature_importance	Return an attribute of instance, which is of type owner.
needs_fitting	
parameters	Returns the parameters which were used to initialize the component



**Methods:**

<code>__init__</code>	Initialize self.
<code>clone</code>	Constructs a new component with the same parameters and random state.
<code>describe</code>	Describe a component and its parameters
<code>fit</code>	Fits component to data
<code>load</code>	Loads component at file path
<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

**evalml.pipelines.components.ElasticNetClassifier.\_\_init\_\_**

`ElasticNetClassifier.__init__` (*alpha=0.5, l1\_ratio=0.5, n\_jobs=- 1, max\_iter=1000, random\_seed=0, penalty='elasticnet', \*\*kwargs*)  
 Initialize self. See help(type(self)) for accurate signature.

**evalml.pipelines.components.ElasticNetClassifier.clone**

`ElasticNetClassifier.clone()`  
 Constructs a new component with the same parameters and random state.

**Returns** A new instance of this component with identical parameters and random state.

**evalml.pipelines.components.ElasticNetClassifier.describe**

`ElasticNetClassifier.describe` (*print\_name=False, return\_dict=False*)  
 Describe a component and its parameters

**Parameters**

- **print\_name** (*bool, optional*) – whether to print name of component
- **return\_dict** (*bool, optional*) – whether to return description as dictionary in the format {"name": name, "parameters": parameters}

**Returns** prints and returns dictionary

**Return type** None or dict

**evalml.pipelines.components.ElasticNetClassifier.fit**

`ElasticNetClassifier.fit` (*X, y=None*)  
 Fits component to data

**Parameters**

- **X** (*list, ww.DataTable, pd.DataFrame or np.ndarray*) – The input training data of shape [n\_samples, n\_features]
- **y** (*list, ww.DataColumn, pd.Series, np.ndarray, optional*) – The target training data of length [n\_samples]

**Returns** self

### `evalml.pipelines.components.ElasticNetClassifier.load`

**static** `ElasticNetClassifier.load(file_path)`

Loads component at file path

**Parameters** `file_path` (*str*) – Location to load file

**Returns** ComponentBase object

### `evalml.pipelines.components.ElasticNetClassifier.predict`

`ElasticNetClassifier.predict(X)`

Make predictions using selected features.

**Parameters** `X` (*ww.DataTable*, *pd.DataFrame*, or *np.ndarray*) – Data of shape [n\_samples, n\_features]

**Returns** Predicted values

**Return type** *ww.DataColumn*

### `evalml.pipelines.components.ElasticNetClassifier.predict_proba`

`ElasticNetClassifier.predict_proba(X)`

Make probability estimates for labels.

**Parameters** `X` (*ww.DataTable*, *pd.DataFrame*, or *np.ndarray*) – Features

**Returns** Probability estimates

**Return type** *ww.DataTable*

### `evalml.pipelines.components.ElasticNetClassifier.save`

`ElasticNetClassifier.save(file_path, pickle_protocol=5)`

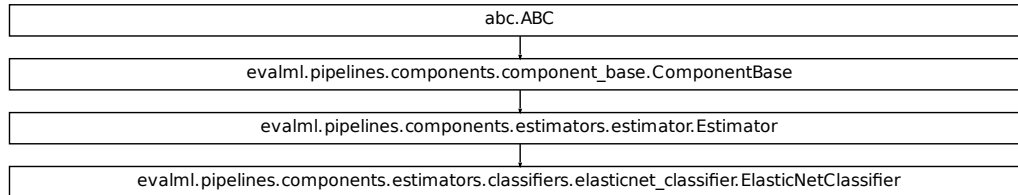
Saves component at file path

**Parameters**

- **file\_path** (*str*) – Location to save file
- **pickle\_protocol** (*int*) – The pickle data stream format.

**Returns** None

## Class Inheritance



### evalml.pipelines.components.ExtraTreesClassifier

```

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.

```
name = 'Extra Trees Classifier'
```

```
model_family = 'extra_trees'
```

```
supported_problem_types = [<ProblemTypes.BINARY: 'binary'>, <ProblemTypes.MULTICLASS:
```

```
hyperparameter_ranges = {'max_depth': Integer(low=4, high=10, prior='uniform', transfo
```

```
default_parameters = {'max_depth': 6, 'max_features': 'auto', 'min_samples_split': 2,
```

```
predict_uses_y = False
```

#### Instance attributes

<code>feature_importance</code>	Returns importance associated with each feature.
<code>needs_fitting</code>	
<code>parameters</code>	Returns the parameters which were used to initialize the component

**Methods:**

<code>__init__</code>	Initialize self.
<code>clone</code>	Constructs a new component with the same parameters and random state.
<code>describe</code>	Describe a component and its parameters
<code>fit</code>	Fits component to data
<code>load</code>	Loads component at file path
<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

**evalml.pipelines.components.ExtraTreesClassifier.\_\_init\_\_**

`ExtraTreesClassifier.__init__(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)`  
Initialize self. See help(type(self)) for accurate signature.

**evalml.pipelines.components.ExtraTreesClassifier.clone**

`ExtraTreesClassifier.clone()`  
Constructs a new component with the same parameters and random state.

**Returns** A new instance of this component with identical parameters and random state.

**evalml.pipelines.components.ExtraTreesClassifier.describe**

`ExtraTreesClassifier.describe(print_name=False, return_dict=False)`  
Describe a component and its parameters

**Parameters**

- **print\_name** (*bool, optional*) – whether to print name of component
- **return\_dict** (*bool, optional*) – whether to return description as dictionary in the format {"name": name, "parameters": parameters}

**Returns** prints and returns dictionary

**Return type** None or dict

**evalml.pipelines.components.ExtraTreesClassifier.fit**

`ExtraTreesClassifier.fit(X, y=None)`  
Fits component to data

**Parameters**

- **X** (*list, ww.DataTable, pd.DataFrame or np.ndarray*) – The input training data of shape [n\_samples, n\_features]

- **y** (*list, ww.DataColumn, pd.Series, np.ndarray, optional*) – The target training data of length [n\_samples]

**Returns** self

### evalml.pipelines.components.ExtraTreesClassifier.load

**static** ExtraTreesClassifier.**load** (*file\_path*)

Loads component at file path

**Parameters** **file\_path** (*str*) – Location to load file

**Returns** ComponentBase object

### evalml.pipelines.components.ExtraTreesClassifier.predict

ExtraTreesClassifier.**predict** (*X*)

Make predictions using selected features.

**Parameters** **X** (*ww.DataTable, pd.DataFrame, or np.ndarray*) – Data of shape [n\_samples, n\_features]

**Returns** Predicted values

**Return type** ww.DataColumn

### evalml.pipelines.components.ExtraTreesClassifier.predict\_proba

ExtraTreesClassifier.**predict\_proba** (*X*)

Make probability estimates for labels.

**Parameters** **X** (*ww.DataTable, pd.DataFrame, or np.ndarray*) – Features

**Returns** Probability estimates

**Return type** ww.DataTable

### evalml.pipelines.components.ExtraTreesClassifier.save

ExtraTreesClassifier.**save** (*file\_path, pickle\_protocol=5*)

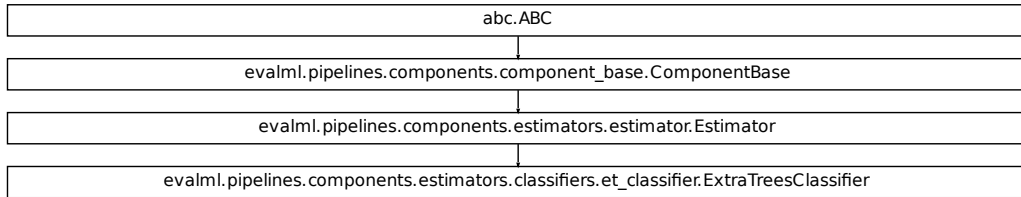
Saves component at file path

**Parameters**

- **file\_path** (*str*) – Location to save file
- **pickle\_protocol** (*int*) – The pickle data stream format.

**Returns** None

## Class Inheritance



### evalml.pipelines.components.RandomForestClassifier

```
class evalml.pipelines.components.RandomForestClassifier (n_estimators=100,  
                                                    max_depth=6, n_jobs=-  
                                                    1, random_seed=0,  
                                                    **kwargs)  
  
    Random Forest Classifier.  
  
    name = 'Random Forest Classifier'  
    model_family = 'random_forest'  
    supported_problem_types = [<ProblemTypes.BINARY: 'binary'>, <ProblemTypes.MULTICLASS:  
    hyperparameter_ranges = {'max_depth': Integer(low=1, high=10, prior='uniform', transfo  
    default_parameters = {'max_depth': 6, 'n_estimators': 100, 'n_jobs': -1}  
    predict_uses_y = False
```

#### Instance attributes

feature_importance	Returns importance associated with each feature.
needs_fitting	
parameters	Returns the parameters which were used to initialize the component

**Methods:**

<code>__init__</code>	Initialize self.
<code>clone</code>	Constructs a new component with the same parameters and random state.
<code>describe</code>	Describe a component and its parameters
<code>fit</code>	Fits component to data
<code>load</code>	Loads component at file path
<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

**evalml.pipelines.components.RandomForestClassifier.\_\_init\_\_**

`RandomForestClassifier.__init__(n_estimators=100, max_depth=6, n_jobs=-1, random_seed=0, **kwargs)`  
 Initialize self. See help(type(self)) for accurate signature.

**evalml.pipelines.components.RandomForestClassifier.clone**

`RandomForestClassifier.clone()`  
 Constructs a new component with the same parameters and random state.

**Returns** A new instance of this component with identical parameters and random state.

**evalml.pipelines.components.RandomForestClassifier.describe**

`RandomForestClassifier.describe(print_name=False, return_dict=False)`  
 Describe a component and its parameters

**Parameters**

- **print\_name** (*bool, optional*) – whether to print name of component
- **return\_dict** (*bool, optional*) – whether to return description as dictionary in the format {"name": name, "parameters": parameters}

**Returns** prints and returns dictionary

**Return type** None or dict

**evalml.pipelines.components.RandomForestClassifier.fit**

`RandomForestClassifier.fit(X, y=None)`  
 Fits component to data

**Parameters**

- **X** (*list, ww.DataTable, pd.DataFrame or np.ndarray*) – The input training data of shape [n\_samples, n\_features]
- **y** (*list, ww.DataColumn, pd.Series, np.ndarray, optional*) – The target training data of length [n\_samples]

**Returns** self

### **evalml.pipelines.components.RandomForestClassifier.load**

**static** RandomForestClassifier.**load**(*file\_path*)

Loads component at file path

**Parameters** **file\_path** (*str*) – Location to load file

**Returns** ComponentBase object

### **evalml.pipelines.components.RandomForestClassifier.predict**

RandomForestClassifier.**predict**(*X*)

Make predictions using selected features.

**Parameters** **X** (*ww.DataTable*, *pd.DataFrame*, or *np.ndarray*) – Data of shape [n\_samples, n\_features]

**Returns** Predicted values

**Return type** ww.DataColumn

### **evalml.pipelines.components.RandomForestClassifier.predict\_proba**

RandomForestClassifier.**predict\_proba**(*X*)

Make probability estimates for labels.

**Parameters** **X** (*ww.DataTable*, *pd.DataFrame*, or *np.ndarray*) – Features

**Returns** Probability estimates

**Return type** ww.DataTable

### **evalml.pipelines.components.RandomForestClassifier.save**

RandomForestClassifier.**save**(*file\_path*, *pickle\_protocol=5*)

Saves component at file path

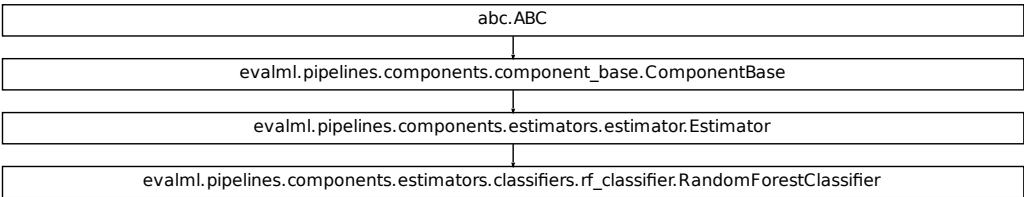
**Parameters**

- **file\_path** (*str*) – Location to save file
- **pickle\_protocol** (*int*) – The pickle data stream format.

**Returns** None



Class Inheritance



evalml.pipelines.components.LightGBMClassifier

```
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,
                                                       n_jobs=- 1, random_seed=0,
                                                       bagging_fraction=0.9,  bag-
                                                       ging_freq=0, **kwargs)

LightGBM Classifier
name = 'LightGBM Classifier'
model_family = 'lightgbm'
supported_problem_types = [<ProblemTypes.BINARY: 'binary'>, <ProblemTypes.MULTICLASS:
hyperparameter_ranges = {'bagging_fraction': Real(low=1e-06, high=1, prior='uniform',
default_parameters = {'bagging_fraction': 0.9, 'bagging_freq': 0, 'boosting_type': 'gb
predict_uses_y = False
```

Instance attributes

SEED_MAX	
SEED_MIN	
feature_importance	Returns importance associated with each feature.
needs_fitting	
parameters	Returns the parameters which were used to initialize the component

**Methods:**

<code>__init__</code>	Initialize self.
<code>clone</code>	Constructs a new component with the same parameters and random state.
<code>describe</code>	Describe a component and its parameters
<code>fit</code>	Fits component to data
<code>load</code>	Loads component at file path
<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

**evalml.pipelines.components.LightGBMClassifier.\_\_init\_\_**

`LightGBMClassifier.__init__` (*boosting\_type='gbdt', learning\_rate=0.1, n\_estimators=100, max\_depth=0, num\_leaves=31, min\_child\_samples=20, n\_jobs=- 1, random\_seed=0, bagging\_fraction=0.9, bagging\_freq=0, \*\*kwargs*)

Initialize self. See help(type(self)) for accurate signature.

**evalml.pipelines.components.LightGBMClassifier.clone**

`LightGBMClassifier.clone()`

Constructs a new component with the same parameters and random state.

**Returns** A new instance of this component with identical parameters and random state.

**evalml.pipelines.components.LightGBMClassifier.describe**

`LightGBMClassifier.describe` (*print\_name=False, return\_dict=False*)

Describe a component and its parameters

**Parameters**

- **print\_name** (*bool, optional*) – whether to print name of component
- **return\_dict** (*bool, optional*) – whether to return description as dictionary in the format {"name": name, "parameters": parameters}

**Returns** prints and returns dictionary

**Return type** None or dict

**evalml.pipelines.components.LightGBMClassifier.fit**`LightGBMClassifier.fit(X, y=None)`

Fits component to data

**Parameters**

- **X** (*list*, *ww.DataTable*, *pd.DataFrame* or *np.ndarray*) – The input training data of shape [n\_samples, n\_features]
- **y** (*list*, *ww.DataColumn*, *pd.Series*, *np.ndarray*, *optional*) – The target training data of length [n\_samples]

**Returns** self**evalml.pipelines.components.LightGBMClassifier.load**`static LightGBMClassifier.load(file_path)`

Loads component at file path

**Parameters** **file\_path** (*str*) – Location to load file**Returns** ComponentBase object**evalml.pipelines.components.LightGBMClassifier.predict**`LightGBMClassifier.predict(X)`

Make predictions using selected features.

**Parameters** **X** (*ww.DataTable*, *pd.DataFrame*, or *np.ndarray*) – Data of shape [n\_samples, n\_features]**Returns** Predicted values**Return type** *ww.DataColumn***evalml.pipelines.components.LightGBMClassifier.predict\_proba**`LightGBMClassifier.predict_proba(X)`

Make probability estimates for labels.

**Parameters** **X** (*ww.DataTable*, *pd.DataFrame*, or *np.ndarray*) – Features**Returns** Probability estimates**Return type** *ww.DataTable*

### `evalml.pipelines.components.LightGBMClassifier.save`

`LightGBMClassifier.save` (*file\_path*, *pickle\_protocol*=5)

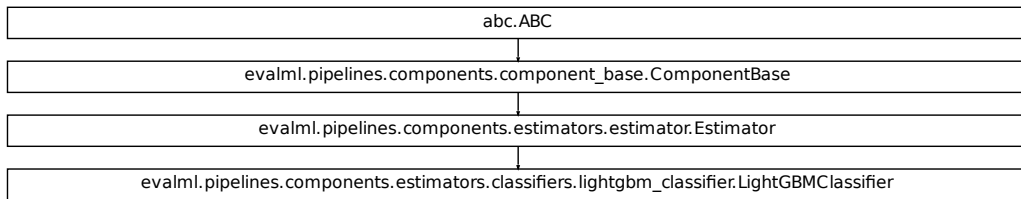
Saves component at file path

#### Parameters

- **file\_path** (*str*) – Location to save file
- **pickle\_protocol** (*int*) – The pickle data stream format.

**Returns** None

### Class Inheritance



### `evalml.pipelines.components.LogisticRegressionClassifier`

```
class evalml.pipelines.components.LogisticRegressionClassifier (penalty='l2',  
                                                             C=1.0,  
                                                             n_jobs=-1,  
                                                             multi_class='auto',  
                                                             solver='lbfgs',  
                                                             random_state=0,  
                                                             **kwargs)
```

Logistic Regression Classifier.

```
name = 'Logistic Regression Classifier'
```

```
model_family = 'linear_model'
```

```
supported_problem_types = [<ProblemTypes.BINARY: 'binary'>, <ProblemTypes.MULTICLASS:
```

```
hyperparameter_ranges = {'C': Real(low=0.01, high=10, prior='uniform', transform='iden
```

```
default_parameters = {'C': 1.0, 'multi_class': 'auto', 'n_jobs': -1, 'penalty': 'l2',
```

```
predict_uses_y = False
```

**Instance attributes**

<code>feature_importance</code>	Return an attribute of instance, which is of type owner.
<code>needs_fitting</code>	
<code>parameters</code>	Returns the parameters which were used to initialize the component

**Methods:**

<code>__init__</code>	Initialize self.
<code>clone</code>	Constructs a new component with the same parameters and random state.
<code>describe</code>	Describe a component and its parameters
<code>fit</code>	Fits component to data
<code>load</code>	Loads component at file path
<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

**evalml.pipelines.components.LogisticRegressionClassifier.\_\_init\_\_**

`LogisticRegressionClassifier.__init__(penalty='l2', C=1.0, n_jobs=-1, multi_class='auto', solver='lbfgs', random_seed=0, **kwargs)`

Initialize self. See help(type(self)) for accurate signature.

**evalml.pipelines.components.LogisticRegressionClassifier.clone**

`LogisticRegressionClassifier.clone()`

Constructs a new component with the same parameters and random state.

**Returns** A new instance of this component with identical parameters and random state.

**evalml.pipelines.components.LogisticRegressionClassifier.describe**

`LogisticRegressionClassifier.describe(print_name=False, return_dict=False)`

Describe a component and its parameters

**Parameters**

- **print\_name** (*bool, optional*) – whether to print name of component
- **return\_dict** (*bool, optional*) – whether to return description as dictionary in the format {"name": name, "parameters": parameters}

**Returns** prints and returns dictionary

**Return type** None or dict

**evalml.pipelines.components.LogisticRegressionClassifier.fit**

LogisticRegressionClassifier.**fit** (*X*, *y=None*)

Fits component to data

**Parameters**

- **X** (*list*, *ww.DataTable*, *pd.DataFrame* or *np.ndarray*) – The input training data of shape [n\_samples, n\_features]
- **y** (*list*, *ww.DataColumn*, *pd.Series*, *np.ndarray*, *optional*) – The target training data of length [n\_samples]

**Returns** self

**evalml.pipelines.components.LogisticRegressionClassifier.load**

**static** LogisticRegressionClassifier.**load** (*file\_path*)

Loads component at file path

**Parameters** **file\_path** (*str*) – Location to load file

**Returns** ComponentBase object

**evalml.pipelines.components.LogisticRegressionClassifier.predict**

LogisticRegressionClassifier.**predict** (*X*)

Make predictions using selected features.

**Parameters** **X** (*ww.DataTable*, *pd.DataFrame*, or *np.ndarray*) – Data of shape [n\_samples, n\_features]

**Returns** Predicted values

**Return type** ww.DataColumn

**evalml.pipelines.components.LogisticRegressionClassifier.predict\_proba**

LogisticRegressionClassifier.**predict\_proba** (*X*)

Make probability estimates for labels.

**Parameters** **X** (*ww.DataTable*, *pd.DataFrame*, or *np.ndarray*) – Features

**Returns** Probability estimates

**Return type** ww.DataTable

**evalml.pipelines.components.LogisticRegressionClassifier.save**

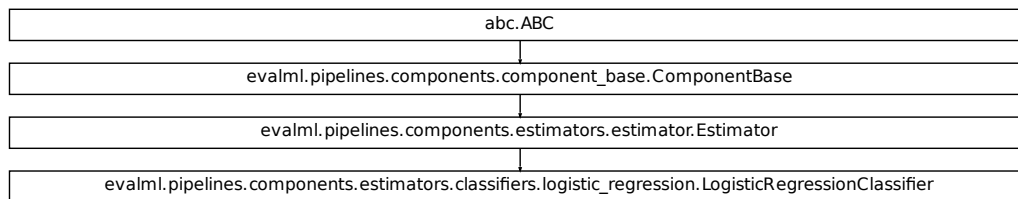
`LogisticRegressionClassifier.save(file_path, pickle_protocol=5)`

Saves component at file path

**Parameters**

- **file\_path** (*str*) – Location to save file
- **pickle\_protocol** (*int*) – The pickle data stream format.

**Returns** None

**Class Inheritance****evalml.pipelines.components.XGBoostClassifier**

```

class evalml.pipelines.components.XGBoostClassifier(eta=0.1,          max_depth=6,
                                                    min_child_weight=1,
                                                    n_estimators=100,      ran-
                                                    dom_seed=0, **kwargs)

```

XGBoost Classifier.

**name** = 'XGBoost Classifier'

**model\_family** = 'xgboost'

**supported\_problem\_types** = [<ProblemTypes.BINARY: 'binary'>, <ProblemTypes.MULTICLASS:

**hyperparameter\_ranges** = {'eta': Real(low=1e-06, high=1, prior='uniform', transform='id

**default\_parameters** = {'eta': 0.1, 'max\_depth': 6, 'min\_child\_weight': 1, 'n\_estimators

**predict\_uses\_y** = False

### Instance attributes

SEED_MAX	
SEED_MIN	
feature_importance	Return an attribute of instance, which is of type owner.
needs_fitting	
parameters	Returns the parameters which were used to initialize the component

### Methods:

<code>__init__</code>	Initialize self.
<code>clone</code>	Constructs a new component with the same parameters and random state.
<code>describe</code>	Describe a component and its parameters
<code>fit</code>	Fits component to data
<code>load</code>	Loads component at file path
<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

### `evalml.pipelines.components.XGBoostClassifier.__init__`

`XGBoostClassifier.__init__(eta=0.1, max_depth=6, min_child_weight=1, n_estimators=100, random_seed=0, **kwargs)`  
Initialize self. See `help(type(self))` for accurate signature.

### `evalml.pipelines.components.XGBoostClassifier.clone`

`XGBoostClassifier.clone()`  
Constructs a new component with the same parameters and random state.

**Returns** A new instance of this component with identical parameters and random state.

### `evalml.pipelines.components.XGBoostClassifier.describe`

`XGBoostClassifier.describe(print_name=False, return_dict=False)`  
Describe a component and its parameters

#### Parameters

- **print\_name** (*bool, optional*) – whether to print name of component
- **return\_dict** (*bool, optional*) – whether to return description as dictionary in the format {"name": name, "parameters": parameters}



**Returns** prints and returns dictionary

**Return type** None or dict

### `evalml.pipelines.components.XGBoostClassifier.fit`

`XGBoostClassifier.fit(X, y=None)`

Fits component to data

#### **Parameters**

- **X** (*list*, *ww.DataTable*, *pd.DataFrame* or *np.ndarray*) – The input training data of shape [n\_samples, n\_features]
- **y** (*list*, *ww.DataColumn*, *pd.Series*, *np.ndarray*, *optional*) – The target training data of length [n\_samples]

**Returns** self

### `evalml.pipelines.components.XGBoostClassifier.load`

**static** `XGBoostClassifier.load(file_path)`

Loads component at file path

**Parameters** **file\_path** (*str*) – Location to load file

**Returns** ComponentBase object

### `evalml.pipelines.components.XGBoostClassifier.predict`

`XGBoostClassifier.predict(X)`

Make predictions using selected features.

**Parameters** **X** (*ww.DataTable*, *pd.DataFrame*, or *np.ndarray*) – Data of shape [n\_samples, n\_features]

**Returns** Predicted values

**Return type** *ww.DataColumn*

### `evalml.pipelines.components.XGBoostClassifier.predict_proba`

`XGBoostClassifier.predict_proba(X)`

Make probability estimates for labels.

**Parameters** **X** (*ww.DataTable*, *pd.DataFrame*, or *np.ndarray*) – Features

**Returns** Probability estimates

**Return type** *ww.DataTable*

### `evalml.pipelines.components.XGBoostClassifier.save`

`XGBoostClassifier.save(file_path, pickle_protocol=5)`

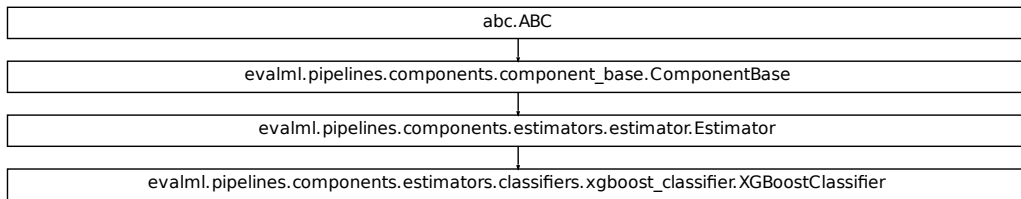
Saves component at file path

#### Parameters

- **file\_path** (*str*) – Location to save file
- **pickle\_protocol** (*int*) – The pickle data stream format.

**Returns** None

### Class Inheritance



### `evalml.pipelines.components.BaselineClassifier`

**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.

**name** = 'Baseline Classifier'

**model\_family** = 'baseline'

**supported\_problem\_types** = [`<ProblemTypes.BINARY: 'binary'>`, `<ProblemTypes.MULTICLASS:`

`hyperparameter_ranges` = {}

**default\_parameters** = {'strategy': 'mode'}

**predict\_uses\_y** = False

**Instance attributes**

<code>classes_</code>	Returns class labels.
<code>feature_importance</code>	Returns importance associated with each feature.
<code>needs_fitting</code>	
<code>parameters</code>	Returns the parameters which were used to initialize the component

**Methods:**

<code>__init__</code>	Baseline classifier that uses a simple strategy to make predictions.
<code>clone</code>	Constructs a new component with the same parameters and random state.
<code>describe</code>	Describe a component and its parameters
<code>fit</code>	Fits component to data
<code>load</code>	Loads component at file path
<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

**evalml.pipelines.components.BaselineClassifier.\_\_init\_\_**

`BaselineClassifier.__init__(strategy='mode', random_seed=0, **kwargs)`

Baseline classifier that uses a simple strategy to make predictions.

**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.

**evalml.pipelines.components.BaselineClassifier.clone**

`BaselineClassifier.clone()`

Constructs a new component with the same parameters and random state.

**Returns** A new instance of this component with identical parameters and random state.

**evalml.pipelines.components.BaselineClassifier.describe**

`BaselineClassifier.describe(print_name=False, return_dict=False)`

Describe a component and its parameters

**Parameters**

- **print\_name** (*bool, optional*) – whether to print name of component
- **return\_dict** (*bool, optional*) – whether to return description as dictionary in the format {"name": name, "parameters": parameters}

**Returns** prints and returns dictionary

**Return type** None or dict

**evalml.pipelines.components.BaselineClassifier.fit**

`BaselineClassifier.fit(X, y=None)`

Fits component to data

**Parameters**

- **X** (*list, ww.DataTable, pd.DataFrame or np.ndarray*) – The input training data of shape [n\_samples, n\_features]
- **y** (*list, ww.DataColumn, pd.Series, np.ndarray, optional*) – The target training data of length [n\_samples]

**Returns** self

**evalml.pipelines.components.BaselineClassifier.load**

**static** `BaselineClassifier.load(file_path)`

Loads component at file path

**Parameters** **file\_path** (*str*) – Location to load file

**Returns** ComponentBase object

**evalml.pipelines.components.BaselineClassifier.predict**`BaselineClassifier.predict(X)`

Make predictions using selected features.

**Parameters** *X* (*ww.DataTable*, *pd.DataFrame*, or *np.ndarray*) – Data of shape [n\_samples, n\_features]**Returns** Predicted values**Return type** *ww.DataColumn***evalml.pipelines.components.BaselineClassifier.predict\_proba**`BaselineClassifier.predict_proba(X)`

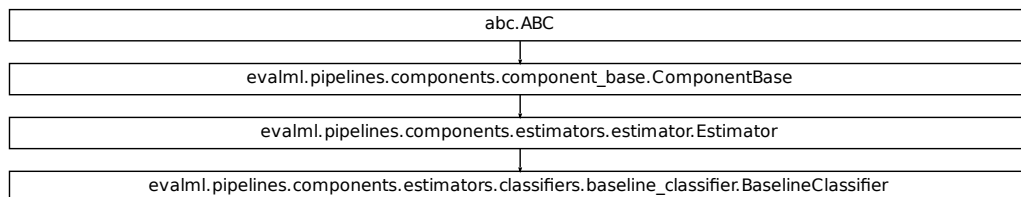
Make probability estimates for labels.

**Parameters** *X* (*ww.DataTable*, *pd.DataFrame*, or *np.ndarray*) – Features**Returns** Probability estimates**Return type** *ww.DataTable***evalml.pipelines.components.BaselineClassifier.save**`BaselineClassifier.save(file_path, pickle_protocol=5)`

Saves component at file path

**Parameters**

- **file\_path** (*str*) – Location to save file
- **pickle\_protocol** (*int*) – The pickle data stream format.

**Returns** None**Class Inheritance**

**evalml.pipelines.components.StackedEnsembleClassifier**

```
class evalml.pipelines.components.StackedEnsembleClassifier(input_pipelines=None,
                                                           fi-
                                                           nal_estimator=None,
                                                           cv=None, n_jobs=-
                                                           1, random_seed=0,
                                                           **kwargs)

    Stacked Ensemble Classifier.

    name = 'Stacked Ensemble Classifier'
    model_family = 'ensemble'
    supported_problem_types = [<ProblemTypes.BINARY: 'binary'>, <ProblemTypes.MULTICLASS:
    hyperparameter_ranges = {}
    default_parameters = {'cv': None, 'final_estimator': None, 'n_jobs': -1}
    predict_uses_y = False
```

**Instance attributes**

feature_importance	Not implemented for StackedEnsembleClassifier and StackedEnsembleRegressor
needs_fitting	
parameters	Returns the parameters which were used to initialize the component

**Methods:**

<code>__init__</code>	Stacked ensemble classifier.
<code>clone</code>	Constructs a new component with the same parameters and random state.
<code>describe</code>	Describe a component and its parameters
<code>fit</code>	Fits component to data
<code>load</code>	Loads component at file path
<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

**evalml.pipelines.components.StackedEnsembleClassifier.\_\_init\_\_**

`StackedEnsembleClassifier.__init__(input_pipelines=None, final_estimator=None, cv=None, n_jobs=-1, random_seed=0, **kwargs)`

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 None. - 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.

**evalml.pipelines.components.StackedEnsembleClassifier.clone**

`StackedEnsembleClassifier.clone()`

Constructs a new component with the same parameters and random state.

**Returns** A new instance of this component with identical parameters and random state.

**evalml.pipelines.components.StackedEnsembleClassifier.describe**

`StackedEnsembleClassifier.describe(print_name=False, return_dict=False)`

Describe a component and its parameters

**Parameters**

- **print\_name** (*bool, optional*) – whether to print name of component
- **return\_dict** (*bool, optional*) – whether to return description as dictionary in the format {"name": name, "parameters": parameters}

**Returns** prints and returns dictionary

**Return type** None or dict

**evalml.pipelines.components.StackedEnsembleClassifier.fit**

StackedEnsembleClassifier.**fit** (*X*, *y=None*)

Fits component to data

**Parameters**

- **X** (*list*, *ww.DataTable*, *pd.DataFrame* or *np.ndarray*) – The input training data of shape [n\_samples, n\_features]
- **y** (*list*, *ww.DataColumn*, *pd.Series*, *np.ndarray*, *optional*) – The target training data of length [n\_samples]

**Returns** self

**evalml.pipelines.components.StackedEnsembleClassifier.load**

**static** StackedEnsembleClassifier.**load** (*file\_path*)

Loads component at file path

**Parameters** **file\_path** (*str*) – Location to load file

**Returns** ComponentBase object

**evalml.pipelines.components.StackedEnsembleClassifier.predict**

StackedEnsembleClassifier.**predict** (*X*)

Make predictions using selected features.

**Parameters** **X** (*ww.DataTable*, *pd.DataFrame*, or *np.ndarray*) – Data of shape [n\_samples, n\_features]

**Returns** Predicted values

**Return type** ww.DataColumn

**evalml.pipelines.components.StackedEnsembleClassifier.predict\_proba**

StackedEnsembleClassifier.**predict\_proba** (*X*)

Make probability estimates for labels.

**Parameters** **X** (*ww.DataTable*, *pd.DataFrame*, or *np.ndarray*) – Features

**Returns** Probability estimates

**Return type** ww.DataTable



**evalml.pipelines.components.StackedEnsembleClassifier.save**

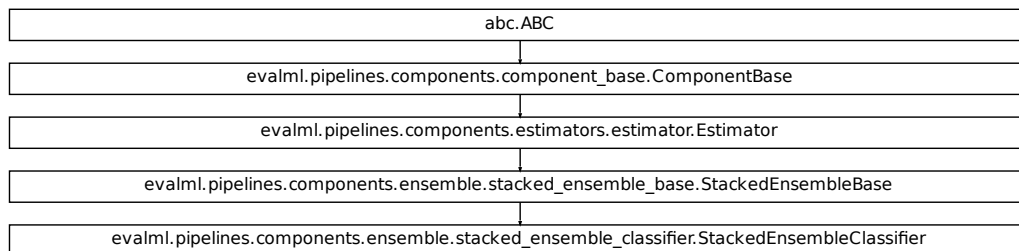
`StackedEnsembleClassifier.save(file_path, pickle_protocol=5)`

Saves component at file path

**Parameters**

- **file\_path** (*str*) – Location to save file
- **pickle\_protocol** (*int*) – The pickle data stream format.

**Returns** None

**Class Inheritance****evalml.pipelines.components.DecisionTreeClassifier**

```

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.

**name** = 'Decision Tree Classifier'

**model\_family** = 'decision\_tree'

**supported\_problem\_types** = [<ProblemTypes.BINARY: 'binary'>, <ProblemTypes.MULTICLASS:

**hyperparameter\_ranges** = {'criterion': ['gini', 'entropy'], 'max\_depth': Integer(low=4,

**default\_parameters** = {'criterion': 'gini', 'max\_depth': 6, 'max\_features': 'auto', 'mi

**predict\_uses\_y** = False

### Instance attributes

<code>feature_importance</code>	Returns importance associated with each feature.
<code>needs_fitting</code>	
<code>parameters</code>	Returns the parameters which were used to initialize the component

### Methods:

<code>__init__</code>	Initialize self.
<code>clone</code>	Constructs a new component with the same parameters and random state.
<code>describe</code>	Describe a component and its parameters
<code>fit</code>	Fits component to data
<code>load</code>	Loads component at file path
<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

### `evalml.pipelines.components.DecisionTreeClassifier.__init__`

`DecisionTreeClassifier.__init__(criterion='gini', max_features='auto', max_depth=6, min_samples_split=2, min_weight_fraction_leaf=0.0, random_seed=0, **kwargs)`

Initialize self. See `help(type(self))` for accurate signature.

### `evalml.pipelines.components.DecisionTreeClassifier.clone`

`DecisionTreeClassifier.clone()`

Constructs a new component with the same parameters and random state.

**Returns** A new instance of this component with identical parameters and random state.

### `evalml.pipelines.components.DecisionTreeClassifier.describe`

`DecisionTreeClassifier.describe(print_name=False, return_dict=False)`

Describe a component and its parameters

#### Parameters

- **print\_name** (*bool, optional*) – whether to print name of component
- **return\_dict** (*bool, optional*) – whether to return description as dictionary in the format {"name": name, "parameters": parameters}

**Returns** prints and returns dictionary

**Return type** None or dict

**evalml.pipelines.components.DecisionTreeClassifier.fit**

`DecisionTreeClassifier.fit(X, y=None)`

Fits component to data

**Parameters**

- **X** (*list, ww.DataTable, pd.DataFrame or np.ndarray*) – The input training data of shape [n\_samples, n\_features]
- **y** (*list, ww.DataColumn, pd.Series, np.ndarray, optional*) – The target training data of length [n\_samples]

**Returns** self

**evalml.pipelines.components.DecisionTreeClassifier.load**

**static** `DecisionTreeClassifier.load(file_path)`

Loads component at file path

**Parameters** **file\_path** (*str*) – Location to load file

**Returns** ComponentBase object

**evalml.pipelines.components.DecisionTreeClassifier.predict**

`DecisionTreeClassifier.predict(X)`

Make predictions using selected features.

**Parameters** **X** (*ww.DataTable, pd.DataFrame, or np.ndarray*) – Data of shape [n\_samples, n\_features]

**Returns** Predicted values

**Return type** ww.DataColumn

**evalml.pipelines.components.DecisionTreeClassifier.predict\_proba**

`DecisionTreeClassifier.predict_proba(X)`

Make probability estimates for labels.

**Parameters** **X** (*ww.DataTable, pd.DataFrame, or np.ndarray*) – Features

**Returns** Probability estimates

**Return type** ww.DataTable

### `evalml.pipelines.components.DecisionTreeClassifier.save`

`DecisionTreeClassifier.save` (*file\_path*, *pickle\_protocol=5*)

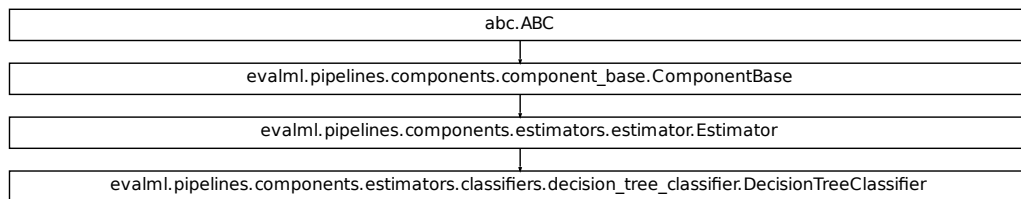
Saves component at file path

#### Parameters

- **file\_path** (*str*) – Location to save file
- **pickle\_protocol** (*int*) – The pickle data stream format.

**Returns** None

### Class Inheritance



### `evalml.pipelines.components.KNeighborsClassifier`

```
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.

**name** = 'KNN Classifier'

**model\_family** = 'k\_neighbors'

**supported\_problem\_types** = [`<ProblemTypes.BINARY: 'binary'>`, `<ProblemTypes.MULTICLASS:`

`hyperparameter_ranges` = {'algorithm': ['auto', 'ball\_tree', 'kd\_tree', 'brute'], 'leaf

`default_parameters` = {'algorithm': 'auto', 'leaf\_size': 30, 'n\_neighbors': 5, 'p': 2,

`predict_uses_y` = False

**Instance attributes**

<code>feature_importance</code>	Returns array of 0's matching the input number of features as <code>feature_importance</code> is not defined for KNN classifiers.
<code>needs_fitting</code>	
<code>parameters</code>	Returns the parameters which were used to initialize the component

**Methods:**

<code>__init__</code>	Initialize self.
<code>clone</code>	Constructs a new component with the same parameters and random state.
<code>describe</code>	Describe a component and its parameters
<code>fit</code>	Fits component to data
<code>load</code>	Loads component at file path
<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

**evalml.pipelines.components.KNeighborsClassifier.\_\_init\_\_**

`KNeighborsClassifier.__init__(n_neighbors=5, weights='uniform', algorithm='auto', leaf_size=30, p=2, random_seed=0, **kwargs)`  
 Initialize self. See help(type(self)) for accurate signature.

**evalml.pipelines.components.KNeighborsClassifier.clone**

`KNeighborsClassifier.clone()`  
 Constructs a new component with the same parameters and random state.

**Returns** A new instance of this component with identical parameters and random state.

**evalml.pipelines.components.KNeighborsClassifier.describe**

`KNeighborsClassifier.describe(print_name=False, return_dict=False)`  
 Describe a component and its parameters

**Parameters**

- **print\_name** (*bool, optional*) – whether to print name of component
- **return\_dict** (*bool, optional*) – whether to return description as dictionary in the format {"name": name, "parameters": parameters}

**Returns** prints and returns dictionary

**Return type** None or dict

**evalml.pipelines.components.KNeighborsClassifier.fit**

`KNeighborsClassifier.fit(X, y=None)`

Fits component to data

**Parameters**

- **X** (*list*, *ww.DataTable*, *pd.DataFrame* or *np.ndarray*) – The input training data of shape [n\_samples, n\_features]
- **y** (*list*, *ww.DataColumn*, *pd.Series*, *np.ndarray*, *optional*) – The target training data of length [n\_samples]

**Returns** self

**evalml.pipelines.components.KNeighborsClassifier.load**

**static** `KNeighborsClassifier.load(file_path)`

Loads component at file path

**Parameters** **file\_path** (*str*) – Location to load file

**Returns** ComponentBase object

**evalml.pipelines.components.KNeighborsClassifier.predict**

`KNeighborsClassifier.predict(X)`

Make predictions using selected features.

**Parameters** **X** (*ww.DataTable*, *pd.DataFrame*, or *np.ndarray*) – Data of shape [n\_samples, n\_features]

**Returns** Predicted values

**Return type** *ww.DataColumn*

**evalml.pipelines.components.KNeighborsClassifier.predict\_proba**

`KNeighborsClassifier.predict_proba(X)`

Make probability estimates for labels.

**Parameters** **X** (*ww.DataTable*, *pd.DataFrame*, or *np.ndarray*) – Features

**Returns** Probability estimates

**Return type** *ww.DataTable*

**evalml.pipelines.components.KNeighborsClassifier.save**

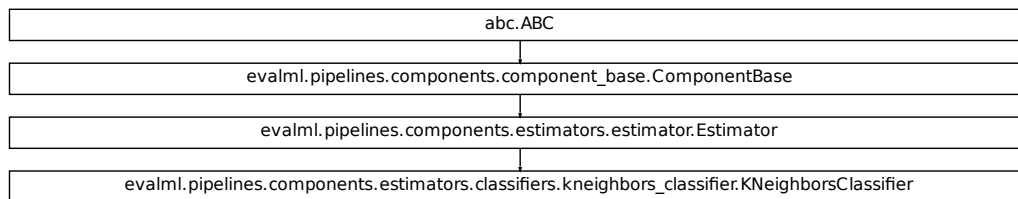
`KNeighborsClassifier.save(file_path, pickle_protocol=5)`

Saves component at file path

**Parameters**

- **file\_path** (*str*) – Location to save file
- **pickle\_protocol** (*int*) – The pickle data stream format.

**Returns** None

**Class Inheritance****evalml.pipelines.components.SVMClassifier**

```

class evalml.pipelines.components.SVMClassifier(C=1.0, kernel='rbf', gamma='scale',
                                              probability=True, random_seed=0,
                                              **kwargs)

```

Support Vector Machine Classifier.

**name** = 'SVM Classifier'

**model\_family** = 'svm'

**supported\_problem\_types** = [<ProblemTypes.BINARY: 'binary'>, <ProblemTypes.MULTICLASS:

**hyperparameter\_ranges** = {'C': Real(low=0, high=10, prior='uniform', transform='identit

**default\_parameters** = {'C': 1.0, 'gamma': 'scale', 'kernel': 'rbf', 'probability': True

**predict\_uses\_y** = False

### Instance attributes

<code>feature_importance</code>	Feature importance only works with linear kernels.
<code>needs_fitting</code>	
<code>parameters</code>	Returns the parameters which were used to initialize the component

### Methods:

<code>__init__</code>	Initialize self.
<code>clone</code>	Constructs a new component with the same parameters and random state.
<code>describe</code>	Describe a component and its parameters
<code>fit</code>	Fits component to data
<code>load</code>	Loads component at file path
<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

### `evalml.pipelines.components.SVMClassifier.__init__`

`SVMClassifier.__init__(C=1.0, kernel='rbf', gamma='scale', probability=True, random_seed=0, **kwargs)`  
Initialize self. See `help(type(self))` for accurate signature.

### `evalml.pipelines.components.SVMClassifier.clone`

`SVMClassifier.clone()`  
Constructs a new component with the same parameters and random state.

**Returns** A new instance of this component with identical parameters and random state.

### `evalml.pipelines.components.SVMClassifier.describe`

`SVMClassifier.describe(print_name=False, return_dict=False)`  
Describe a component and its parameters

#### Parameters

- **print\_name** (*bool, optional*) – whether to print name of component
- **return\_dict** (*bool, optional*) – whether to return description as dictionary in the format {"name": name, "parameters": parameters}

**Returns** prints and returns dictionary

**Return type** None or dict



**evalml.pipelines.components.SVMClassifier.fit****SVMClassifier.fit** (*X*, *y=None*)

Fits component to data

**Parameters**

- **X** (*list*, *ww.DataTable*, *pd.DataFrame* or *np.ndarray*) – The input training data of shape [n\_samples, n\_features]
- **y** (*list*, *ww.DataColumn*, *pd.Series*, *np.ndarray*, *optional*) – The target training data of length [n\_samples]

**Returns** self**evalml.pipelines.components.SVMClassifier.load****static SVMClassifier.load** (*file\_path*)

Loads component at file path

**Parameters** **file\_path** (*str*) – Location to load file**Returns** ComponentBase object**evalml.pipelines.components.SVMClassifier.predict****SVMClassifier.predict** (*X*)

Make predictions using selected features.

**Parameters** **X** (*ww.DataTable*, *pd.DataFrame*, or *np.ndarray*) – Data of shape [n\_samples, n\_features]**Returns** Predicted values**Return type** ww.DataColumn**evalml.pipelines.components.SVMClassifier.predict\_proba****SVMClassifier.predict\_proba** (*X*)

Make probability estimates for labels.

**Parameters** **X** (*ww.DataTable*, *pd.DataFrame*, or *np.ndarray*) – Features**Returns** Probability estimates**Return type** ww.DataTable

**evalml.pipelines.components.SVMClassifier.save**

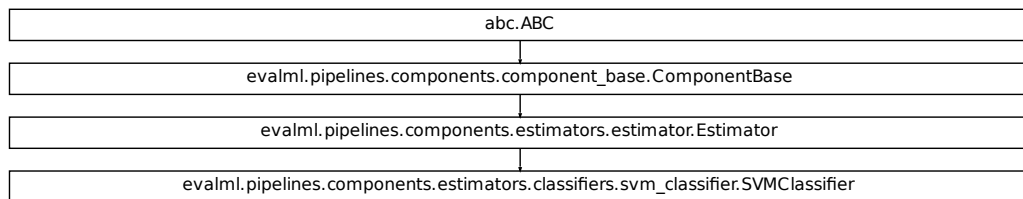
`SVMClassifier.save(file_path, pickle_protocol=5)`

Saves component at file path

**Parameters**

- **file\_path** (*str*) – Location to save file
- **pickle\_protocol** (*int*) – The pickle data stream format.

**Returns** None

**Class Inheritance****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>	Regressor that predicts using the specified strategy.
<i>TimeSeriesBaselineEstimator</i>	Time series estimator that predicts using the naive forecasting approach.
<i>StackedEnsembleRegressor</i>	Stacked Ensemble Regressor.
<i>DecisionTreeRegressor</i>	Decision Tree Regressor.
<i>LightGBMRegressor</i>	LightGBM Regressor
<i>SVMRegressor</i>	Support Vector Machine Regressor.

**evalml.pipelines.components.ARIMAREgressor**

```
class evalml.pipelines.components.ARIMAREgressor (date_column=None, trend='n',
                                                  p=1, d=0, q=0, 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)

```
name = 'ARIMA Regressor'
```

```
model_family = 'ARIMA'
```

```
supported_problem_types = [<ProblemTypes.TIME_SERIES_REGRESSION: 'time series regression']
```

```
hyperparameter_ranges = {'d': Integer(low=0, high=10, prior='uniform', transform='identity')}
```

```
default_parameters = {'d': 0, 'order': (1, 0, 0), 'p': 1, 'q': 0, 'trend': 'n'}
```

```
predict_uses_y = False
```

**Instance attributes**

<code>feature_importance</code>	Returns array of 0's with a length of 1 as feature_importance is not defined for ARIMA regressor.
<code>needs_fitting</code>	
<code>parameters</code>	Returns the parameters which were used to initialize the component

**Methods:**

<code>__init__</code>	<b>param date_column</b> Specifies the name of the column in X that provides the datetime objects. Defaults to None.
<code>clone</code>	Constructs a new component with the same parameters and random state.
<code>describe</code>	Describe a component and its parameters
<code>fit</code>	Fits component to data
<code>load</code>	Loads component at file path
<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

### evalml.pipelines.components.ARIMAREgressor.\_\_init\_\_

ARIMAREgressor.\_\_init\_\_(date\_column=None, trend='n', p=1, d=0, q=0, random\_seed=0, \*\*kwargs)

#### Parameters

- **date\_column** (*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].
- **p** (*int or list(int)*) – Autoregressive order.
- **d** (*int*) – Differencing degree.
- **q** (*int or list(int)*) – Moving Average order.

### evalml.pipelines.components.ARIMAREgressor.clone

ARIMAREgressor.clone()

Constructs a new component with the same parameters and random state.

**Returns** A new instance of this component with identical parameters and random state.

### evalml.pipelines.components.ARIMAREgressor.describe

ARIMAREgressor.describe(print\_name=False, return\_dict=False)

Describe a component and its parameters

#### Parameters

- **print\_name** (*bool, optional*) – whether to print name of component
- **return\_dict** (*bool, optional*) – whether to return description as dictionary in the format {"name": name, "parameters": parameters}

**Returns** prints and returns dictionary

**Return type** None or dict

### evalml.pipelines.components.ARIMAREgressor.fit

ARIMAREgressor.fit(X, y=None)

Fits component to data

#### Parameters

- **X** (*list, ww.DataTable, pd.DataFrame or np.ndarray*) – The input training data of shape [n\_samples, n\_features]
- **y** (*list, ww.DataColumn, pd.Series, np.ndarray, optional*) – The target training data of length [n\_samples]

**Returns** self

**evalml.pipelines.components.ARIMAREgressor.load****static** `ARIMAREgressor.load(file_path)`

Loads component at file path

**Parameters** `file_path` (*str*) – Location to load file**Returns** ComponentBase object**evalml.pipelines.components.ARIMAREgressor.predict**`ARIMAREgressor.predict(X, y=None)`

Make predictions using selected features.

**Parameters** `X` (*ww.DataTable*, *pd.DataFrame*, or *np.ndarray*) – Data of shape [n\_samples, n\_features]**Returns** Predicted values**Return type** *ww.DataColumn***evalml.pipelines.components.ARIMAREgressor.predict\_proba**`ARIMAREgressor.predict_proba(X)`

Make probability estimates for labels.

**Parameters** `X` (*ww.DataTable*, *pd.DataFrame*, or *np.ndarray*) – Features**Returns** Probability estimates**Return type** *ww.DataTable***evalml.pipelines.components.ARIMAREgressor.save**`ARIMAREgressor.save(file_path, pickle_protocol=5)`

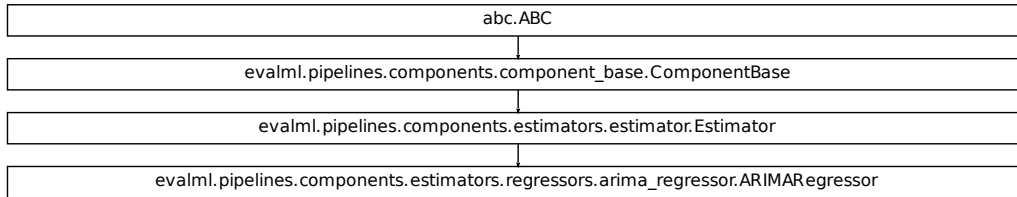
Saves component at file path

**Parameters**

- `file_path` (*str*) – Location to save file
- `pickle_protocol` (*int*) – The pickle data stream format.

**Returns** None

## Class Inheritance



### evalml.pipelines.components.CatBoostRegressor

```
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, **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/>

```
name = 'CatBoost Regressor'
```

```
model_family = 'catboost'
```

```
supported_problem_types = [<ProblemTypes.REGRESSION: 'regression'>, <ProblemTypes.TIME
```

```
hyperparameter_ranges = {'eta': Real(low=1e-06, high=1, prior='uniform', transform='id
```

```
default_parameters = {'allow_writing_files': False, 'bootstrap_type': None, 'eta': 0.0
```

```
predict_uses_y = False
```

#### Instance attributes

feature_importance	Return an attribute of instance, which is of type owner.
needs_fitting	
parameters	Returns the parameters which were used to initialize the component

**Methods:**

<code>__init__</code>	Initialize self.
<code>clone</code>	Constructs a new component with the same parameters and random state.
<code>describe</code>	Describe a component and its parameters
<code>fit</code>	Fits component to data
<code>load</code>	Loads component at file path
<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

**evalml.pipelines.components.CatBoostRegressor.\_\_init\_\_**

`CatBoostRegressor.__init__(n_estimators=10, eta=0.03, max_depth=6, bootstrap_type=None, silent=False, allow_writing_files=False, random_seed=0, **kwargs)`

Initialize self. See help(type(self)) for accurate signature.

**evalml.pipelines.components.CatBoostRegressor.clone**

`CatBoostRegressor.clone()`

Constructs a new component with the same parameters and random state.

**Returns** A new instance of this component with identical parameters and random state.

**evalml.pipelines.components.CatBoostRegressor.describe**

`CatBoostRegressor.describe(print_name=False, return_dict=False)`

Describe a component and its parameters

**Parameters**

- **print\_name** (*bool, optional*) – whether to print name of component
- **return\_dict** (*bool, optional*) – whether to return description as dictionary in the format {"name": name, "parameters": parameters}

**Returns** prints and returns dictionary

**Return type** None or dict

**evalml.pipelines.components.CatBoostRegressor.fit**

`CatBoostRegressor.fit(X, y=None)`

Fits component to data

**Parameters**

- **X** (*list, ww.DataTable, pd.DataFrame or np.ndarray*) – The input training data of shape [n\_samples, n\_features]

- **y** (*list, ww.DataColumn, pd.Series, np.ndarray, optional*) – The target training data of length [n\_samples]

**Returns** self

### **evalml.pipelines.components.CatBoostRegressor.load**

**static** CatBoostRegressor.**load** (*file\_path*)

Loads component at file path

**Parameters** **file\_path** (*str*) – Location to load file

**Returns** ComponentBase object

### **evalml.pipelines.components.CatBoostRegressor.predict**

CatBoostRegressor.**predict** (*X*)

Make predictions using selected features.

**Parameters** **X** (*ww.DataTable, pd.DataFrame, or np.ndarray*) – Data of shape [n\_samples, n\_features]

**Returns** Predicted values

**Return type** ww.DataColumn

### **evalml.pipelines.components.CatBoostRegressor.predict\_proba**

CatBoostRegressor.**predict\_proba** (*X*)

Make probability estimates for labels.

**Parameters** **X** (*ww.DataTable, pd.DataFrame, or np.ndarray*) – Features

**Returns** Probability estimates

**Return type** ww.DataTable

### **evalml.pipelines.components.CatBoostRegressor.save**

CatBoostRegressor.**save** (*file\_path, pickle\_protocol=5*)

Saves component at file path

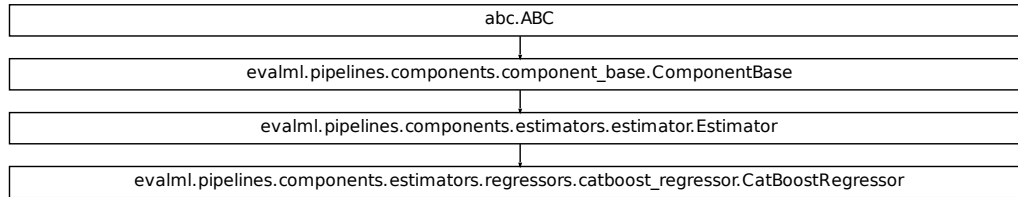
**Parameters**

- **file\_path** (*str*) – Location to save file
- **pickle\_protocol** (*int*) – The pickle data stream format.

**Returns** None



## Class Inheritance



### evalml.pipelines.components.ElasticNetRegressor

```

class evalml.pipelines.components.ElasticNetRegressor(alpha=0.5, l1_ratio=0.5,
max_iter=1000, normalize=False, random_seed=0,
**kwargs)

```

Elastic Net Regressor.

```
name = 'Elastic Net Regressor'
```

```
model_family = 'linear_model'
```

```
supported_problem_types = [<ProblemTypes.REGRESSION: 'regression'>, <ProblemTypes.TIME
```

```
hyperparameter_ranges = {'alpha': Real(low=0, high=1, prior='uniform', transform='iden
```

```
default_parameters = {'alpha': 0.5, 'l1_ratio': 0.5, 'max_iter': 1000, 'normalize': Fa
```

```
predict_uses_y = False
```

#### Instance attributes

feature_importance	Return an attribute of instance, which is of type owner.
needs_fitting	
parameters	Returns the parameters which were used to initialize the component

**Methods:**

<code>__init__</code>	Initialize self.
<code>clone</code>	Constructs a new component with the same parameters and random state.
<code>describe</code>	Describe a component and its parameters
<code>fit</code>	Fits component to data
<code>load</code>	Loads component at file path
<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

**evalml.pipelines.components.ElasticNetRegressor.\_\_init\_\_**

`ElasticNetRegressor.__init__` (*alpha=0.5, l1\_ratio=0.5, max\_iter=1000, normalize=False, random\_seed=0, \*\*kwargs*)  
Initialize self. See help(type(self)) for accurate signature.

**evalml.pipelines.components.ElasticNetRegressor.clone**

`ElasticNetRegressor.clone()`  
Constructs a new component with the same parameters and random state.

**Returns** A new instance of this component with identical parameters and random state.

**evalml.pipelines.components.ElasticNetRegressor.describe**

`ElasticNetRegressor.describe` (*print\_name=False, return\_dict=False*)  
Describe a component and its parameters

**Parameters**

- **print\_name** (*bool, optional*) – whether to print name of component
- **return\_dict** (*bool, optional*) – whether to return description as dictionary in the format {"name": name, "parameters": parameters}

**Returns** prints and returns dictionary

**Return type** None or dict

**evalml.pipelines.components.ElasticNetRegressor.fit**

`ElasticNetRegressor.fit` (*X, y=None*)  
Fits component to data

**Parameters**

- **X** (*list, ww.DataTable, pd.DataFrame or np.ndarray*) – The input training data of shape [n\_samples, n\_features]
- **y** (*list, ww.DataColumn, pd.Series, np.ndarray, optional*) – The target training data of length [n\_samples]

**Returns** self

### evalml.pipelines.components.ElasticNetRegressor.load

**static** ElasticNetRegressor.load(*file\_path*)

Loads component at file path

**Parameters** *file\_path* (*str*) – Location to load file

**Returns** ComponentBase object

### evalml.pipelines.components.ElasticNetRegressor.predict

ElasticNetRegressor.predict(*X*)

Make predictions using selected features.

**Parameters** *X* (*ww.DataTable*, *pd.DataFrame*, or *np.ndarray*) – Data of shape [n\_samples, n\_features]

**Returns** Predicted values

**Return type** ww.DataColumn

### evalml.pipelines.components.ElasticNetRegressor.predict\_proba

ElasticNetRegressor.predict\_proba(*X*)

Make probability estimates for labels.

**Parameters** *X* (*ww.DataTable*, *pd.DataFrame*, or *np.ndarray*) – Features

**Returns** Probability estimates

**Return type** ww.DataTable

### evalml.pipelines.components.ElasticNetRegressor.save

ElasticNetRegressor.save(*file\_path*, *pickle\_protocol*=5)

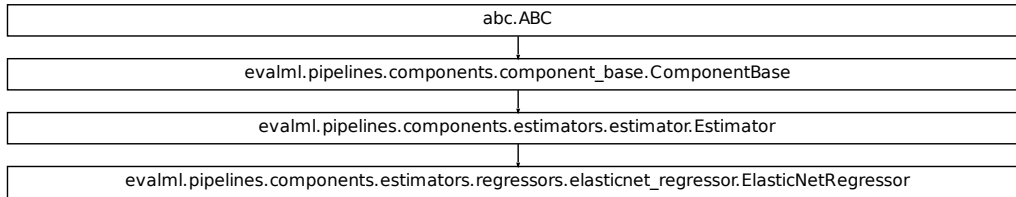
Saves component at file path

**Parameters**

- **file\_path** (*str*) – Location to save file
- **pickle\_protocol** (*int*) – The pickle data stream format.

**Returns** None

## Class Inheritance



### `evalml.pipelines.components.LinearRegressor`

```
class evalml.pipelines.components.LinearRegressor (fit_intercept=True, normalize=False, n_jobs=-1, random_seed=0, **kwargs)
```

Linear Regressor.

```
name = 'Linear Regressor'
```

```
model_family = 'linear_model'
```

```
supported_problem_types = [<ProblemTypes.REGRESSION: 'regression'>, <ProblemTypes.TIME
```

```
hyperparameter_ranges = {'fit_intercept': [True, False], 'normalize': [True, False]}
```

```
default_parameters = {'fit_intercept': True, 'n_jobs': -1, 'normalize': False}
```

```
predict_uses_y = False
```

#### Instance attributes

<code>feature_importance</code>	Return an attribute of instance, which is of type owner.
<code>needs_fitting</code>	
<code>parameters</code>	Returns the parameters which were used to initialize the component

**Methods:**

<code>__init__</code>	Initialize self.
<code>clone</code>	Constructs a new component with the same parameters and random state.
<code>describe</code>	Describe a component and its parameters
<code>fit</code>	Fits component to data
<code>load</code>	Loads component at file path
<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

**evalml.pipelines.components.LinearRegressor.\_\_init\_\_**

`LinearRegressor.__init__` (*fit\_intercept=True, normalize=False, n\_jobs=-1, random\_seed=0, \*\*kwargs*)

Initialize self. See help(type(self)) for accurate signature.

**evalml.pipelines.components.LinearRegressor.clone**

`LinearRegressor.clone()`

Constructs a new component with the same parameters and random state.

**Returns** A new instance of this component with identical parameters and random state.

**evalml.pipelines.components.LinearRegressor.describe**

`LinearRegressor.describe` (*print\_name=False, return\_dict=False*)

Describe a component and its parameters

**Parameters**

- **print\_name** (*bool, optional*) – whether to print name of component
- **return\_dict** (*bool, optional*) – whether to return description as dictionary in the format {"name": name, "parameters": parameters}

**Returns** prints and returns dictionary

**Return type** None or dict

**evalml.pipelines.components.LinearRegressor.fit**

`LinearRegressor.fit` (*X, y=None*)

Fits component to data

**Parameters**

- **X** (*list, ww.DataTable, pd.DataFrame or np.ndarray*) – The input training data of shape [n\_samples, n\_features]
- **y** (*list, ww.DataColumn, pd.Series, np.ndarray, optional*) – The target training data of length [n\_samples]

**Returns** self

### `evalml.pipelines.components.LinearRegressor.load`

**static** `LinearRegressor.load(file_path)`

Loads component at file path

**Parameters** `file_path` (*str*) – Location to load file

**Returns** ComponentBase object

### `evalml.pipelines.components.LinearRegressor.predict`

`LinearRegressor.predict(X)`

Make predictions using selected features.

**Parameters** `X` (*ww.DataTable*, *pd.DataFrame*, or *np.ndarray*) – Data of shape [n\_samples, n\_features]

**Returns** Predicted values

**Return type** *ww.DataColumn*

### `evalml.pipelines.components.LinearRegressor.predict_proba`

`LinearRegressor.predict_proba(X)`

Make probability estimates for labels.

**Parameters** `X` (*ww.DataTable*, *pd.DataFrame*, or *np.ndarray*) – Features

**Returns** Probability estimates

**Return type** *ww.DataTable*

### `evalml.pipelines.components.LinearRegressor.save`

`LinearRegressor.save(file_path, pickle_protocol=5)`

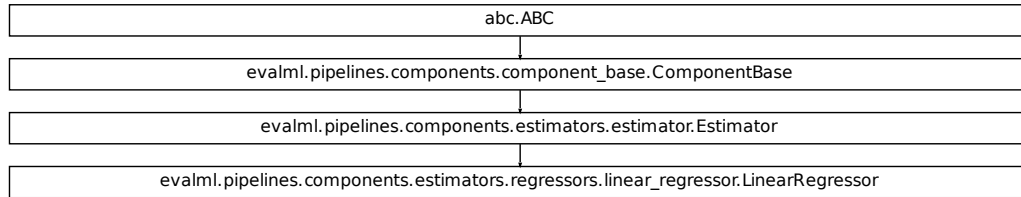
Saves component at file path

**Parameters**

- `file_path` (*str*) – Location to save file
- `pickle_protocol` (*int*) – The pickle data stream format.

**Returns** None

## Class Inheritance



### evalml.pipelines.components.ExtraTreesRegressor

```

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.

```
name = 'Extra Trees Regressor'
```

```
model_family = 'extra_trees'
```

```
supported_problem_types = [<ProblemTypes.REGRESSION: 'regression'>, <ProblemTypes.TIME
```

```
hyperparameter_ranges = {'max_depth': Integer(low=4, high=10, prior='uniform', transfo
```

```
default_parameters = {'max_depth': 6, 'max_features': 'auto', 'min_samples_split': 2,
```

```
predict_uses_y = False
```

#### Instance attributes

feature_importance	Returns importance associated with each feature.
needs_fitting	
parameters	Returns the parameters which were used to initialize the component

**Methods:**

<code>__init__</code>	Initialize self.
<code>clone</code>	Constructs a new component with the same parameters and random state.
<code>describe</code>	Describe a component and its parameters
<code>fit</code>	Fits component to data
<code>load</code>	Loads component at file path
<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

**evalml.pipelines.components.ExtraTreesRegressor.\_\_init\_\_**

`ExtraTreesRegressor.__init__(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)`  
Initialize self. See `help(type(self))` for accurate signature.

**evalml.pipelines.components.ExtraTreesRegressor.clone**

`ExtraTreesRegressor.clone()`  
Constructs a new component with the same parameters and random state.  
**Returns** A new instance of this component with identical parameters and random state.

**evalml.pipelines.components.ExtraTreesRegressor.describe**

`ExtraTreesRegressor.describe(print_name=False, return_dict=False)`  
Describe a component and its parameters

**Parameters**

- **print\_name** (*bool, optional*) – whether to print name of component
- **return\_dict** (*bool, optional*) – whether to return description as dictionary in the format {"name": name, "parameters": parameters}

**Returns** prints and returns dictionary

**Return type** None or dict

**evalml.pipelines.components.ExtraTreesRegressor.fit**

`ExtraTreesRegressor.fit(X, y=None)`  
Fits component to data

**Parameters**

- **X** (*list, ww.DataTable, pd.DataFrame or np.ndarray*) – The input training data of shape [n\_samples, n\_features]



- **y** (*list, ww.DataColumn, pd.Series, np.ndarray, optional*) – The target training data of length [n\_samples]

**Returns** self

### evalml.pipelines.components.ExtraTreesRegressor.load

**static** ExtraTreesRegressor.**load** (*file\_path*)

Loads component at file path

**Parameters** **file\_path** (*str*) – Location to load file

**Returns** ComponentBase object

### evalml.pipelines.components.ExtraTreesRegressor.predict

ExtraTreesRegressor.**predict** (*X*)

Make predictions using selected features.

**Parameters** **X** (*ww.DataTable, pd.DataFrame, or np.ndarray*) – Data of shape [n\_samples, n\_features]

**Returns** Predicted values

**Return type** ww.DataColumn

### evalml.pipelines.components.ExtraTreesRegressor.predict\_proba

ExtraTreesRegressor.**predict\_proba** (*X*)

Make probability estimates for labels.

**Parameters** **X** (*ww.DataTable, pd.DataFrame, or np.ndarray*) – Features

**Returns** Probability estimates

**Return type** ww.DataTable

### evalml.pipelines.components.ExtraTreesRegressor.save

ExtraTreesRegressor.**save** (*file\_path, pickle\_protocol=5*)

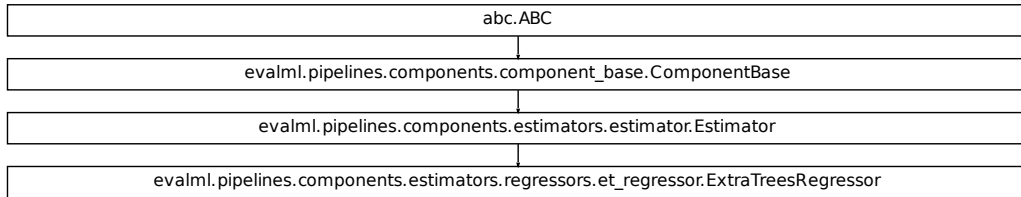
Saves component at file path

**Parameters**

- **file\_path** (*str*) – Location to save file
- **pickle\_protocol** (*int*) – The pickle data stream format.

**Returns** None

## Class Inheritance



### evalml.pipelines.components.RandomForestRegressor

```
class evalml.pipelines.components.RandomForestRegressor (n_estimators=100,  
                                                    max_depth=6, n_jobs=-  
                                                    1, random_seed=0,  
                                                    **kwargs)  
  
    Random Forest Regressor.  
  
    name = 'Random Forest Regressor'  
    model_family = 'random_forest'  
    supported_problem_types = [<ProblemTypes.REGRESSION: 'regression'>, <ProblemTypes.TIME  
    hyperparameter_ranges = {'max_depth': Integer(low=1, high=32, prior='uniform', transfo  
    default_parameters = {'max_depth': 6, 'n_estimators': 100, 'n_jobs': -1}  
    predict_uses_y = False
```

#### Instance attributes

feature_importance	Returns importance associated with each feature.
needs_fitting	
parameters	Returns the parameters which were used to initialize the component

**Methods:**

<code>__init__</code>	Initialize self.
<code>clone</code>	Constructs a new component with the same parameters and random state.
<code>describe</code>	Describe a component and its parameters
<code>fit</code>	Fits component to data
<code>load</code>	Loads component at file path
<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

**evalml.pipelines.components.RandomForestRegressor.\_\_init\_\_**

`RandomForestRegressor.__init__(n_estimators=100, max_depth=6, n_jobs=-1, random_seed=0, **kwargs)`  
 Initialize self. See help(type(self)) for accurate signature.

**evalml.pipelines.components.RandomForestRegressor.clone**

`RandomForestRegressor.clone()`  
 Constructs a new component with the same parameters and random state.

**Returns** A new instance of this component with identical parameters and random state.

**evalml.pipelines.components.RandomForestRegressor.describe**

`RandomForestRegressor.describe(print_name=False, return_dict=False)`  
 Describe a component and its parameters

**Parameters**

- **print\_name** (*bool, optional*) – whether to print name of component
- **return\_dict** (*bool, optional*) – whether to return description as dictionary in the format {"name": name, "parameters": parameters}

**Returns** prints and returns dictionary

**Return type** None or dict

**evalml.pipelines.components.RandomForestRegressor.fit**

`RandomForestRegressor.fit(X, y=None)`  
 Fits component to data

**Parameters**

- **X** (*list, ww.DataTable, pd.DataFrame or np.ndarray*) – The input training data of shape [n\_samples, n\_features]
- **y** (*list, ww.DataColumn, pd.Series, np.ndarray, optional*) – The target training data of length [n\_samples]

**Returns** self

### `evalml.pipelines.components.RandomForestRegressor.load`

**static** `RandomForestRegressor.load(file_path)`

Loads component at file path

**Parameters** `file_path` (*str*) – Location to load file

**Returns** ComponentBase object

### `evalml.pipelines.components.RandomForestRegressor.predict`

`RandomForestRegressor.predict(X)`

Make predictions using selected features.

**Parameters** `X` (*ww.DataTable*, *pd.DataFrame*, or *np.ndarray*) – Data of shape [n\_samples, n\_features]

**Returns** Predicted values

**Return type** *ww.DataColumn*

### `evalml.pipelines.components.RandomForestRegressor.predict_proba`

`RandomForestRegressor.predict_proba(X)`

Make probability estimates for labels.

**Parameters** `X` (*ww.DataTable*, *pd.DataFrame*, or *np.ndarray*) – Features

**Returns** Probability estimates

**Return type** *ww.DataTable*

### `evalml.pipelines.components.RandomForestRegressor.save`

`RandomForestRegressor.save(file_path, pickle_protocol=5)`

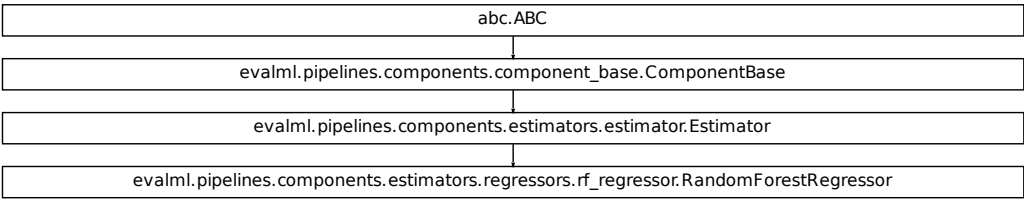
Saves component at file path

**Parameters**

- `file_path` (*str*) – Location to save file
- `pickle_protocol` (*int*) – The pickle data stream format.

**Returns** None

Class Inheritance



evalml.pipelines.components.XGBoostRegressor

```
class evalml.pipelines.components.XGBoostRegressor(eta=0.1, max_depth=6,
                                                    min_child_weight=1,
                                                    n_estimators=100, random_seed=0, **kwargs)
    XGBoost Regressor.
    name = 'XGBoost Regressor'
    model_family = 'xgboost'
    supported_problem_types = [<ProblemTypes.REGRESSION: 'regression'>, <ProblemTypes.TIME
    hyperparameter_ranges = {'eta': Real(low=1e-06, high=1, prior='uniform', transform='id
    default_parameters = {'eta': 0.1, 'max_depth': 6, 'min_child_weight': 1, 'n_estimators
    predict_uses_y = False
```

Instance attributes

SEED_MAX	
SEED_MIN	
feature_importance	Return an attribute of instance, which is of type owner.
needs_fitting	
parameters	Returns the parameters which were used to initialize the component

**Methods:**

<code>__init__</code>	Initialize self.
<code>clone</code>	Constructs a new component with the same parameters and random state.
<code>describe</code>	Describe a component and its parameters
<code>fit</code>	Fits component to data
<code>load</code>	Loads component at file path
<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

**evalml.pipelines.components.XGBoostRegressor.\_\_init\_\_**

`XGBoostRegressor.__init__(eta=0.1, max_depth=6, min_child_weight=1, n_estimators=100, random_seed=0, **kwargs)`  
Initialize self. See `help(type(self))` for accurate signature.

**evalml.pipelines.components.XGBoostRegressor.clone**

`XGBoostRegressor.clone()`  
Constructs a new component with the same parameters and random state.

**Returns** A new instance of this component with identical parameters and random state.

**evalml.pipelines.components.XGBoostRegressor.describe**

`XGBoostRegressor.describe(print_name=False, return_dict=False)`  
Describe a component and its parameters

**Parameters**

- **print\_name** (*bool, optional*) – whether to print name of component
- **return\_dict** (*bool, optional*) – whether to return description as dictionary in the format {"name": name, "parameters": parameters}

**Returns** prints and returns dictionary

**Return type** None or dict

**evalml.pipelines.components.XGBoostRegressor.fit**

`XGBoostRegressor.fit(X, y=None)`  
Fits component to data

**Parameters**

- **X** (*list, ww.DataTable, pd.DataFrame or np.ndarray*) – The input training data of shape [n\_samples, n\_features]
- **y** (*list, ww.DataColumn, pd.Series, np.ndarray, optional*) – The target training data of length [n\_samples]

**Returns** self

### evalml.pipelines.components.XGBoostRegressor.load

**static** XGBoostRegressor.load(*file\_path*)

Loads component at file path

**Parameters** *file\_path* (*str*) – Location to load file

**Returns** ComponentBase object

### evalml.pipelines.components.XGBoostRegressor.predict

XGBoostRegressor.predict(*X*)

Make predictions using selected features.

**Parameters** *X* (*ww.DataTable*, *pd.DataFrame*, or *np.ndarray*) – Data of shape [n\_samples, n\_features]

**Returns** Predicted values

**Return type** ww.DataColumn

### evalml.pipelines.components.XGBoostRegressor.predict\_proba

XGBoostRegressor.predict\_proba(*X*)

Make probability estimates for labels.

**Parameters** *X* (*ww.DataTable*, *pd.DataFrame*, or *np.ndarray*) – Features

**Returns** Probability estimates

**Return type** ww.DataTable

### evalml.pipelines.components.XGBoostRegressor.save

XGBoostRegressor.save(*file\_path*, *pickle\_protocol*=5)

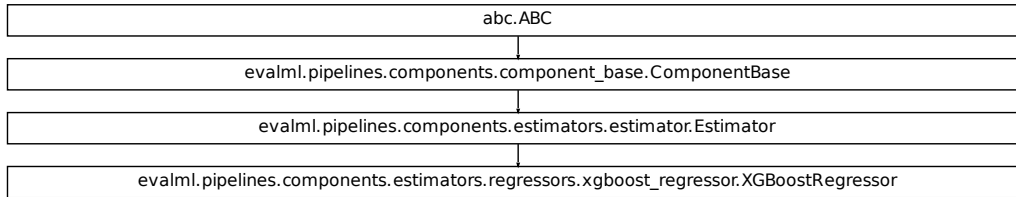
Saves component at file path

**Parameters**

- **file\_path** (*str*) – Location to save file
- **pickle\_protocol** (*int*) – The pickle data stream format.

**Returns** None

## Class Inheritance



### evalml.pipelines.components.BaselineRegressor

**class** evalml.pipelines.components.**BaselineRegressor** (*strategy='mean', random\_seed=0, \*\*kwargs*)

Regressor that predicts using the specified strategy.

This is useful as a simple baseline regressor to compare with other regressors.

**name** = 'Baseline Regressor'

**model\_family** = 'baseline'

**supported\_problem\_types** = [<ProblemTypes.REGRESSION: 'regression'>, <ProblemTypes.TIME

**hyperparameter\_ranges** = {}

**default\_parameters** = {'strategy': 'mean'}

**predict\_uses\_y** = False

#### Instance attributes

feature_importance	Returns importance associated with each feature.
needs_fitting	
parameters	Returns the parameters which were used to initialize the component



**Methods:**

<code>__init__</code>	Baseline regressor that uses a simple strategy to make predictions.
<code>clone</code>	Constructs a new component with the same parameters and random state.
<code>describe</code>	Describe a component and its parameters
<code>fit</code>	Fits component to data
<code>load</code>	Loads component at file path
<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

**evalml.pipelines.components.BaselineRegressor.\_\_init\_\_**

`BaselineRegressor.__init__(strategy='mean', random_seed=0, **kwargs)`

Baseline regressor that uses a simple strategy to make predictions.

**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.

**evalml.pipelines.components.BaselineRegressor.clone**

`BaselineRegressor.clone()`

Constructs a new component with the same parameters and random state.

**Returns** A new instance of this component with identical parameters and random state.

**evalml.pipelines.components.BaselineRegressor.describe**

`BaselineRegressor.describe(print_name=False, return_dict=False)`

Describe a component and its parameters

**Parameters**

- **print\_name** (*bool*, *optional*) – whether to print name of component
- **return\_dict** (*bool*, *optional*) – whether to return description as dictionary in the format {“name”: name, “parameters”: parameters}

**Returns** prints and returns dictionary

**Return type** None or dict

**evalml.pipelines.components.BaselineRegressor.fit**

`BaselineRegressor.fit(X, y=None)`

Fits component to data

**Parameters**

- **X** (*list*, *ww.DataTable*, *pd.DataFrame* or *np.ndarray*) – The input training data of shape [n\_samples, n\_features]
- **y** (*list*, *ww.DataColumn*, *pd.Series*, *np.ndarray*, *optional*) – The target training data of length [n\_samples]

**Returns** `self`

**evalml.pipelines.components.BaselineRegressor.load**

**static** `BaselineRegressor.load(file_path)`

Loads component at file path

**Parameters** **file\_path** (*str*) – Location to load file

**Returns** `ComponentBase` object

**evalml.pipelines.components.BaselineRegressor.predict**

`BaselineRegressor.predict(X)`

Make predictions using selected features.

**Parameters** **X** (*ww.DataTable*, *pd.DataFrame*, or *np.ndarray*) – Data of shape [n\_samples, n\_features]

**Returns** Predicted values

**Return type** `ww.DataColumn`

**evalml.pipelines.components.BaselineRegressor.predict\_proba**

`BaselineRegressor.predict_proba(X)`

Make probability estimates for labels.

**Parameters** **X** (*ww.DataTable*, *pd.DataFrame*, or *np.ndarray*) – Features

**Returns** Probability estimates

**Return type** `ww.DataTable`

**evalml.pipelines.components.BaselineRegressor.save**

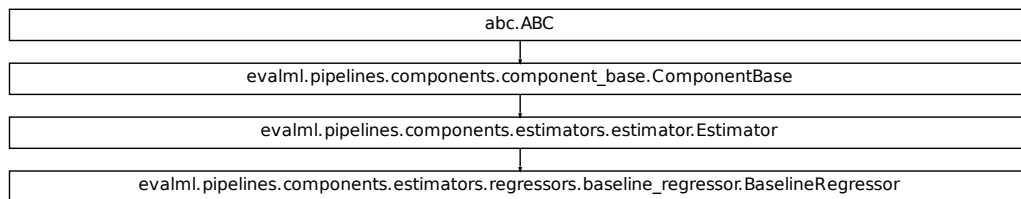
`BaselineRegressor.save(file_path, pickle_protocol=5)`

Saves component at file path

**Parameters**

- **file\_path** (*str*) – Location to save file
- **pickle\_protocol** (*int*) – The pickle data stream format.

**Returns** None

**Class Inheritance****evalml.pipelines.components.TimeSeriesBaselineEstimator**

```
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

```
name = 'Time Series Baseline Estimator'
```

```
model_family = 'baseline'
```

```
supported_problem_types = [<ProblemTypes.TIME_SERIES_REGRESSION: 'time series regression']
```

```
hyperparameter_ranges = {}
```

```
default_parameters = {'gap': 1}
```

```
predict_uses_y = True
```

### Instance attributes

<code>feature_importance</code>	Returns importance associated with each feature.
<code>needs_fitting</code>	
<code>parameters</code>	Returns the parameters which were used to initialize the component

### Methods:

<code>__init__</code>	Baseline time series estimator that predicts using the naive forecasting approach.
<code>clone</code>	Constructs a new component with the same parameters and random state.
<code>describe</code>	Describe a component and its parameters
<code>fit</code>	Fits component to data
<code>load</code>	Loads component at file path
<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

### `evalml.pipelines.components.TimeSeriesBaselineEstimator.__init__`

`TimeSeriesBaselineEstimator.__init__(gap=1, random_seed=0, **kwargs)`

Baseline time series estimator that predicts using the naive forecasting approach.

#### 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.
- **random\_state** (*None, int*) – Deprecated - use `random_seed` instead.
- **random\_seed** (*int*) – Seed for the random number generator. Defaults to 0.

**evalml.pipelines.components.TimeSeriesBaselineEstimator.clone**`TimeSeriesBaselineEstimator.clone()`

Constructs a new component with the same parameters and random state.

**Returns** A new instance of this component with identical parameters and random state.

**evalml.pipelines.components.TimeSeriesBaselineEstimator.describe**`TimeSeriesBaselineEstimator.describe(print_name=False, return_dict=False)`

Describe a component and its parameters

**Parameters**

- **print\_name** (*bool, optional*) – whether to print name of component
- **return\_dict** (*bool, optional*) – whether to return description as dictionary in the format {"name": name, "parameters": parameters}

**Returns** prints and returns dictionary

**Return type** None or dict

**evalml.pipelines.components.TimeSeriesBaselineEstimator.fit**`TimeSeriesBaselineEstimator.fit(X, y=None)`

Fits component to data

**Parameters**

- **X** (*list, ww.DataTable, pd.DataFrame or np.ndarray*) – The input training data of shape [n\_samples, n\_features]
- **y** (*list, ww.DataColumn, pd.Series, np.ndarray, optional*) – The target training data of length [n\_samples]

**Returns** self

**evalml.pipelines.components.TimeSeriesBaselineEstimator.load**`static TimeSeriesBaselineEstimator.load(file_path)`

Loads component at file path

**Parameters** **file\_path** (*str*) – Location to load file

**Returns** ComponentBase object

### evalml.pipelines.components.TimeSeriesBaselineEstimator.predict

`TimeSeriesBaselineEstimator.predict(X, y=None)`

Make predictions using selected features.

**Parameters** **X** (*ww.DataTable*, *pd.DataFrame*, or *np.ndarray*) – Data of shape [n\_samples, n\_features]

**Returns** Predicted values

**Return type** *ww.DataColumn*

### evalml.pipelines.components.TimeSeriesBaselineEstimator.predict\_proba

`TimeSeriesBaselineEstimator.predict_proba(X, y=None)`

Make probability estimates for labels.

**Parameters** **X** (*ww.DataTable*, *pd.DataFrame*, or *np.ndarray*) – Features

**Returns** Probability estimates

**Return type** *ww.DataTable*

### evalml.pipelines.components.TimeSeriesBaselineEstimator.save

`TimeSeriesBaselineEstimator.save(file_path, pickle_protocol=5)`

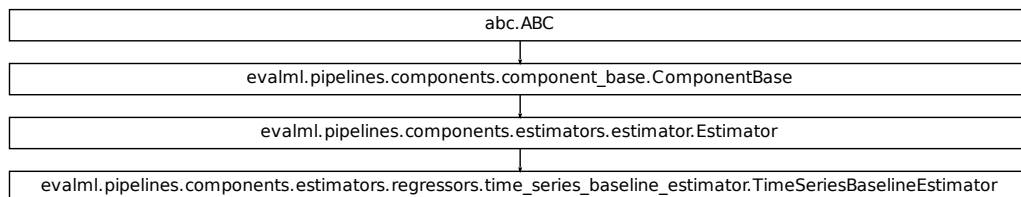
Saves component at file path

**Parameters**

- **file\_path** (*str*) – Location to save file
- **pickle\_protocol** (*int*) – The pickle data stream format.

**Returns** None

## Class Inheritance



**evalml.pipelines.components.StackedEnsembleRegressor**

```

class evalml.pipelines.components.StackedEnsembleRegressor (input_pipelines=None,
                                                            fi-
                                                            nal_estimator=None,
                                                            cv=None,    n_jobs=-
                                                            1,    random_seed=0,
                                                            **kwargs)

    Stacked Ensemble Regressor.

    name = 'Stacked Ensemble Regressor'
    model_family = 'ensemble'
    supported_problem_types = [<ProblemTypes.REGRESSION: 'regression'>, <ProblemTypes.TIME
    hyperparameter_ranges = {}
    default_parameters = {'cv': None, 'final_estimator': None, 'n_jobs': -1}
    predict_uses_y = False

```

**Instance attributes**

feature_importance	Not implemented for StackedEnsembleClassifier and StackedEnsembleRegressor
needs_fitting	
parameters	Returns the parameters which were used to initialize the component

**Methods:**

<code>__init__</code>	Stacked ensemble regressor.
<code>clone</code>	Constructs a new component with the same parameters and random state.
<code>describe</code>	Describe a component and its parameters
<code>fit</code>	Fits component to data
<code>load</code>	Loads component at file path
<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

### evalml.pipelines.components.StackedEnsembleRegressor.\_\_init\_\_

`StackedEnsembleRegressor.__init__(input_pipelines=None, final_estimator=None, cv=None, n_jobs=-1, random_seed=0, **kwargs)`

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 None. - 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.

### evalml.pipelines.components.StackedEnsembleRegressor.clone

`StackedEnsembleRegressor.clone()`

Constructs a new component with the same parameters and random state.

**Returns** A new instance of this component with identical parameters and random state.

### evalml.pipelines.components.StackedEnsembleRegressor.describe

`StackedEnsembleRegressor.describe(print_name=False, return_dict=False)`

Describe a component and its parameters

#### Parameters

- **print\_name** (*bool, optional*) – whether to print name of component
- **return\_dict** (*bool, optional*) – whether to return description as dictionary in the format {"name": name, "parameters": parameters}

**Returns** prints and returns dictionary

**Return type** None or dict



**evalml.pipelines.components.StackedEnsembleRegressor.fit**

StackedEnsembleRegressor.**fit** (*X*, *y=None*)

Fits component to data

**Parameters**

- **X** (*list*, *ww.DataTable*, *pd.DataFrame* or *np.ndarray*) – The input training data of shape [n\_samples, n\_features]
- **y** (*list*, *ww.DataColumn*, *pd.Series*, *np.ndarray*, *optional*) – The target training data of length [n\_samples]

**Returns** self

**evalml.pipelines.components.StackedEnsembleRegressor.load**

**static** StackedEnsembleRegressor.**load** (*file\_path*)

Loads component at file path

**Parameters** **file\_path** (*str*) – Location to load file

**Returns** ComponentBase object

**evalml.pipelines.components.StackedEnsembleRegressor.predict**

StackedEnsembleRegressor.**predict** (*X*)

Make predictions using selected features.

**Parameters** **X** (*ww.DataTable*, *pd.DataFrame*, or *np.ndarray*) – Data of shape [n\_samples, n\_features]

**Returns** Predicted values

**Return type** ww.DataColumn

**evalml.pipelines.components.StackedEnsembleRegressor.predict\_proba**

StackedEnsembleRegressor.**predict\_proba** (*X*)

Make probability estimates for labels.

**Parameters** **X** (*ww.DataTable*, *pd.DataFrame*, or *np.ndarray*) – Features

**Returns** Probability estimates

**Return type** ww.DataTable

## evalml.pipelines.components.StackedEnsembleRegressor.save

`StackedEnsembleRegressor.save(file_path, pickle_protocol=5)`

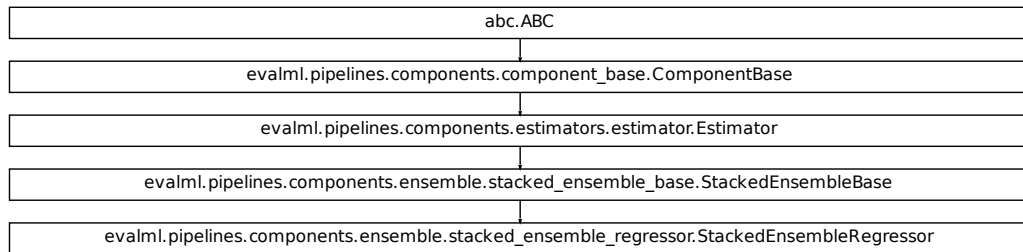
Saves component at file path

### Parameters

- **file\_path** (*str*) – Location to save file
- **pickle\_protocol** (*int*) – The pickle data stream format.

**Returns** None

## Class Inheritance



## evalml.pipelines.components.DecisionTreeRegressor

```

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.

```
name = 'Decision Tree Regressor'
```

```
model_family = 'decision_tree'
```

```
supported_problem_types = [<ProblemTypes.REGRESSION: 'regression'>, <ProblemTypes.TIME
```

```
hyperparameter_ranges = {'criterion': ['mse', 'friedman_mse', 'mae'], 'max_depth': Int
```

```
default_parameters = {'criterion': 'mse', 'max_depth': 6, 'max_features': 'auto', 'min
```

```
predict_uses_y = False
```

**Instance attributes**

<code>feature_importance</code>	Returns importance associated with each feature.
<code>needs_fitting</code>	
<code>parameters</code>	Returns the parameters which were used to initialize the component

**Methods:**

<code>__init__</code>	Initialize self.
<code>clone</code>	Constructs a new component with the same parameters and random state.
<code>describe</code>	Describe a component and its parameters
<code>fit</code>	Fits component to data
<code>load</code>	Loads component at file path
<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

**evalml.pipelines.components.DecisionTreeRegressor.\_\_init\_\_**

`DecisionTreeRegressor.__init__(criterion='mse', max_features='auto', max_depth=6, min_samples_split=2, min_weight_fraction_leaf=0.0, random_seed=0, **kwargs)`

Initialize self. See help(type(self)) for accurate signature.

**evalml.pipelines.components.DecisionTreeRegressor.clone**

`DecisionTreeRegressor.clone()`

Constructs a new component with the same parameters and random state.

**Returns** A new instance of this component with identical parameters and random state.

**evalml.pipelines.components.DecisionTreeRegressor.describe**

`DecisionTreeRegressor.describe(print_name=False, return_dict=False)`

Describe a component and its parameters

**Parameters**

- **print\_name** (*bool, optional*) – whether to print name of component
- **return\_dict** (*bool, optional*) – whether to return description as dictionary in the format {"name": name, "parameters": parameters}

**Returns** prints and returns dictionary

**Return type** None or dict

**evalml.pipelines.components.DecisionTreeRegressor.fit**

`DecisionTreeRegressor.fit(X, y=None)`

Fits component to data

**Parameters**

- **X** (*list*, *ww.DataTable*, *pd.DataFrame* or *np.ndarray*) – The input training data of shape [n\_samples, n\_features]
- **y** (*list*, *ww.DataColumn*, *pd.Series*, *np.ndarray*, *optional*) – The target training data of length [n\_samples]

**Returns** self

**evalml.pipelines.components.DecisionTreeRegressor.load**

**static** `DecisionTreeRegressor.load(file_path)`

Loads component at file path

**Parameters** **file\_path** (*str*) – Location to load file

**Returns** ComponentBase object

**evalml.pipelines.components.DecisionTreeRegressor.predict**

`DecisionTreeRegressor.predict(X)`

Make predictions using selected features.

**Parameters** **X** (*ww.DataTable*, *pd.DataFrame*, or *np.ndarray*) – Data of shape [n\_samples, n\_features]

**Returns** Predicted values

**Return type** *ww.DataColumn*

**evalml.pipelines.components.DecisionTreeRegressor.predict\_proba**

`DecisionTreeRegressor.predict_proba(X)`

Make probability estimates for labels.

**Parameters** **X** (*ww.DataTable*, *pd.DataFrame*, or *np.ndarray*) – Features

**Returns** Probability estimates

**Return type** *ww.DataTable*

**evalml.pipelines.components.DecisionTreeRegressor.save**

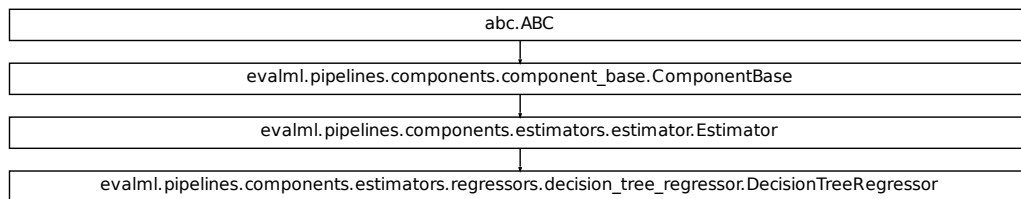
`DecisionTreeRegressor.save(file_path, pickle_protocol=5)`

Saves component at file path

**Parameters**

- **file\_path** (*str*) – Location to save file
- **pickle\_protocol** (*int*) – The pickle data stream format.

**Returns** None

**Class Inheritance****evalml.pipelines.components.LightGBMRegressor**

```

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,
                                                    n_jobs=-1, random_seed=0,
                                                    bagging_fraction=0.9, bagging_freq=0, **kwargs)

```

LightGBM Regressor

**name** = 'LightGBM Regressor'

**model\_family** = 'lightgbm'

**supported\_problem\_types** = [<ProblemTypes.REGRESSION: 'regression'>]

**hyperparameter\_ranges** = {'bagging\_fraction': Real(low=1e-06, high=1, prior='uniform',

**default\_parameters** = {'bagging\_fraction': 0.9, 'bagging\_freq': 0, 'boosting\_type': 'gb

**predict\_uses\_y** = False

### Instance attributes

SEED_MAX	
SEED_MIN	
feature_importance	Returns importance associated with each feature.
needs_fitting	
parameters	Returns the parameters which were used to initialize the component

### Methods:

<code>__init__</code>	Initialize self.
<code>clone</code>	Constructs a new component with the same parameters and random state.
<code>describe</code>	Describe a component and its parameters
<code>fit</code>	Fits component to data
<code>load</code>	Loads component at file path
<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

### `evalml.pipelines.components.LightGBMRegressor.__init__`

`LightGBMRegressor.__init__(boosting_type='gbdt', learning_rate=0.1, n_estimators=20, max_depth=0, num_leaves=31, min_child_samples=20, n_jobs=-1, random_seed=0, bagging_fraction=0.9, bagging_freq=0, **kwargs)`  
Initialize self. See `help(type(self))` for accurate signature.

### `evalml.pipelines.components.LightGBMRegressor.clone`

`LightGBMRegressor.clone()`  
Constructs a new component with the same parameters and random state.

**Returns** A new instance of this component with identical parameters and random state.

**evalml.pipelines.components.LightGBMRegressor.describe**`LightGBMRegressor.describe` (*print\_name=False, return\_dict=False*)

Describe a component and its parameters

**Parameters**

- **print\_name** (*bool, optional*) – whether to print name of component
- **return\_dict** (*bool, optional*) – whether to return description as dictionary in the format {"name": name, "parameters": parameters}

**Returns** prints and returns dictionary**Return type** None or dict**evalml.pipelines.components.LightGBMRegressor.fit**`LightGBMRegressor.fit` (*X, y=None*)

Fits component to data

**Parameters**

- **X** (*list, ww.DataTable, pd.DataFrame or np.ndarray*) – The input training data of shape [n\_samples, n\_features]
- **y** (*list, ww.DataColumn, pd.Series, np.ndarray, optional*) – The target training data of length [n\_samples]

**Returns** self**evalml.pipelines.components.LightGBMRegressor.load****static** `LightGBMRegressor.load` (*file\_path*)

Loads component at file path

**Parameters** **file\_path** (*str*) – Location to load file**Returns** ComponentBase object**evalml.pipelines.components.LightGBMRegressor.predict**`LightGBMRegressor.predict` (*X*)

Make predictions using selected features.

**Parameters** **X** (*ww.DataTable, pd.DataFrame, or np.ndarray*) – Data of shape [n\_samples, n\_features]**Returns** Predicted values**Return type** ww.DataColumn

### `evalml.pipelines.components.LightGBMRegressor.predict_proba`

`LightGBMRegressor.predict_proba(X)`

Make probability estimates for labels.

**Parameters** **X** (*ww.DataTable*, *pd.DataFrame*, or *np.ndarray*) – Features

**Returns** Probability estimates

**Return type** *ww.DataTable*

### `evalml.pipelines.components.LightGBMRegressor.save`

`LightGBMRegressor.save(file_path, pickle_protocol=5)`

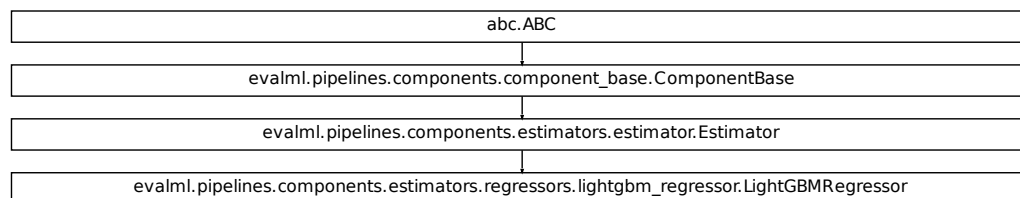
Saves component at file path

**Parameters**

- **file\_path** (*str*) – Location to save file
- **pickle\_protocol** (*int*) – The pickle data stream format.

**Returns** None

## Class Inheritance



### `evalml.pipelines.components.SVMRegressor`

```
class evalml.pipelines.components.SVMRegressor (C=1.0, kernel='rbf', gamma='scale',  
random_seed=0, **kwargs)
```

Support Vector Machine Regressor.

**name** = 'SVM Regressor'

**model\_family** = 'svm'

**supported\_problem\_types** = [`<ProblemTypes.REGRESSION: 'regression'>`, `<ProblemTypes.TIME`

`hyperparameter_ranges` = {'C': `Real(low=0, high=10, prior='uniform', transform='identit`

`default_parameters` = {'C': 1.0, 'gamma': 'scale', 'kernel': 'rbf'}

**predict\_uses\_y** = False



**Instance attributes**

<code>feature_importance</code>	Feature importance only works with linear kernels.
<code>needs_fitting</code>	
<code>parameters</code>	Returns the parameters which were used to initialize the component

**Methods:**

<code>__init__</code>	Initialize self.
<code>clone</code>	Constructs a new component with the same parameters and random state.
<code>describe</code>	Describe a component and its parameters
<code>fit</code>	Fits component to data
<code>load</code>	Loads component at file path
<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

**evalml.pipelines.components.SVMRegressor.\_\_init\_\_**

`SVMRegressor.__init__(C=1.0, kernel='rbf', gamma='scale', random_seed=0, **kwargs)`  
 Initialize self. See help(type(self)) for accurate signature.

**evalml.pipelines.components.SVMRegressor.clone**

`SVMRegressor.clone()`  
 Constructs a new component with the same parameters and random state.

**Returns** A new instance of this component with identical parameters and random state.

**evalml.pipelines.components.SVMRegressor.describe**

`SVMRegressor.describe(print_name=False, return_dict=False)`  
 Describe a component and its parameters

**Parameters**

- **print\_name** (*bool, optional*) – whether to print name of component
- **return\_dict** (*bool, optional*) – whether to return description as dictionary in the format {"name": name, "parameters": parameters}

**Returns** prints and returns dictionary

**Return type** None or dict

**evalml.pipelines.components.SVMRegressor.fit**

`SVMRegressor.fit(X, y=None)`

Fits component to data

**Parameters**

- **X** (*list*, *ww.DataTable*, *pd.DataFrame* or *np.ndarray*) – The input training data of shape [n\_samples, n\_features]
- **y** (*list*, *ww.DataColumn*, *pd.Series*, *np.ndarray*, *optional*) – The target training data of length [n\_samples]

**Returns** self

**evalml.pipelines.components.SVMRegressor.load**

**static** `SVMRegressor.load(file_path)`

Loads component at file path

**Parameters** **file\_path** (*str*) – Location to load file

**Returns** ComponentBase object

**evalml.pipelines.components.SVMRegressor.predict**

`SVMRegressor.predict(X)`

Make predictions using selected features.

**Parameters** **X** (*ww.DataTable*, *pd.DataFrame*, or *np.ndarray*) – Data of shape [n\_samples, n\_features]

**Returns** Predicted values

**Return type** *ww.DataColumn*

**evalml.pipelines.components.SVMRegressor.predict\_proba**

`SVMRegressor.predict_proba(X)`

Make probability estimates for labels.

**Parameters** **X** (*ww.DataTable*, *pd.DataFrame*, or *np.ndarray*) – Features

**Returns** Probability estimates

**Return type** *ww.DataTable*

**evalml.pipelines.components.SVMRegressor.save**

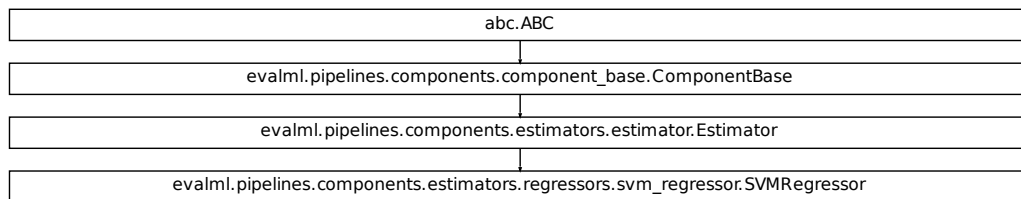
`SVMRegressor.save(file_path, pickle_protocol=5)`

Saves component at file path

**Parameters**

- **file\_path** (*str*) – Location to save file
- **pickle\_protocol** (*int*) – The pickle data stream format.

**Returns** None

**Class Inheritance**

## 5.7 Model Understanding

### 5.7.1 Utility Methods

<code>confusion_matrix</code>	Confusion matrix for binary and multiclass classification.
<code>normalize_confusion_matrix</code>	Normalizes a confusion matrix.
<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.
<code>calculate_permutation_importance</code>	Calculates permutation importance for features.
<code>binary_objective_vs_threshold</code>	Computes objective score as a function of potential binary classification
<code>get_prediction_vs_actual_over_time_data</code>	Get the data needed for the prediction_vs_actual_over_time plot.
<code>partial_dependence</code>	Calculates one or two-way partial dependence.
<code>get_prediction_vs_actual_data</code>	Combines y_true and y_pred into a single dataframe and adds a column for outliers.

continues on next page

Table 153 – continued from previous page

<code>get_linear_coefficients</code>	Returns a dataframe showing the features with the greatest predictive power for a linear model.
<code>t_sne</code>	Get the transformed output after fitting X to the embedded space using t-SNE.

**evalml.model\_understanding.confusion\_matrix**

`evalml.model_understanding.confusion_matrix(y_true, y_predicted, normalize_method='true')`

Confusion matrix for binary and multiclass classification.

**Parameters**

- **y\_true** (*ww.DataColumn, pd.Series or np.ndarray*) – True binary labels.
- **y\_pred** (*ww.DataColumn, 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.normalize\_confusion\_matrix**

`evalml.model_understanding.normalize_confusion_matrix(conf_mat, normalize_method='true')`

Normalizes a confusion matrix.

**Parameters**

- **conf\_mat** (*ww.DataTable, 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.precision\_recall\_curve

evalml.model\_understanding.**precision\_recall\_curve**(*y\_true*, *y\_pred\_proba*)

Given labels and binary classifier predicted probabilities, compute and return the data representing a precision-recall curve.

#### Parameters

- **y\_true** (*ww.DataColumn*, *pd.Series* or *np.ndarray*) – True binary labels.
- **y\_pred\_proba** (*ww.DataColumn*, *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.

#### 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

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** (*ww.DataColumn*, *pd.Series* or *np.ndarray*) – True labels.
- **y\_pred\_proba** (*ww.DataColumn*, *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.calculate\_permutation\_importance

`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** (*ww.DataTable*, *pd.DataFrame*) – The input data used to score and compute permutation importance
- **y** (*ww.DataColumn*, *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.
- **random\_seed** (*int*) – Seed for the random number generator. Defaults to 0.

**Returns** *pd.DataFrame*, Mean feature importance scores over 5 shuffles.

## evalml.model\_understanding.binary\_objective\_vs\_threshold

`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** (*ww.DataTable*, *pd.DataFrame*) – The input data used to compute objective score
- **y** (*ww.DataColumn*, *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.get\_prediction\_vs\_actual\_over\_time\_data

```
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** (`ww.DataTable`, `pd.DataFrame`) – Features used to generate new predictions.
- **y** (`ww.DataColumn`, `pd.Series`) – Target values to compare predictions against.
- **dates** (`ww.DataColumn`, `pd.Series`) – Dates corresponding to target values and predictions.

**Returns** `pd.DataFrame`

## evalml.model\_understanding.partial\_dependence

```
evalml.model_understanding.partial_dependence(pipeline, X, features, percentiles=(0.05,
                                                                                   0.95), grid_resolution=100)
```

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** (`ww.DataTable`, `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.

### Returns

**DataFrame with averaged predictions for all points in the grid averaged** over all samples of X and the values used to calculate those predictions.

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`

**Raises**

- **ValueError** – if the user provides a tuple of not exactly two features.
- **ValueError** – if the provided pipeline isn't fitted.
- **ValueError** – if the provided pipeline is a Baseline pipeline.
- **ValueError** – if any of the features passed in are completely NaN
- **ValueError** – 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.get_prediction_vs_actual_data`

`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`, `ww.DataColumn`, or `np.ndarray`) – The real target values of the data
- **y\_pred** (`pd.Series`, `ww.DataColumn`, 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_linear_coefficients`

`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.t\_sne

`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, ww.DataTable, 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)

## 5.7.2 Graph Utility Methods

<code>graph_precision_recall_curve</code>	Generate and display a precision-recall plot.
<code>graph_roc_curve</code>	Generate and display a Receiver Operating Characteristic (ROC) plot for binary and multiclass classification problems.
<code>graph_confusion_matrix</code>	Generate and display a confusion matrix plot.
<code>graph_permutation_importance</code>	Generate a bar graph of the pipeline's permutation importance.
<code>graph_binary_objective_vs_threshold</code>	Generates a plot graphing objective score vs.
<code>graph_prediction_vs_actual</code>	Generate a scatter plot comparing the true and predicted values.
<code>graph_prediction_vs_actual_over_time</code>	Plot the target values and predictions against time on the x-axis.
<code>graph_partial_dependence</code>	Create an one-way or two-way partial dependence plot.
<code>graph_t_sne</code>	Plot high dimensional data into lower dimensional space using t-SNE .

## evalml.model\_understanding.graph\_precision\_recall\_curve

`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** (ww.DataColumn, pd.Series or np.ndarray) – True binary labels.
- **y\_pred\_proba** (ww.DataColumn, 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\_roc\_curve

```
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** (*ww.DataColumn, pd.Series or np.ndarray*) – True labels.
- **y\_pred\_proba** (*ww.DataColumn, 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\_confusion\_matrix

```
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** (*ww.DataColumn, pd.Series or np.ndarray*) – True binary labels.
- **y\_pred** (*ww.DataColumn, 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\_permutation\_importance

```
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** (*ww.DataTable, pd.DataFrame*) – The input data used to score and compute permutation importance

- **y** (*ww.DataColumn*, *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\_binary\_objective\_vs\_threshold

`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** (*ww.DataTable*, *pd.DataFrame*) – The input data used to score and compute scores
- **y** (*ww.DataColumn*, *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\_prediction\_vs\_actual

`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** (*ww.DataColumn*, *pd.Series*) – The real target values of the data
- **y\_pred** (*ww.DataColumn*, *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

`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** (`ww.DataTable`, `pd.DataFrame`) – Features used to generate new predictions.
- **y** (`ww.DataColumn`, `pd.Series`) – Target values to compare predictions against.
- **dates** (`ww.DataColumn`, `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\_partial\_dependence

`evalml.model_understanding.graph_partial_dependence` (*pipeline*, *X*, *features*,  
*class\_label=None*,  
*grid\_resolution=100*)

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** (`ww.DataTable`, `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

**Returns** figure object containing the partial dependence data for plotting

**Return type** `plotly.graph_objects.Figure`

**Raises** **ValueError** – if a graph is requested for a class name that isn't present in the pipeline

## evalml.model\_understanding.graph\_t\_sne

```
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*, *ww.DataTable*) – 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

## 5.7.3 Prediction Explanations

<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.

**evalml.model\_understanding.prediction\_explanations.explain\_predictions**

```
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,  
                                                                    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** (*ww.DataTable, pd.DataFrame*) – Dataframe of input data to evaluate the pipeline on.
- **y** (*ww.DataColumn, 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.
- **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.explain\_predictions\_best\_worst**

```
evalml.model_understanding.prediction_explanations.explain_predictions_best_worst (pipeline,
in-
put_features
y_true,
num_to_expl
top_k_featur
in-
clude_shap_
met-
ric=None,
out-
put_format=
```

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** (`ww.DataTable`, `pd.DataFrame`) – Input data to evaluate the pipeline on.
- **y\_true** (`ww.DataColumn`, `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”.

**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.

## 5.8 Objective Functions

### 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.

#### **evalml.objectives.ObjectiveBase**

**class** evalml.objectives.ObjectiveBase

Base class for all objectives.

**problem\_types** = None

#### **Methods**

<i>__init__</i>	Initialize self.
<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>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.

#### **evalml.objectives.ObjectiveBase.\_\_init\_\_**

ObjectiveBase.\_\_init\_\_()

Initialize self. See help(type(self)) for accurate signature.

#### **evalml.objectives.ObjectiveBase.calculate\_percent\_difference**

**classmethod** ObjectiveBase.calculate\_percent\_difference(*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

### `evalml.objectives.ObjectiveBase.is_defined_for_problem_type`

**classmethod** `ObjectiveBase.is_defined_for_problem_type(problem_type)`

### `evalml.objectives.ObjectiveBase.objective_function`

**abstract classmethod** `ObjectiveBase.objective_function(y_true, y_predicted, X=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

**Returns** Numerical value used to calculate score

### `evalml.objectives.ObjectiveBase.score`

`ObjectiveBase.score(y_true, y_predicted, X=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

**Returns** score

### `evalml.objectives.ObjectiveBase.validate_inputs`

`ObjectiveBase.validate_inputs(y_true, y_predicted)`

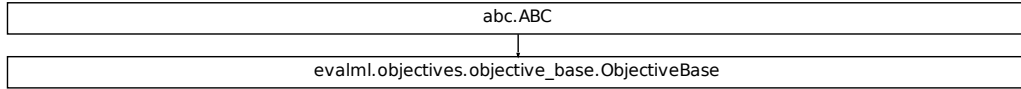
Validates the input based on a few simple checks.

#### Parameters

- **y\_predicted** (`ww.DataColumn`, `ww.DataTable`, `pd.Series`, or `pd.DataFrame`) – Predicted values of length `[n_samples]`
- **y\_true** (`ww.DataColumn`, `pd.Series`) – Actual class labels of length `[n_samples]`

**Returns** None

## Class Inheritance



### evalml.objectives.BinaryClassificationObjective

**class** evalml.objectives.BinaryClassificationObjective

Base class for all binary classification objectives.

**problem\_types** = [<ProblemTypes.BINARY: 'binary'>, <ProblemTypes.TIME\_SERIES\_BINARY: 't

#### Methods

<code>__init__</code>	Initialize self.
<code>calculate_percent_difference</code>	Calculate the percent difference between scores.
<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>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.

#### evalml.objectives.BinaryClassificationObjective.\_\_init\_\_

BinaryClassificationObjective.\_\_init\_\_()

Initialize self. See help(type(self)) for accurate signature.

**evalml.objectives.BinaryClassificationObjective.calculate\_percent\_difference**

**classmethod** BinaryClassificationObjective.**calculate\_percent\_difference** (*score*,  
*base-*  
*line\_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

**evalml.objectives.BinaryClassificationObjective.decision\_function**

BinaryClassificationObjective.**decision\_function** (*ypred\_proba*, *threshold=0.5*,  
*X=None*)

Apply a learned threshold to predicted probabilities to get predicted classes.

**Parameters**

- **ypred\_proba** (*ww.DataColumn*, *pd.Series*, *np.ndarray*) – The classifier's predicted probabilities
- **threshold** (*float*, *optional*) – Threshold used to make a prediction. Defaults to 0.5.
- **X** (*ww.DataTable*, *pd.DataFrame*, *optional*) – Any extra columns that are needed from training data.

**Returns** predictions

**evalml.objectives.BinaryClassificationObjective.is\_defined\_for\_problem\_type**

**classmethod** BinaryClassificationObjective.**is\_defined\_for\_problem\_type** (*problem\_type*)

**evalml.objectives.BinaryClassificationObjective.objective\_function**

**abstract classmethod** BinaryClassificationObjective.**objective\_function** (*y\_true*,  
*y\_predicted*,  
*X=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

**Returns** Numerical value used to calculate score

### `evalml.objectives.BinaryClassificationObjective.optimize_threshold`

`BinaryClassificationObjective.optimize_threshold(ypred_proba, y_true, X=None)`

Learn a binary classification threshold which optimizes the current objective.

#### **Parameters**

- **ypred\_proba** (*ww.DataColumn*, *pd.Series*) – The classifier’s predicted probabilities
- **y\_true** (*ww.DataColumn*, *pd.Series*) – The ground truth for the predictions.
- **X** (*ww.DataTable*, *pd.DataFrame*, *optional*) – Any extra columns that are needed from training data.

**Returns** Optimal threshold for this objective

### `evalml.objectives.BinaryClassificationObjective.score`

`BinaryClassificationObjective.score(y_true, y_predicted, X=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

**Returns** score

### `evalml.objectives.BinaryClassificationObjective.validate_inputs`

`BinaryClassificationObjective.validate_inputs(y_true, y_predicted)`

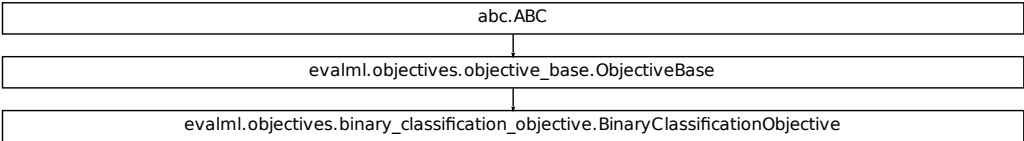
Validates the input based on a few simple checks.

#### **Parameters**

- **y\_predicted** (*ww.DataColumn*, *ww.DataTable*, *pd.Series*, or *pd.DataFrame*) – Predicted values of length [n\_samples]
- **y\_true** (*ww.DataColumn*, *pd.Series*) – Actual class labels of length [n\_samples]

**Returns** None

Class Inheritance



evalml.objectives.MulticlassClassificationObjective

**class** evalml.objectives.MulticlassClassificationObjective

Base class for all multiclass classification objectives.

**problem\_types** = [<ProblemTypes.MULTICLASS: 'multiclass'>, <ProblemTypes.TIME\_SERIES\_MULTICLASS: 'multiclass\_time\_series'>]

Methods

<code>__init__</code>	Initialize self.
<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>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.

evalml.objectives.MulticlassClassificationObjective.\_\_init\_\_

MulticlassClassificationObjective.\_\_init\_\_() Initialize self. See help(type(self)) for accurate signature.

### `evalml.objectives.MulticlassClassificationObjective.calculate_percent_difference`

**classmethod** `MulticlassClassificationObjective.calculate_percent_difference` (*score*,  
*base-*  
*line\_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`

### `evalml.objectives.MulticlassClassificationObjective.is_defined_for_problem_type`

**classmethod** `MulticlassClassificationObjective.is_defined_for_problem_type` (*problem\_type*)

### `evalml.objectives.MulticlassClassificationObjective.objective_function`

**abstract classmethod** `MulticlassClassificationObjective.objective_function` (*y\_true*,  
*y\_predicted*,  
*X=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

**Returns** Numerical value used to calculate score

### `evalml.objectives.MulticlassClassificationObjective.score`

`MulticlassClassificationObjective.score` (*y\_true*, *y\_predicted*, *X=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

**Returns** `score`

**evalml.objectives.MulticlassClassificationObjective.validate\_inputs**

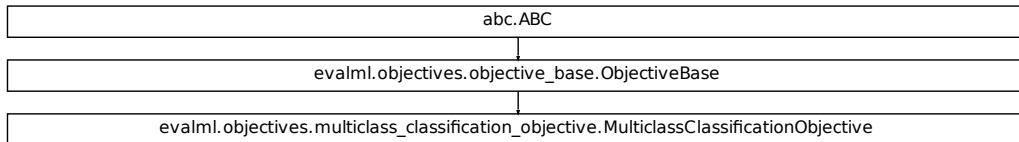
`MulticlassClassificationObjective.validate_inputs(y_true, y_predicted)`

Validates the input based on a few simple checks.

**Parameters**

- **y\_predicted** (`ww.DataColumn`, `ww.DataTable`, `pd.Series`, or `pd.DataFrame`) – Predicted values of length [n\_samples]
- **y\_true** (`ww.DataColumn`, `pd.Series`) – Actual class labels of length [n\_samples]

**Returns** None

**Class Inheritance****evalml.objectives.RegressionObjective**

**class** `evalml.objectives.RegressionObjective`

Base class for all regression objectives.

**problem\_types** = [`<ProblemTypes.REGRESSION: 'regression'>`, `<ProblemTypes.TIME_SERIES_REGRESSION: 'time_series_regression'>`]

**Methods**

<code>__init__</code>	Initialize self.
<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>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.

### `evalml.objectives.RegressionObjective.__init__`

`RegressionObjective.__init__()`

Initialize self. See `help(type(self))` for accurate signature.

### `evalml.objectives.RegressionObjective.calculate_percent_difference`

**classmethod** `RegressionObjective.calculate_percent_difference` (*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

### `evalml.objectives.RegressionObjective.is_defined_for_problem_type`

**classmethod** `RegressionObjective.is_defined_for_problem_type` (*problem\_type*)

### `evalml.objectives.RegressionObjective.objective_function`

**abstract classmethod** `RegressionObjective.objective_function` (*y\_true*,  
*y\_predicted*,  
*X=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

**Returns** Numerical value used to calculate score



**evalml.objectives.RegressionObjective.score**

`RegressionObjective.score(y_true, y_predicted, X=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

**Returns** score

**evalml.objectives.RegressionObjective.validate\_inputs**

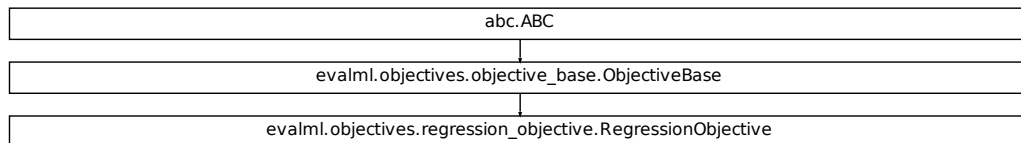
`RegressionObjective.validate_inputs(y_true, y_predicted)`

Validates the input based on a few simple checks.

**Parameters**

- **y\_predicted** (*ww.DataColumn*, *ww.DataTable*, *pd.Series*, or *pd.DataFrame*) – Predicted values of length [n\_samples]
- **y\_true** (*ww.DataColumn*, *pd.Series*) – Actual class labels of length [n\_samples]

**Returns** None

**Class Inheritance**

## 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.

### evalml.objectives.FraudCost

```
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.

    name = 'Fraud Cost'
    greater_is_better = False
    perfect_score = 0.0
    positive_only = False
    problem_types = [<ProblemTypes.BINARY: 'binary'>, <ProblemTypes.TIME_SERIES_BINARY: 't
    score_needs_proba = False
```

#### Methods

<i>__init__</i>	Create instance of FraudCost
<i>calculate_percent_difference</i>	Calculate the percent difference between scores.
<i>decision_function</i>	Determine if a transaction is fraud given predicted probabilities, threshold, and dataframe with transaction amount.
<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>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.

**evalml.objectives.FraudCost.\_\_init\_\_**

`FraudCost.__init__` (*retry\_percentage=0.5, interchange\_fee=0.02, fraud\_payout\_percentage=1.0, amount\_col='amount'*)  
Create instance of FraudCost

**Parameters**

- **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”

**evalml.objectives.FraudCost.calculate\_percent\_difference**

**classmethod** `FraudCost.calculate_percent_difference` (*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

**evalml.objectives.FraudCost.decision\_function**

`FraudCost.decision_function` (*ypred\_proba, threshold=0.0, X=None*)  
Determine if a transaction is fraud given predicted probabilities, threshold, and dataframe with transaction amount.

**Parameters**

- **ypred\_proba** (*ww.DataColumn, pd.Series*) – Predicted probabilities
- **threshold** (*float*) – Dollar threshold to determine if transaction is fraud
- **X** (*ww.DataTable, pd.DataFrame*) – Data containing transaction amounts

**Returns** pd.Series of predicted fraud labels using X and threshold

**Return type** pd.Series

**evalml.objectives.FraudCost.is\_defined\_for\_problem\_type**

**classmethod** `FraudCost.is_defined_for_problem_type(problem_type)`

**evalml.objectives.FraudCost.objective\_function**

`FraudCost.objective_function(y_true, y_predicted, X)`

Calculate amount lost to fraud per transaction given predictions, true values, and dataframe with transaction amount.

**Parameters**

- **y\_predicted** (*ww.DataColumn*, *pd.Series*) – Predicted fraud labels
- **y\_true** (*ww.DataColumn*, *pd.Series*) – True fraud labels
- **X** (*ww.DataTable*, *pd.DataFrame*) – Data with transaction amounts

**Returns** Amount lost to fraud per transaction

**Return type** float

**evalml.objectives.FraudCost.optimize\_threshold**

`FraudCost.optimize_threshold(ypred_proba, y_true, X=None)`

Learn a binary classification threshold which optimizes the current objective.

**Parameters**

- **ypred\_proba** (*ww.DataColumn*, *pd.Series*) – The classifier’s predicted probabilities
- **y\_true** (*ww.DataColumn*, *pd.Series*) – The ground truth for the predictions.
- **X** (*ww.DataTable*, *pd.DataFrame*, *optional*) – Any extra columns that are needed from training data.

**Returns** Optimal threshold for this objective

**evalml.objectives.FraudCost.score**

`FraudCost.score(y_true, y_predicted, X=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

**Returns** score

**evalml.objectives.FraudCost.validate\_inputs**

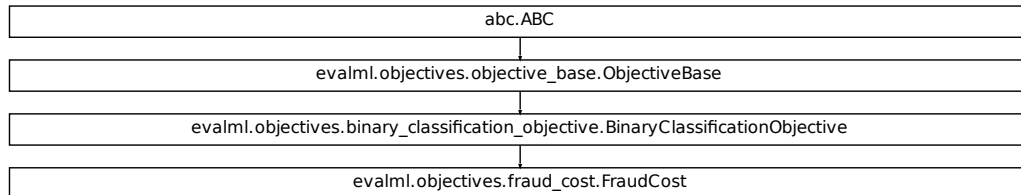
`FraudCost.validate_inputs(y_true, y_predicted)`

Validates the input based on a few simple checks.

**Parameters**

- **y\_predicted** (*ww.DataColumn*, *ww.DataTable*, *pd.Series*, or *pd.DataFrame*) – Predicted values of length [n\_samples]
- **y\_true** (*ww.DataColumn*, *pd.Series*) – Actual class labels of length [n\_samples]

**Returns** None

**Class Inheritance****evalml.objectives.LeadScoring**

**class** `evalml.objectives.LeadScoring(true_positives=1, false_positives=-1)`

Lead scoring.

**name** = 'Lead Scoring'

**greater\_is\_better** = True

**perfect\_score** = inf

**positive\_only** = False

**problem\_types** = [<ProblemTypes.BINARY: 'binary'>, <ProblemTypes.TIME\_SERIES\_BINARY: 't

**score\_needs\_proba** = False

## Methods

<code>__init__</code>	Create instance.
<code>calculate_percent_difference</code>	Calculate the percent difference between scores.
<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>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.

### `evalml.objectives.LeadScoring.__init__`

`LeadScoring.__init__(true_positives=1, false_positives=-1)`  
Create instance.

#### Parameters

- **true\_positives** (*int*) – Reward for a true positive
- **false\_positives** (*int*) – Cost for a false positive. Should be negative.

### `evalml.objectives.LeadScoring.calculate_percent_difference`

**classmethod** `LeadScoring.calculate_percent_difference(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

**evalml.objectives.LeadScoring.decision\_function**

`LeadScoring.decision_function(ypred_proba, threshold=0.5, X=None)`

Apply a learned threshold to predicted probabilities to get predicted classes.

**Parameters**

- **ypred\_proba** (*ww.DataColumn, pd.Series, np.ndarray*) – The classifier’s predicted probabilities
- **threshold** (*float, optional*) – Threshold used to make a prediction. Defaults to 0.5.
- **X** (*ww.DataTable, pd.DataFrame, optional*) – Any extra columns that are needed from training data.

**Returns** predictions

**evalml.objectives.LeadScoring.is\_defined\_for\_problem\_type**

**classmethod** `LeadScoring.is_defined_for_problem_type(problem_type)`

**evalml.objectives.LeadScoring.objective\_function**

`LeadScoring.objective_function(y_true, y_predicted, X=None)`

Calculate the profit per lead.

**Parameters**

- **y\_predicted** (*ww.DataColumn, pd.Series*) – Predicted labels
- **y\_true** (*ww.DataColumn, pd.Series*) – True labels
- **X** (*ww.DataTable, pd.DataFrame*) – Ignored.

**Returns** Profit per lead

**Return type** float

**evalml.objectives.LeadScoring.optimize\_threshold**

`LeadScoring.optimize_threshold(ypred_proba, y_true, X=None)`

Learn a binary classification threshold which optimizes the current objective.

**Parameters**

- **ypred\_proba** (*ww.DataColumn, pd.Series*) – The classifier’s predicted probabilities
- **y\_true** (*ww.DataColumn, pd.Series*) – The ground truth for the predictions.
- **X** (*ww.DataTable, pd.DataFrame, optional*) – Any extra columns that are needed from training data.

**Returns** Optimal threshold for this objective

### evalml.objectives.LeadScoring.score

LeadScoring.**score**(*y\_true*, *y\_predicted*, *X=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

**Returns** score

### evalml.objectives.LeadScoring.validate\_inputs

LeadScoring.**validate\_inputs**(*y\_true*, *y\_predicted*)

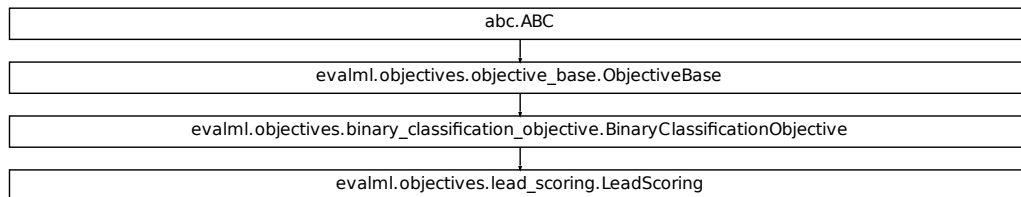
Validates the input based on a few simple checks.

#### Parameters

- **y\_predicted** (*ww.DataColumn*, *ww.DataTable*, *pd.Series*, or *pd.DataFrame*) – Predicted values of length [n\_samples]
- **y\_true** (*ww.DataColumn*, *pd.Series*) – Actual class labels of length [n\_samples]

**Returns** None

## Class Inheritance





## evalml.objectives.CostBenefitMatrix

```
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.

```
name = 'Cost Benefit Matrix'
```

```
greater_is_better = True
```

```
perfect_score = inf
```

```
positive_only = False
```

```
problem_types = [<ProblemTypes.BINARY: 'binary'>, <ProblemTypes.TIME_SERIES_BINARY: 't
```

```
score_needs_proba = False
```

### Methods

<code>__init__</code>	Create instance of CostBenefitMatrix.
<code>calculate_percent_difference</code>	Calculate the percent difference between scores.
<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>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.

### evalml.objectives.CostBenefitMatrix.\_\_init\_\_

```
CostBenefitMatrix.__init__(true_positive, true_negative, false_positive, false_negative)
```

Create instance of CostBenefitMatrix.

#### 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

### `evalml.objectives.CostBenefitMatrix.calculate_percent_difference`

**classmethod** `CostBenefitMatrix.calculate_percent_difference` (*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`

### `evalml.objectives.CostBenefitMatrix.decision_function`

`CostBenefitMatrix.decision_function` (*ypred\_proba*, *threshold=0.5*, *X=None*)

Apply a learned threshold to predicted probabilities to get predicted classes.

#### Parameters

- **ypred\_proba** (*ww.DataColumn*, *pd.Series*, *np.ndarray*) – The classifier's predicted probabilities
- **threshold** (*float*, *optional*) – Threshold used to make a prediction. Defaults to 0.5.
- **X** (*ww.DataTable*, *pd.DataFrame*, *optional*) – Any extra columns that are needed from training data.

**Returns** `predictions`

### `evalml.objectives.CostBenefitMatrix.is_defined_for_problem_type`

**classmethod** `CostBenefitMatrix.is_defined_for_problem_type` (*problem\_type*)

### `evalml.objectives.CostBenefitMatrix.objective_function`

`CostBenefitMatrix.objective_function` (*y\_true*, *y\_predicted*, *X=None*)

Calculates cost-benefit of the using the predicted and true values.

#### Parameters

- **y\_predicted** (*pd.Series*, *ww.DataColumn*) – Predicted labels
- **y\_true** (*pd.Series*, *ww.DataColumn*) – True labels
- **X** (*pd.DataFrame*, *ww.DataTable*) – Ignored.

**Returns** `Cost-benefit matrix score`

**Return type** `float`

**evalml.objectives.CostBenefitMatrix.optimize\_threshold**

`CostBenefitMatrix.optimize_threshold(ypred_proba, y_true, X=None)`

Learn a binary classification threshold which optimizes the current objective.

**Parameters**

- **ypred\_proba** (*ww.DataColumn, pd.Series*) – The classifier’s predicted probabilities
- **y\_true** (*ww.DataColumn, pd.Series*) – The ground truth for the predictions.
- **X** (*ww.DataTable, pd.DataFrame, optional*) – Any extra columns that are needed from training data.

**Returns** Optimal threshold for this objective

**evalml.objectives.CostBenefitMatrix.score**

`CostBenefitMatrix.score(y_true, y_predicted, X=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

**Returns** score

**evalml.objectives.CostBenefitMatrix.validate\_inputs**

`CostBenefitMatrix.validate_inputs(y_true, y_predicted)`

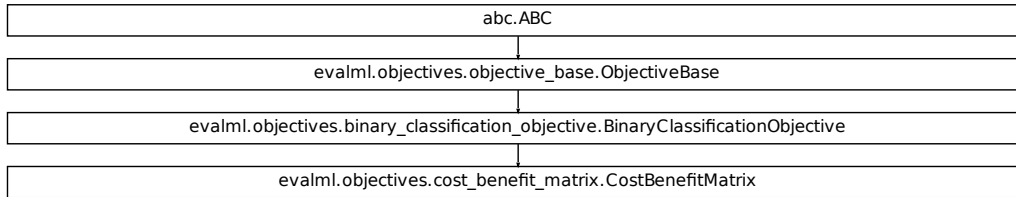
Validates the input based on a few simple checks.

**Parameters**

- **y\_predicted** (*ww.DataColumn, ww.DataTable, pd.Series, or pd.DataFrame*) – Predicted values of length [n\_samples]
- **y\_true** (*ww.DataColumn, pd.Series*) – Actual class labels of length [n\_samples]

**Returns** None

## Class Inheritance



### 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>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.

continues on next page

Table 165 – continued from previous page

<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.

**evalml.objectives.AccuracyBinary**

**class** evalml.objectives.**AccuracyBinary**

Accuracy score for binary classification.

**name** = 'Accuracy Binary'

**greater\_is\_better** = True

**perfect\_score** = 1.0

**positive\_only** = False

**problem\_types** = [<ProblemTypes.BINARY: 'binary'>, <ProblemTypes.TIME\_SERIES\_BINARY: 't

**score\_needs\_proba** = False

**Methods**

<i>__init__</i>	Initialize self.
<i>calculate_percent_difference</i>	Calculate the percent difference between scores.
<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>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.

### `evalml.objectives.AccuracyBinary.__init__`

`AccuracyBinary.__init__()`  
Initialize self. See `help(type(self))` for accurate signature.

### `evalml.objectives.AccuracyBinary.calculate_percent_difference`

**classmethod** `AccuracyBinary.calculate_percent_difference(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`

### `evalml.objectives.AccuracyBinary.decision_function`

`AccuracyBinary.decision_function(ypred_proba, threshold=0.5, X=None)`  
Apply a learned threshold to predicted probabilities to get predicted classes.

#### Parameters

- **ypred\_proba** (*ww.DataColumn, pd.Series, np.ndarray*) – The classifier’s predicted probabilities
- **threshold** (*float, optional*) – Threshold used to make a prediction. Defaults to 0.5.
- **X** (*ww.DataTable, pd.DataFrame, optional*) – Any extra columns that are needed from training data.

**Returns** `predictions`

### `evalml.objectives.AccuracyBinary.is_defined_for_problem_type`

**classmethod** `AccuracyBinary.is_defined_for_problem_type(problem_type)`

**evalml.objectives.AccuracyBinary.objective\_function**

`AccuracyBinary.objective_function(y_true, y_predicted, X=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

**Returns** Numerical value used to calculate score

**evalml.objectives.AccuracyBinary.optimize\_threshold**

`AccuracyBinary.optimize_threshold(ypred_proba, y_true, X=None)`

Learn a binary classification threshold which optimizes the current objective.

**Parameters**

- **ypred\_proba** (`ww.DataColumn`, `pd.Series`) – The classifier’s predicted probabilities
- **y\_true** (`ww.DataColumn`, `pd.Series`) – The ground truth for the predictions.
- **X** (`ww.DataTable`, `pd.DataFrame`, *optional*) – Any extra columns that are needed from training data.

**Returns** Optimal threshold for this objective

**evalml.objectives.AccuracyBinary.score**

`AccuracyBinary.score(y_true, y_predicted, X=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

**Returns** score

**evalml.objectives.AccuracyBinary.validate\_inputs**

`AccuracyBinary.validate_inputs(y_true, y_predicted)`

Validates the input based on a few simple checks.

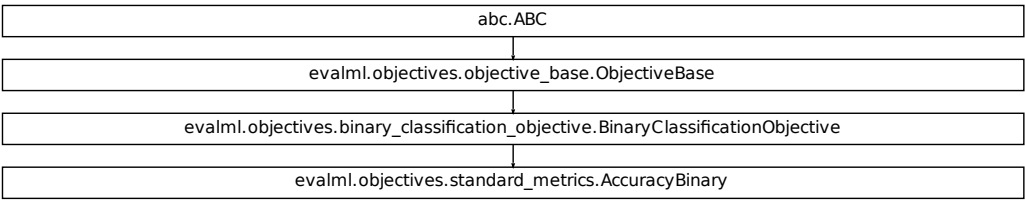
**Parameters**

- **y\_predicted** (`ww.DataColumn`, `ww.DataTable`, `pd.Series`, or `pd.DataFrame`) – Predicted values of length `[n_samples]`

- **y\_true** (*ww.DataColumn*, *pd.Series*) – Actual class labels of length [n\_samples]

**Returns** None

Class Inheritance



evalml.objectives.AccuracyMulticlass

```
class evalml.objectives.AccuracyMulticlass
    Accuracy score for multiclass classification.

    name = 'Accuracy Multiclass'
    greater_is_better = True
    perfect_score = 1.0
    positive_only = False
    problem_types = [<ProblemTypes.MULTICLASS: 'multiclass'>, <ProblemTypes.TIME_SERIES_MULTICLASS: 'multiclass'>]
    score_needs_proba = False
```

Methods

<code>__init__</code>	Initialize self.
<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>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.



**evalml.objectives.AccuracyMulticlass.\_\_init\_\_**

`AccuracyMulticlass.__init__()`

Initialize self. See help(type(self)) for accurate signature.

**evalml.objectives.AccuracyMulticlass.calculate\_percent\_difference**

**classmethod** `AccuracyMulticlass.calculate_percent_difference(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

**evalml.objectives.AccuracyMulticlass.is\_defined\_for\_problem\_type**

**classmethod** `AccuracyMulticlass.is_defined_for_problem_type(problem_type)`

**evalml.objectives.AccuracyMulticlass.objective\_function**

`AccuracyMulticlass.objective_function(y_true, y_predicted, X=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

**Returns** Numerical value used to calculate score

**evalml.objectives.AccuracyMulticlass.score**

`AccuracyMulticlass.score(y_true, y_predicted, X=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

**Returns** score

### `evalml.objectives.AccuracyMulticlass.validate_inputs`

`AccuracyMulticlass.validate_inputs(y_true, y_predicted)`

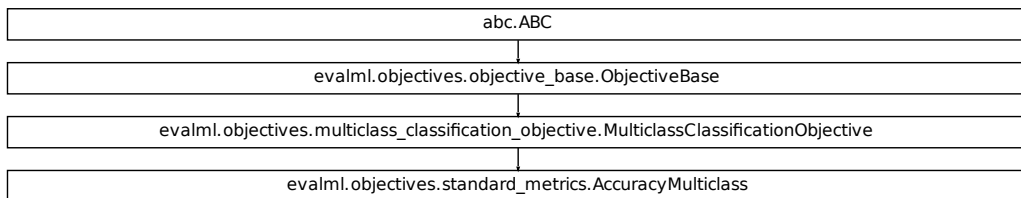
Validates the input based on a few simple checks.

#### Parameters

- **y\_predicted** (*ww.DataColumn*, *ww.DataTable*, *pd.Series*, or *pd.DataFrame*) – Predicted values of length [n\_samples]
- **y\_true** (*ww.DataColumn*, *pd.Series*) – Actual class labels of length [n\_samples]

**Returns** None

### Class Inheritance



### `evalml.objectives.AUC`

**class** `evalml.objectives.AUC`

AUC score for binary classification.

**name** = 'AUC'

**greater\_is\_better** = True

**perfect\_score** = 1.0

**positive\_only** = False

**problem\_types** = [<ProblemTypes.BINARY: 'binary'>, <ProblemTypes.TIME\_SERIES\_BINARY: 't

**score\_needs\_proba** = True

## Methods

<code>__init__</code>	Initialize self.
<code>calculate_percent_difference</code>	Calculate the percent difference between scores.
<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>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.

### `evalml.objectives.AUC.__init__`

`AUC.__init__()`

Initialize self. See `help(type(self))` for accurate signature.

### `evalml.objectives.AUC.calculate_percent_difference`

**classmethod** `AUC.calculate_percent_difference(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`

### `evalml.objectives.AUC.decision_function`

`AUC.decision_function(ypred_proba, threshold=0.5, X=None)`

Apply a learned threshold to predicted probabilities to get predicted classes.

#### Parameters

- **ypred\_proba** (*ww.DataColumn, pd.Series, np.ndarray*) – The classifier's predicted probabilities

- **threshold** (*float, optional*) – Threshold used to make a prediction. Defaults to 0.5.
- **X** (*ww.DataTable, pd.DataFrame, optional*) – Any extra columns that are needed from training data.

**Returns** predictions

### `evalml.objectives.AUC.is_defined_for_problem_type`

**classmethod** `AUC.is_defined_for_problem_type` (*problem\_type*)

### `evalml.objectives.AUC.objective_function`

`AUC.objective_function` (*y\_true, y\_predicted, X=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

**Returns** Numerical value used to calculate score

### `evalml.objectives.AUC.optimize_threshold`

`AUC.optimize_threshold` (*ypred\_proba, y\_true, X=None*)

**Learn a binary classification threshold which optimizes the current objective.**

#### **Parameters**

- **ypred\_proba** (*ww.DataColumn, pd.Series*) – The classifier’s predicted probabilities
- **y\_true** (*ww.DataColumn, pd.Series*) – The ground truth for the predictions.
- **X** (*ww.DataTable, pd.DataFrame, optional*) – Any extra columns that are needed from training data.

**Returns** Optimal threshold for this objective

### `evalml.objectives.AUC.score`

`AUC.score` (*y\_true, y\_predicted, X=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

**Returns** score

### evalml.objectives.AUC.validate\_inputs

**AUC.validate\_inputs** (*y\_true*, *y\_predicted*)

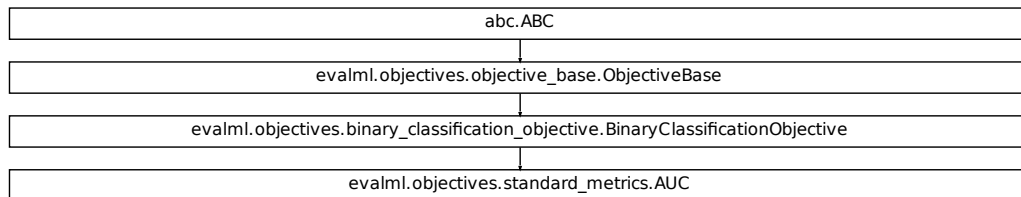
Validates the input based on a few simple checks.

#### Parameters

- **y\_predicted** (*ww.DataColumn*, *ww.DataTable*, *pd.Series*, or *pd.DataFrame*) – Predicted values of length [n\_samples]
- **y\_true** (*ww.DataColumn*, *pd.Series*) – Actual class labels of length [n\_samples]

**Returns** None

### Class Inheritance



### evalml.objectives.AUCMacro

**class** evalml.objectives.**AUCMacro**

AUC score for multiclass classification using macro averaging.

**name** = 'AUC Macro'

**greater\_is\_better** = True

**perfect\_score** = 1.0

**positive\_only** = False

**problem\_types** = [<ProblemTypes.MULTICLASS: 'multiclass'>, <ProblemTypes.TIME\_SERIES\_MULTICLASS: 'multiclass'>]

**score\_needs\_proba** = True

## Methods

<code>__init__</code>	Initialize self.
<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>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.

### `evalml.objectives.AUCMacro.__init__`

`AUCMacro.__init__()`

Initialize self. See help(type(self)) for accurate signature.

### `evalml.objectives.AUCMacro.calculate_percent_difference`

**classmethod** `AUCMacro.calculate_percent_difference` (*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

### `evalml.objectives.AUCMacro.is_defined_for_problem_type`

**classmethod** `AUCMacro.is_defined_for_problem_type` (*problem\_type*)

**evalml.objectives.AUCMacro.objective\_function**

`AUCMacro.objective_function(y_true, y_predicted, X=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

**Returns** Numerical value used to calculate score

**evalml.objectives.AUCMacro.score**

`AUCMacro.score(y_true, y_predicted, X=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

**Returns** score

**evalml.objectives.AUCMacro.validate\_inputs**

`AUCMacro.validate_inputs(y_true, y_predicted)`

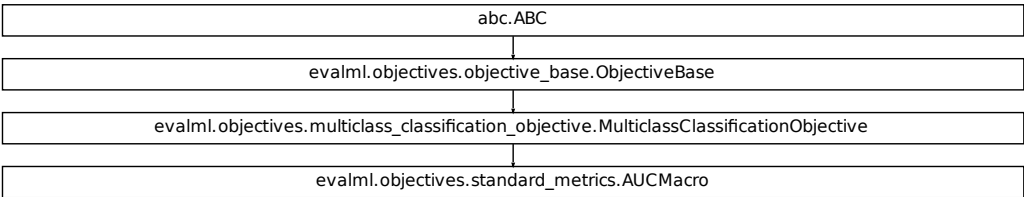
Validates the input based on a few simple checks.

**Parameters**

- **y\_predicted** (`ww.DataColumn`, `ww.DataTable`, `pd.Series`, or `pd.DataFrame`) – Predicted values of length `[n_samples]`
- **y\_true** (`ww.DataColumn`, `pd.Series`) – Actual class labels of length `[n_samples]`

**Returns** None

Class Inheritance



evalml.objectives.AUCMicro

```
class evalml.objectives.AUCMicro
    AUC score for multiclass classification using micro averaging.

    name = 'AUC Micro'
    greater_is_better = True
    perfect_score = 1.0
    positive_only = False
    problem_types = [<ProblemTypes.MULTICLASS: 'multiclass'>, <ProblemTypes.TIME_SERIES_MULTICLASS: 'multiclass_time_series'>]
    score_needs_proba = True
```

Methods

<code>__init__</code>	Initialize self.
<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>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.



**evalml.objectives.AUCMicro.\_\_init\_\_****AUCMicro.\_\_init\_\_()**

Initialize self. See help(type(self)) for accurate signature.

**evalml.objectives.AUCMicro.calculate\_percent\_difference****classmethod AUCMicro.calculate\_percent\_difference** (*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**evalml.objectives.AUCMicro.is\_defined\_for\_problem\_type****classmethod AUCMicro.is\_defined\_for\_problem\_type** (*problem\_type*)**evalml.objectives.AUCMicro.objective\_function****AUCMicro.objective\_function** (*y\_true, y\_predicted, X=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

**Returns** Numerical value used to calculate score**evalml.objectives.AUCMicro.score****AUCMicro.score** (*y\_true, y\_predicted, X=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

**Returns** score

### `evalml.objectives.AUCMicro.validate_inputs`

`AUCMicro.validate_inputs(y_true, y_predicted)`

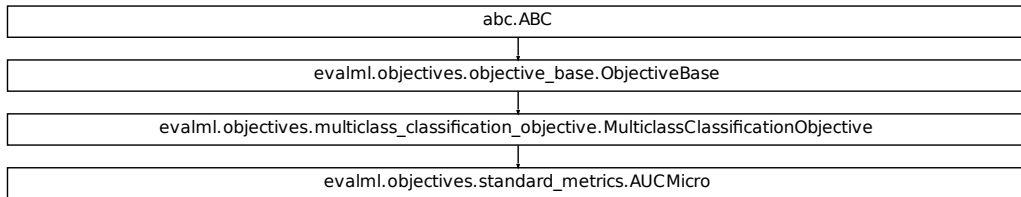
Validates the input based on a few simple checks.

#### Parameters

- **y\_predicted** (*ww.DataColumn, ww.DataTable, pd.Series, or pd.DataFrame*) – Predicted values of length [n\_samples]
- **y\_true** (*ww.DataColumn, pd.Series*) – Actual class labels of length [n\_samples]

**Returns** None

### Class Inheritance



### `evalml.objectives.AUCWeighted`

**class** `evalml.objectives.AUCWeighted`

AUC Score for multiclass classification using weighted averaging.

**name** = 'AUC Weighted'

**greater\_is\_better** = True

**perfect\_score** = 1.0

**positive\_only** = False

**problem\_types** = [`<ProblemTypes.MULTICLASS: 'multiclass'>`, `<ProblemTypes.TIME_SERIES_MULTICLASS: 'multiclass'>`]

**score\_needs\_proba** = True

## Methods

<code>__init__</code>	Initialize self.
<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>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.

### `evalml.objectives.AUCWeighted.__init__`

`AUCWeighted.__init__()`

Initialize self. See `help(type(self))` for accurate signature.

### `evalml.objectives.AUCWeighted.calculate_percent_difference`

**classmethod** `AUCWeighted.calculate_percent_difference(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`

### `evalml.objectives.AUCWeighted.is_defined_for_problem_type`

**classmethod** `AUCWeighted.is_defined_for_problem_type(problem_type)`

### `evalml.objectives.AUCWeighted.objective_function`

`AUCWeighted.objective_function` (*y\_true*, *y\_predicted*, *X=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

**Returns** Numerical value used to calculate score

### `evalml.objectives.AUCWeighted.score`

`AUCWeighted.score` (*y\_true*, *y\_predicted*, *X=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

**Returns** score

### `evalml.objectives.AUCWeighted.validate_inputs`

`AUCWeighted.validate_inputs` (*y\_true*, *y\_predicted*)

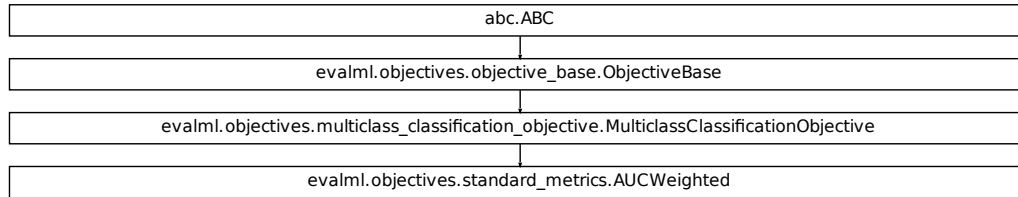
Validates the input based on a few simple checks.

#### Parameters

- ***y\_predicted*** (`ww.DataColumn`, `ww.DataTable`, `pd.Series`, or `pd.DataFrame`) – Predicted values of length [*n\_samples*]
- ***y\_true*** (`ww.DataColumn`, `pd.Series`) – Actual class labels of length [*n\_samples*]

**Returns** None

## Class Inheritance



### evalml.objectives.BalancedAccuracyBinary

**class** evalml.objectives.BalancedAccuracyBinary

Balanced accuracy score for binary classification.

**name** = 'Balanced Accuracy Binary'

**greater\_is\_better** = True

**perfect\_score** = 1.0

**positive\_only** = False

**problem\_types** = [<ProblemTypes.BINARY: 'binary'>, <ProblemTypes.TIME\_SERIES\_BINARY: 't

**score\_needs\_proba** = False

#### Methods

<code>__init__</code>	Initialize self.
<code>calculate_percent_difference</code>	Calculate the percent difference between scores.
<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>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.

### `evalml.objectives.BalancedAccuracyBinary.__init__`

`BalancedAccuracyBinary.__init__()`

Initialize self. See `help(type(self))` for accurate signature.

### `evalml.objectives.BalancedAccuracyBinary.calculate_percent_difference`

**classmethod** `BalancedAccuracyBinary.calculate_percent_difference` (*score*,  
*base-*  
*line\_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`

### `evalml.objectives.BalancedAccuracyBinary.decision_function`

`BalancedAccuracyBinary.decision_function` (*ypred\_proba*, *threshold=0.5*, *X=None*)

Apply a learned threshold to predicted probabilities to get predicted classes.

#### Parameters

- **ypred\_proba** (*ww.DataColumn*, *pd.Series*, *np.ndarray*) – The classifier’s predicted probabilities
- **threshold** (*float*, *optional*) – Threshold used to make a prediction. Defaults to 0.5.
- **X** (*ww.DataTable*, *pd.DataFrame*, *optional*) – Any extra columns that are needed from training data.

**Returns** `predictions`

### `evalml.objectives.BalancedAccuracyBinary.is_defined_for_problem_type`

**classmethod** `BalancedAccuracyBinary.is_defined_for_problem_type` (*problem\_type*)

**evalml.objectives.BalancedAccuracyBinary.objective\_function**

BalancedAccuracyBinary.**objective\_function**(*y\_true*, *y\_predicted*, *X=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

**Returns** Numerical value used to calculate score

**evalml.objectives.BalancedAccuracyBinary.optimize\_threshold**

BalancedAccuracyBinary.**optimize\_threshold**(*ypred\_proba*, *y\_true*, *X=None*)

Learn a binary classification threshold which optimizes the current objective.

**Parameters**

- **ypred\_proba** (*ww.DataColumn*, *pd.Series*) – The classifier’s predicted probabilities
- **y\_true** (*ww.DataColumn*, *pd.Series*) – The ground truth for the predictions.
- **X** (*ww.DataTable*, *pd.DataFrame*, *optional*) – Any extra columns that are needed from training data.

**Returns** Optimal threshold for this objective

**evalml.objectives.BalancedAccuracyBinary.score**

BalancedAccuracyBinary.**score**(*y\_true*, *y\_predicted*, *X=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

**Returns** score

**evalml.objectives.BalancedAccuracyBinary.validate\_inputs**

BalancedAccuracyBinary.**validate\_inputs**(*y\_true*, *y\_predicted*)

Validates the input based on a few simple checks.

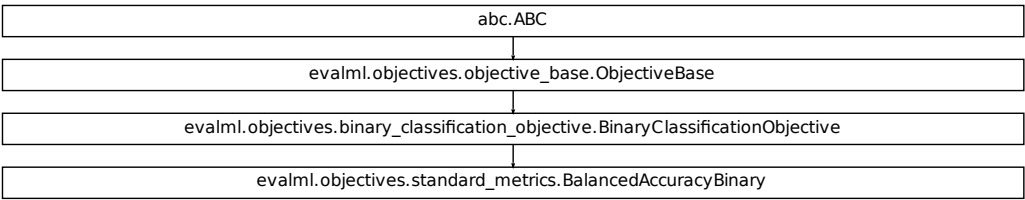
**Parameters**

- **y\_predicted** (*ww.DataColumn*, *ww.DataTable*, *pd.Series*, or *pd.DataFrame*) – Predicted values of length [n\_samples]

- **y\_true** (*ww.DataColumn*, *pd.Series*) – Actual class labels of length [n\_samples]

**Returns** None

Class Inheritance



evalml.objectives.BalancedAccuracyMulticlass

```
class evalml.objectives.BalancedAccuracyMulticlass
    Balanced accuracy score for multiclass classification.

    name = 'Balanced Accuracy Multiclass'

    greater_is_better = True

    perfect_score = 1.0

    positive_only = False

    problem_types = [<ProblemTypes.MULTICLASS: 'multiclass'>, <ProblemTypes.TIME_SERIES_MULTICLASS: 'multiclass'>]

    score_needs_proba = False
```

Methods

<code>__init__</code>	Initialize self.
<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>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.



**evalml.objectives.BalancedAccuracyMulticlass.\_\_init\_\_**

`BalancedAccuracyMulticlass.__init__()`  
 Initialize self. See help(type(self)) for accurate signature.

**evalml.objectives.BalancedAccuracyMulticlass.calculate\_percent\_difference**

**classmethod** `BalancedAccuracyMulticlass.calculate_percent_difference` (*score*,  
*base-*  
*line\_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

**evalml.objectives.BalancedAccuracyMulticlass.is\_defined\_for\_problem\_type**

**classmethod** `BalancedAccuracyMulticlass.is_defined_for_problem_type` (*problem\_type*)

**evalml.objectives.BalancedAccuracyMulticlass.objective\_function**

`BalancedAccuracyMulticlass.objective_function` (*y\_true*, *y\_predicted*, *X=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

**Returns** Numerical value used to calculate score

**evalml.objectives.BalancedAccuracyMulticlass.score**

`BalancedAccuracyMulticlass.score` (*y\_true*, *y\_predicted*, *X=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

**Returns** score

### `evalml.objectives.BalancedAccuracyMulticlass.validate_inputs`

`BalancedAccuracyMulticlass.validate_inputs` (*y\_true*, *y\_predicted*)

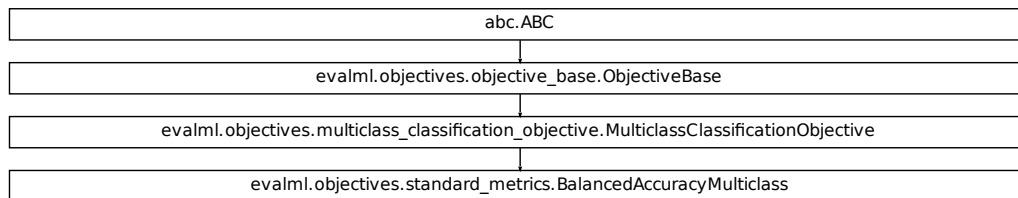
Validates the input based on a few simple checks.

#### Parameters

- **y\_predicted** (*ww.DataColumn*, *ww.DataTable*, *pd.Series*, or *pd.DataFrame*) – Predicted values of length [n\_samples]
- **y\_true** (*ww.DataColumn*, *pd.Series*) – Actual class labels of length [n\_samples]

**Returns** None

## Class Inheritance



### `evalml.objectives.F1`

**class** `evalml.objectives.F1`

F1 score for binary classification.

**name** = 'F1'

**greater\_is\_better** = True

**perfect\_score** = 1.0

**positive\_only** = False

**problem\_types** = [<ProblemTypes.BINARY: 'binary'>, <ProblemTypes.TIME\_SERIES\_BINARY: 't

**score\_needs\_proba** = False

## Methods

<code>__init__</code>	Initialize self.
<code>calculate_percent_difference</code>	Calculate the percent difference between scores.
<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>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.

### `evalml.objectives.F1.__init__`

`F1.__init__()`

Initialize self. See `help(type(self))` for accurate signature.

### `evalml.objectives.F1.calculate_percent_difference`

**classmethod** `F1.calculate_percent_difference(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`

### `evalml.objectives.F1.decision_function`

`F1.decision_function(ypred_proba, threshold=0.5, X=None)`

Apply a learned threshold to predicted probabilities to get predicted classes.

#### Parameters

- **ypred\_proba** (*ww.DataColumn, pd.Series, np.ndarray*) – The classifier’s predicted probabilities

- **threshold** (*float, optional*) – Threshold used to make a prediction. Defaults to 0.5.
- **X** (*ww.DataTable, pd.DataFrame, optional*) – Any extra columns that are needed from training data.

**Returns** predictions

### `evalml.objectives.F1.is_defined_for_problem_type`

**classmethod** `F1.is_defined_for_problem_type(problem_type)`

### `evalml.objectives.F1.objective_function`

`F1.objective_function(y_true, y_predicted, X=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

**Returns** Numerical value used to calculate score

### `evalml.objectives.F1.optimize_threshold`

`F1.optimize_threshold(ypred_proba, y_true, X=None)`

**Learn a binary classification threshold which optimizes the current objective.**

#### **Parameters**

- **ypred\_proba** (*ww.DataColumn, pd.Series*) – The classifier’s predicted probabilities
- **y\_true** (*ww.DataColumn, pd.Series*) – The ground truth for the predictions.
- **X** (*ww.DataTable, pd.DataFrame, optional*) – Any extra columns that are needed from training data.

**Returns** Optimal threshold for this objective

### `evalml.objectives.F1.score`

`F1.score(y_true, y_predicted, X=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

**Returns** score

### evalml.objectives.F1.validate\_inputs

**F1.validate\_inputs** (*y\_true*, *y\_predicted*)

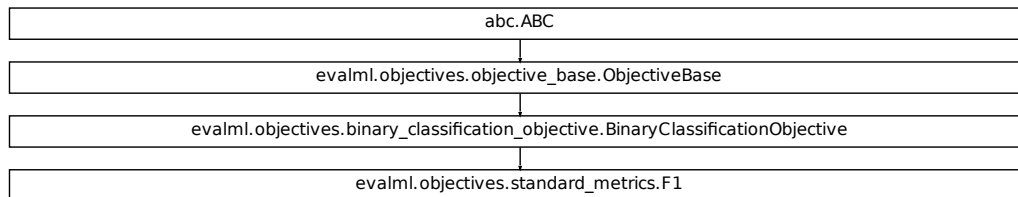
Validates the input based on a few simple checks.

#### Parameters

- **y\_predicted** (*ww.DataColumn*, *ww.DataTable*, *pd.Series*, or *pd.DataFrame*) – Predicted values of length [n\_samples]
- **y\_true** (*ww.DataColumn*, *pd.Series*) – Actual class labels of length [n\_samples]

**Returns** None

### Class Inheritance



### evalml.objectives.F1Micro

**class** evalml.objectives.F1Micro

F1 score for multiclass classification using micro averaging.

**name** = 'F1 Micro'

**greater\_is\_better** = True

**perfect\_score** = 1.0

**positive\_only** = False

**problem\_types** = [<ProblemTypes.MULTICLASS: 'multiclass'>, <ProblemTypes.TIME\_SERIES\_MULTICLASS: 'multiclass'>]

**score\_needs\_proba** = False

## Methods

<code>__init__</code>	Initialize self.
<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>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.

### `evalml.objectives.F1Micro.__init__`

`F1Micro.__init__()`

Initialize self. See `help(type(self))` for accurate signature.

### `evalml.objectives.F1Micro.calculate_percent_difference`

**classmethod** `F1Micro.calculate_percent_difference` (*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

### `evalml.objectives.F1Micro.is_defined_for_problem_type`

**classmethod** `F1Micro.is_defined_for_problem_type` (*problem\_type*)

**evalml.objectives.F1Micro.objective\_function**`F1Micro.objective_function(y_true, y_predicted, X=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

**Returns** Numerical value used to calculate score

**evalml.objectives.F1Micro.score**`F1Micro.score(y_true, y_predicted, X=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

**Returns** score

**evalml.objectives.F1Micro.validate\_inputs**`F1Micro.validate_inputs(y_true, y_predicted)`

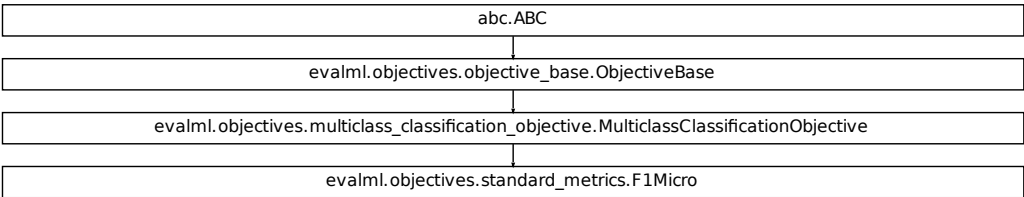
Validates the input based on a few simple checks.

**Parameters**

- **y\_predicted** (`ww.DataColumn`, `ww.DataTable`, `pd.Series`, or `pd.DataFrame`) – Predicted values of length `[n_samples]`
- **y\_true** (`ww.DataColumn`, `pd.Series`) – Actual class labels of length `[n_samples]`

**Returns** None

Class Inheritance



evalml.objectives.F1Macro

```
class evalml.objectives.F1Macro
    F1 score for multiclass classification using macro averaging.

    name = 'F1 Macro'
    greater_is_better = True
    perfect_score = 1.0
    positive_only = False
    problem_types = [<ProblemTypes.MULTICLASS: 'multiclass'>, <ProblemTypes.TIME_SERIES_MULTICLASS: 'multiclass_time_series'>]
    score_needs_proba = False
```

Methods

<code>__init__</code>	Initialize self.
<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>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.



**evalml.objectives.F1Macro.\_\_init\_\_****F1Macro.\_\_init\_\_()**

Initialize self. See help(type(self)) for accurate signature.

**evalml.objectives.F1Macro.calculate\_percent\_difference****classmethod F1Macro.calculate\_percent\_difference** (*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**evalml.objectives.F1Macro.is\_defined\_for\_problem\_type****classmethod F1Macro.is\_defined\_for\_problem\_type** (*problem\_type*)**evalml.objectives.F1Macro.objective\_function****F1Macro.objective\_function** (*y\_true, y\_predicted, X=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

**Returns** Numerical value used to calculate score**evalml.objectives.F1Macro.score****F1Macro.score** (*y\_true, y\_predicted, X=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

**Returns** score

### `evalml.objectives.F1Macro.validate_inputs`

`F1Macro.validate_inputs(y_true, y_predicted)`

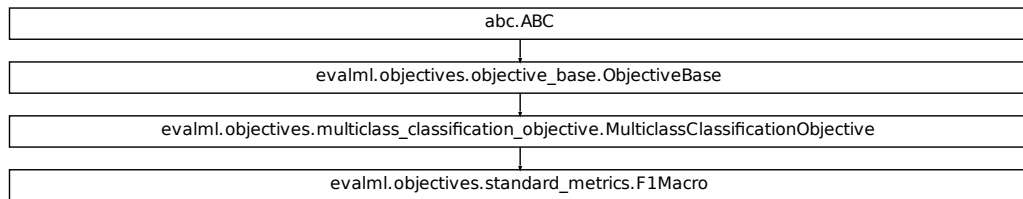
Validates the input based on a few simple checks.

#### Parameters

- **y\_predicted** (*ww.DataColumn, ww.DataTable, pd.Series, or pd.DataFrame*) – Predicted values of length [n\_samples]
- **y\_true** (*ww.DataColumn, pd.Series*) – Actual class labels of length [n\_samples]

**Returns** None

### Class Inheritance



### `evalml.objectives.F1Weighted`

**class** `evalml.objectives.F1Weighted`

F1 score for multiclass classification using weighted averaging.

**name** = 'F1 Weighted'

**greater\_is\_better** = True

**perfect\_score** = 1.0

**positive\_only** = False

**problem\_types** = [<ProblemTypes.MULTICLASS: 'multiclass'>, <ProblemTypes.TIME\_SERIES\_MULTICLASS: 'multiclass'>]

**score\_needs\_proba** = False

## Methods

<code>__init__</code>	Initialize self.
<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>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.

### `evalml.objectives.F1Weighted.__init__`

`F1Weighted.__init__()`

Initialize self. See `help(type(self))` for accurate signature.

### `evalml.objectives.F1Weighted.calculate_percent_difference`

**classmethod** `F1Weighted.calculate_percent_difference(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`

### `evalml.objectives.F1Weighted.is_defined_for_problem_type`

**classmethod** `F1Weighted.is_defined_for_problem_type(problem_type)`

### `evalml.objectives.F1Weighted.objective_function`

`F1Weighted.objective_function` (*y\_true*, *y\_predicted*, *X=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

**Returns** Numerical value used to calculate score

### `evalml.objectives.F1Weighted.score`

`F1Weighted.score` (*y\_true*, *y\_predicted*, *X=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

**Returns** score

### `evalml.objectives.F1Weighted.validate_inputs`

`F1Weighted.validate_inputs` (*y\_true*, *y\_predicted*)

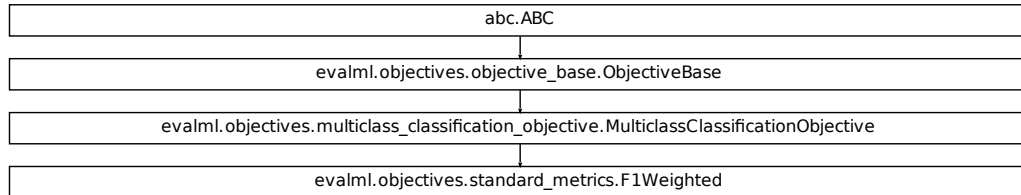
Validates the input based on a few simple checks.

#### Parameters

- ***y\_predicted*** (`ww.DataColumn`, `ww.DataTable`, `pd.Series`, or `pd.DataFrame`) – Predicted values of length [*n\_samples*]
- ***y\_true*** (`ww.DataColumn`, `pd.Series`) – Actual class labels of length [*n\_samples*]

**Returns** None

## Class Inheritance



### evalml.objectives.LogLossBinary

**class** evalml.objectives.LogLossBinary

Log Loss for binary classification.

**name** = 'Log Loss Binary'

**greater\_is\_better** = False

**perfect\_score** = 0.0

**positive\_only** = False

**problem\_types** = [<ProblemTypes.BINARY: 'binary'>, <ProblemTypes.TIME\_SERIES\_BINARY: 't

**score\_needs\_proba** = True

#### Methods

<code>__init__</code>	Initialize self.
<code>calculate_percent_difference</code>	Calculate the percent difference between scores.
<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>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.

### `evalml.objectives.LogLossBinary.__init__`

`LogLossBinary.__init__()`

Initialize self. See `help(type(self))` for accurate signature.

### `evalml.objectives.LogLossBinary.calculate_percent_difference`

**classmethod** `LogLossBinary.calculate_percent_difference(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`

### `evalml.objectives.LogLossBinary.decision_function`

`LogLossBinary.decision_function(ypred_proba, threshold=0.5, X=None)`

Apply a learned threshold to predicted probabilities to get predicted classes.

#### Parameters

- **ypred\_proba** (*ww.DataColumn, pd.Series, np.ndarray*) – The classifier's predicted probabilities
- **threshold** (*float, optional*) – Threshold used to make a prediction. Defaults to 0.5.
- **X** (*ww.DataTable, pd.DataFrame, optional*) – Any extra columns that are needed from training data.

**Returns** `predictions`

### `evalml.objectives.LogLossBinary.is_defined_for_problem_type`

**classmethod** `LogLossBinary.is_defined_for_problem_type(problem_type)`

**evalml.objectives.LogLossBinary.objective\_function**

`LogLossBinary.objective_function(y_true, y_predicted, X=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

**Returns** Numerical value used to calculate score

**evalml.objectives.LogLossBinary.optimize\_threshold**

`LogLossBinary.optimize_threshold(ypred_proba, y_true, X=None)`

Learn a binary classification threshold which optimizes the current objective.

**Parameters**

- **ypred\_proba** (`ww.DataColumn`, `pd.Series`) – The classifier’s predicted probabilities
- **y\_true** (`ww.DataColumn`, `pd.Series`) – The ground truth for the predictions.
- **X** (`ww.DataTable`, `pd.DataFrame`, *optional*) – Any extra columns that are needed from training data.

**Returns** Optimal threshold for this objective

**evalml.objectives.LogLossBinary.score**

`LogLossBinary.score(y_true, y_predicted, X=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

**Returns** score

**evalml.objectives.LogLossBinary.validate\_inputs**

`LogLossBinary.validate_inputs(y_true, y_predicted)`

Validates the input based on a few simple checks.

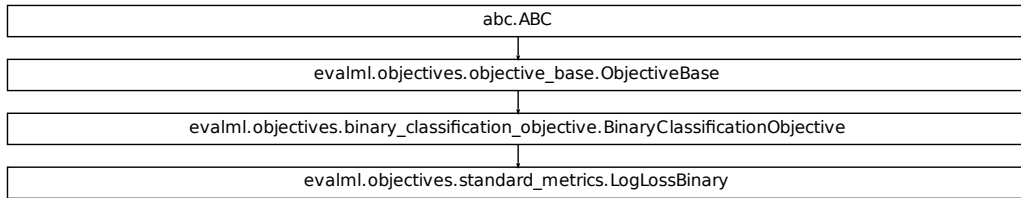
**Parameters**

- **y\_predicted** (`ww.DataColumn`, `ww.DataTable`, `pd.Series`, or `pd.DataFrame`) – Predicted values of length `[n_samples]`

- **y\_true** (*ww.DataColumn*, *pd.Series*) – Actual class labels of length [n\_samples]

**Returns** None

## Class Inheritance



## evalml.objectives.LogLossMulticlass

**class** evalml.objectives.LogLossMulticlass

Log Loss for multiclass classification.

**name** = 'Log Loss Multiclass'

**greater\_is\_better** = False

**perfect\_score** = 0.0

**positive\_only** = False

**problem\_types** = [<ProblemTypes.MULTICLASS: 'multiclass'>, <ProblemTypes.TIME\_SERIES\_MULTICLASS: 'multiclass\_time\_series'>]

**score\_needs\_proba** = True

## Methods

<code>__init__</code>	Initialize self.
<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>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.



**evalml.objectives.LogLossMulticlass.\_\_init\_\_**

`LogLossMulticlass.__init__()`

Initialize self. See help(type(self)) for accurate signature.

**evalml.objectives.LogLossMulticlass.calculate\_percent\_difference**

**classmethod** `LogLossMulticlass.calculate_percent_difference(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

**evalml.objectives.LogLossMulticlass.is\_defined\_for\_problem\_type**

**classmethod** `LogLossMulticlass.is_defined_for_problem_type(problem_type)`

**evalml.objectives.LogLossMulticlass.objective\_function**

`LogLossMulticlass.objective_function(y_true, y_predicted, X=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

**Returns** Numerical value used to calculate score

**evalml.objectives.LogLossMulticlass.score**

`LogLossMulticlass.score(y_true, y_predicted, X=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

**Returns** score

### `evalml.objectives.LogLossMulticlass.validate_inputs`

`LogLossMulticlass.validate_inputs(y_true, y_predicted)`

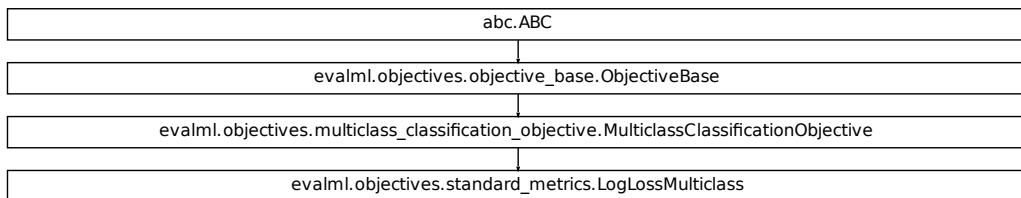
Validates the input based on a few simple checks.

#### Parameters

- **y\_predicted** (*ww.DataColumn*, *ww.DataTable*, *pd.Series*, or *pd.DataFrame*) – Predicted values of length [n\_samples]
- **y\_true** (*ww.DataColumn*, *pd.Series*) – Actual class labels of length [n\_samples]

**Returns** None

## Class Inheritance



### `evalml.objectives.MCCBinary`

**class** `evalml.objectives.MCCBinary`

Matthews correlation coefficient for binary classification.

`name = 'MCC Binary'`

`greater_is_better = True`

`perfect_score = 1.0`

`positive_only = False`

`problem_types = [<ProblemTypes.BINARY: 'binary'>, <ProblemTypes.TIME_SERIES_BINARY: 't`

`score_needs_proba = False`

## Methods

<code>__init__</code>	Initialize self.
<code>calculate_percent_difference</code>	Calculate the percent difference between scores.
<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>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.

### `evalml.objectives.MCCBinary.__init__`

`MCCBinary.__init__()`

Initialize self. See `help(type(self))` for accurate signature.

### `evalml.objectives.MCCBinary.calculate_percent_difference`

**classmethod** `MCCBinary.calculate_percent_difference` (*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`

### `evalml.objectives.MCCBinary.decision_function`

`MCCBinary.decision_function` (*ypred\_proba*, *threshold=0.5*, *X=None*)

Apply a learned threshold to predicted probabilities to get predicted classes.

#### Parameters

- **ypred\_proba** (*ww.DataColumn*, *pd.Series*, *np.ndarray*) – The classifier’s predicted probabilities

- **threshold** (*float, optional*) – Threshold used to make a prediction. Defaults to 0.5.
- **X** (*ww.DataTable, pd.DataFrame, optional*) – Any extra columns that are needed from training data.

**Returns** predictions

### `evalml.objectives.MCCBinary.is_defined_for_problem_type`

**classmethod** `MCCBinary.is_defined_for_problem_type` (*problem\_type*)

### `evalml.objectives.MCCBinary.objective_function`

`MCCBinary.objective_function` (*y\_true, y\_predicted, X=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

**Returns** Numerical value used to calculate score

### `evalml.objectives.MCCBinary.optimize_threshold`

`MCCBinary.optimize_threshold` (*ypred\_proba, y\_true, X=None*)

**Learn a binary classification threshold which optimizes the current objective.**

#### **Parameters**

- **ypred\_proba** (*ww.DataColumn, pd.Series*) – The classifier’s predicted probabilities
- **y\_true** (*ww.DataColumn, pd.Series*) – The ground truth for the predictions.
- **X** (*ww.DataTable, pd.DataFrame, optional*) – Any extra columns that are needed from training data.

**Returns** Optimal threshold for this objective

### `evalml.objectives.MCCBinary.score`

`MCCBinary.score` (*y\_true, y\_predicted, X=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

**Returns** score

### evalml.objectives.MCCBinary.validate\_inputs

MCCBinary.**validate\_inputs** (*y\_true*, *y\_predicted*)

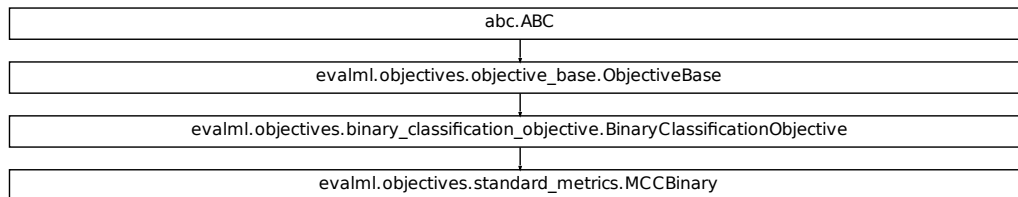
Validates the input based on a few simple checks.

#### Parameters

- **y\_predicted** (*ww.DataColumn*, *ww.DataTable*, *pd.Series*, or *pd.DataFrame*) – Predicted values of length [n\_samples]
- **y\_true** (*ww.DataColumn*, *pd.Series*) – Actual class labels of length [n\_samples]

**Returns** None

### Class Inheritance



### evalml.objectives.MCCMulticlass

**class** evalml.objectives.MCCMulticlass

Matthews correlation coefficient for multiclass classification.

**name** = 'MCC Multiclass'

**greater\_is\_better** = True

**perfect\_score** = 1.0

**positive\_only** = False

**problem\_types** = [<ProblemTypes.MULTICLASS: 'multiclass'>, <ProblemTypes.TIME\_SERIES\_MULTICLASS: 'multiclass\_time\_series'>]

**score\_needs\_proba** = False

## Methods

<code>__init__</code>	Initialize self.
<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>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.

### `evalml.objectives.MCCMulticlass.__init__`

`MCCMulticlass.__init__()`

Initialize self. See `help(type(self))` for accurate signature.

### `evalml.objectives.MCCMulticlass.calculate_percent_difference`

**classmethod** `MCCMulticlass.calculate_percent_difference` (*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`

### `evalml.objectives.MCCMulticlass.is_defined_for_problem_type`

**classmethod** `MCCMulticlass.is_defined_for_problem_type` (*problem\_type*)

**evalml.objectives.MCCMulticlass.objective\_function**

MCCMulticlass.**objective\_function**(*y\_true*, *y\_predicted*, *X=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

**Returns** Numerical value used to calculate score

**evalml.objectives.MCCMulticlass.score**

MCCMulticlass.**score**(*y\_true*, *y\_predicted*, *X=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

**Returns** score

**evalml.objectives.MCCMulticlass.validate\_inputs**

MCCMulticlass.**validate\_inputs**(*y\_true*, *y\_predicted*)

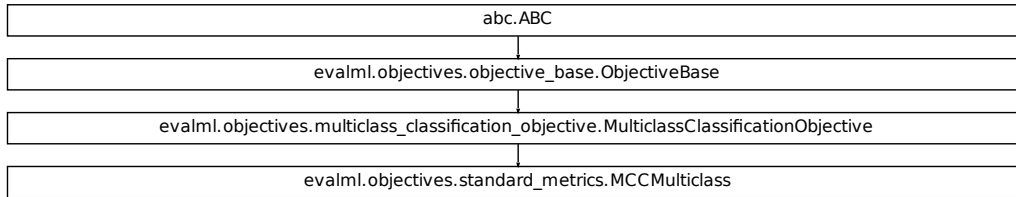
Validates the input based on a few simple checks.

**Parameters**

- **y\_predicted** (ww.DataColumn, ww.DataTable, pd.Series, or pd.DataFrame) – Predicted values of length [n\_samples]
- **y\_true** (ww.DataColumn, pd.Series) – Actual class labels of length [n\_samples]

**Returns** None

## Class Inheritance



### evalml.objectives.Precision

**class** evalml.objectives.Precision

Precision score for binary classification.

**name** = 'Precision'

**greater\_is\_better** = True

**perfect\_score** = 1.0

**positive\_only** = False

**problem\_types** = [<ProblemTypes.BINARY: 'binary'>, <ProblemTypes.TIME\_SERIES\_BINARY: 't

**score\_needs\_proba** = False

#### Methods

<code>__init__</code>	Initialize self.
<code>calculate_percent_difference</code>	Calculate the percent difference between scores.
<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>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.



**evalml.objectives.Precision.\_\_init\_\_****Precision.\_\_init\_\_()**

Initialize self. See help(type(self)) for accurate signature.

**evalml.objectives.Precision.calculate\_percent\_difference****classmethod Precision.calculate\_percent\_difference** (*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**evalml.objectives.Precision.decision\_function****Precision.decision\_function** (*ypred\_proba, threshold=0.5, X=None*)

Apply a learned threshold to predicted probabilities to get predicted classes.

**Parameters**

- **ypred\_proba** (*ww.DataColumn, pd.Series, np.ndarray*) – The classifier's predicted probabilities
- **threshold** (*float, optional*) – Threshold used to make a prediction. Defaults to 0.5.
- **X** (*ww.DataTable, pd.DataFrame, optional*) – Any extra columns that are needed from training data.

**Returns** predictions**evalml.objectives.Precision.is\_defined\_for\_problem\_type****classmethod Precision.is\_defined\_for\_problem\_type** (*problem\_type*)

### `evalml.objectives.Precision.objective_function`

`Precision.objective_function` (*y\_true*, *y\_predicted*, *X=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

**Returns** Numerical value used to calculate score

### `evalml.objectives.Precision.optimize_threshold`

`Precision.optimize_threshold` (*ypred\_proba*, *y\_true*, *X=None*)

Learn a binary classification threshold which optimizes the current objective.

#### Parameters

- ***ypred\_proba*** (*ww.DataColumn*, *pd.Series*) – The classifier’s predicted probabilities
- ***y\_true*** (*ww.DataColumn*, *pd.Series*) – The ground truth for the predictions.
- ***X*** (*ww.DataTable*, *pd.DataFrame*, *optional*) – Any extra columns that are needed from training data.

**Returns** Optimal threshold for this objective

### `evalml.objectives.Precision.score`

`Precision.score` (*y\_true*, *y\_predicted*, *X=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

**Returns** score

### `evalml.objectives.Precision.validate_inputs`

`Precision.validate_inputs` (*y\_true*, *y\_predicted*)

Validates the input based on a few simple checks.

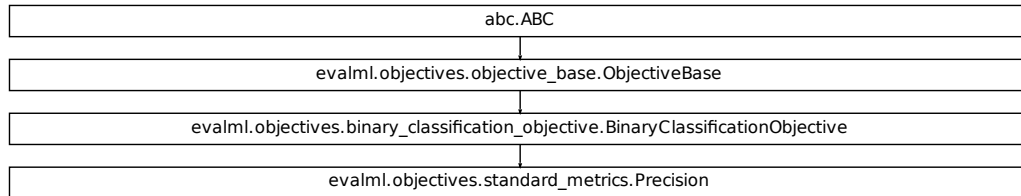
#### Parameters

- ***y\_predicted*** (*ww.DataColumn*, *ww.DataTable*, *pd.Series*, or *pd.DataFrame*) – Predicted values of length [*n\_samples*]

- **y\_true** (*ww.DataColumn*, *pd.Series*) – Actual class labels of length [n\_samples]

**Returns** None

## Class Inheritance



## evalml.objectives.PrecisionMicro

**class** evalml.objectives.PrecisionMicro

Precision score for multiclass classification using micro averaging.

**name** = 'Precision Micro'

**greater\_is\_better** = True

**perfect\_score** = 1.0

**positive\_only** = False

**problem\_types** = [<ProblemTypes.MULTICLASS: 'multiclass'>, <ProblemTypes.TIME\_SERIES\_MULTICLASS: 'multiclass'>]

**score\_needs\_proba** = False

## Methods

<code>__init__</code>	Initialize self.
<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>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.

### `evalml.objectives.PrecisionMicro.__init__`

`PrecisionMicro.__init__()`  
Initialize self. See `help(type(self))` for accurate signature.

### `evalml.objectives.PrecisionMicro.calculate_percent_difference`

**classmethod** `PrecisionMicro.calculate_percent_difference` (*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`

### `evalml.objectives.PrecisionMicro.is_defined_for_problem_type`

**classmethod** `PrecisionMicro.is_defined_for_problem_type` (*problem\_type*)

### `evalml.objectives.PrecisionMicro.objective_function`

`PrecisionMicro.objective_function` (*y\_true*, *y\_predicted*, *X=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

**Returns** Numerical value used to calculate score

### `evalml.objectives.PrecisionMicro.score`

`PrecisionMicro.score` (*y\_true*, *y\_predicted*, *X=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

**Returns** score

### evalml.objectives.PrecisionMicro.validate\_inputs

`PrecisionMicro.validate_inputs(y_true, y_predicted)`

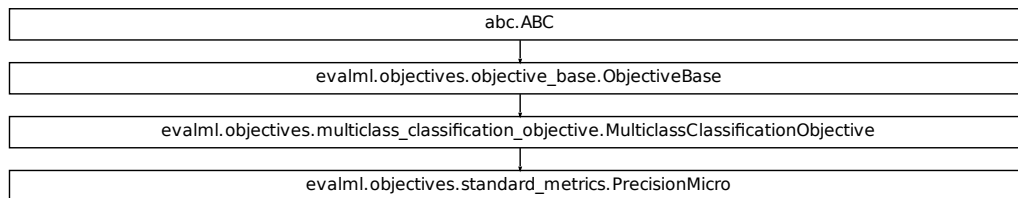
Validates the input based on a few simple checks.

#### Parameters

- **y\_predicted** (*ww.DataColumn*, *ww.DataTable*, *pd.Series*, or *pd.DataFrame*) – Predicted values of length [n\_samples]
- **y\_true** (*ww.DataColumn*, *pd.Series*) – Actual class labels of length [n\_samples]

**Returns** None

## Class Inheritance



### evalml.objectives.PrecisionMacro

**class** evalml.objectives.PrecisionMacro

Precision score for multiclass classification using macro averaging.

**name** = 'Precision Macro'

**greater\_is\_better** = True

**perfect\_score** = 1.0

**positive\_only** = False

**problem\_types** = [<ProblemTypes.MULTICLASS: 'multiclass'>, <ProblemTypes.TIME\_SERIES\_MULTICLASS: 'multiclass'>]

**score\_needs\_proba** = False

## Methods

<code>__init__</code>	Initialize self.
<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>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.

### `evalml.objectives.PrecisionMacro.__init__`

`PrecisionMacro.__init__()`  
Initialize self. See `help(type(self))` for accurate signature.

### `evalml.objectives.PrecisionMacro.calculate_percent_difference`

**classmethod** `PrecisionMacro.calculate_percent_difference` (*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`

### `evalml.objectives.PrecisionMacro.is_defined_for_problem_type`

**classmethod** `PrecisionMacro.is_defined_for_problem_type` (*problem\_type*)

**evalml.objectives.PrecisionMacro.objective\_function**

`PrecisionMacro.objective_function(y_true, y_predicted, X=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

**Returns** Numerical value used to calculate score

**evalml.objectives.PrecisionMacro.score**

`PrecisionMacro.score(y_true, y_predicted, X=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

**Returns** score

**evalml.objectives.PrecisionMacro.validate\_inputs**

`PrecisionMacro.validate_inputs(y_true, y_predicted)`

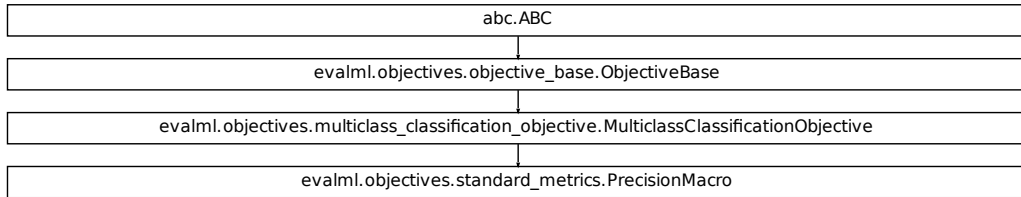
Validates the input based on a few simple checks.

**Parameters**

- **y\_predicted** (`ww.DataColumn`, `ww.DataTable`, `pd.Series`, or `pd.DataFrame`) – Predicted values of length `[n_samples]`
- **y\_true** (`ww.DataColumn`, `pd.Series`) – Actual class labels of length `[n_samples]`

**Returns** None

## Class Inheritance



### evalml.objectives.PrecisionWeighted

**class** evalml.objectives.PrecisionWeighted

Precision score for multiclass classification using weighted averaging.

**name** = 'Precision Weighted'

**greater\_is\_better** = True

**perfect\_score** = 1.0

**positive\_only** = False

**problem\_types** = [<ProblemTypes.MULTICLASS: 'multiclass'>, <ProblemTypes.TIME\_SERIES\_MULTICLASS: 'multiclass'>]

**score\_needs\_proba** = False

#### Methods

<code>__init__</code>	Initialize self.
<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>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.



**evalml.objectives.PrecisionWeighted.\_\_init\_\_**

`PrecisionWeighted.__init__()`

Initialize self. See `help(type(self))` for accurate signature.

**evalml.objectives.PrecisionWeighted.calculate\_percent\_difference**

**classmethod** `PrecisionWeighted.calculate_percent_difference(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

**evalml.objectives.PrecisionWeighted.is\_defined\_for\_problem\_type**

**classmethod** `PrecisionWeighted.is_defined_for_problem_type(problem_type)`

**evalml.objectives.PrecisionWeighted.objective\_function**

`PrecisionWeighted.objective_function(y_true, y_predicted, X=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

**Returns** Numerical value used to calculate score

**evalml.objectives.PrecisionWeighted.score**

`PrecisionWeighted.score(y_true, y_predicted, X=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

**Returns** score

### `evalml.objectives.PrecisionWeighted.validate_inputs`

`PrecisionWeighted.validate_inputs(y_true, y_predicted)`

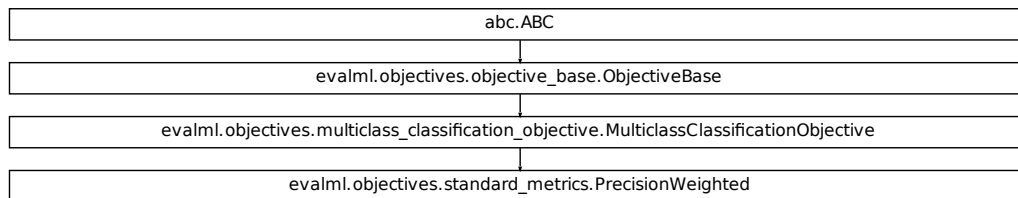
Validates the input based on a few simple checks.

#### Parameters

- **y\_predicted** (*ww.DataColumn*, *ww.DataTable*, *pd.Series*, or *pd.DataFrame*) – Predicted values of length [n\_samples]
- **y\_true** (*ww.DataColumn*, *pd.Series*) – Actual class labels of length [n\_samples]

**Returns** None

## Class Inheritance



### `evalml.objectives.Recall`

```
class evalml.objectives.Recall
```

```
    Recall score for binary classification.
```

```
    name = 'Recall'
```

```
    greater_is_better = True
```

```
    perfect_score = 1.0
```

```
    positive_only = False
```

```
    problem_types = [<ProblemTypes.BINARY: 'binary'>, <ProblemTypes.TIME_SERIES_BINARY: 't
```

```
    score_needs_proba = False
```

## Methods

<code>__init__</code>	Initialize self.
<code>calculate_percent_difference</code>	Calculate the percent difference between scores.
<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>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.

### `evalml.objectives.Recall.__init__`

`Recall.__init__()`

Initialize self. See `help(type(self))` for accurate signature.

### `evalml.objectives.Recall.calculate_percent_difference`

**classmethod** `Recall.calculate_percent_difference(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`

### `evalml.objectives.Recall.decision_function`

`Recall.decision_function(ypred_proba, threshold=0.5, X=None)`

Apply a learned threshold to predicted probabilities to get predicted classes.

#### Parameters

- **ypred\_proba** (*ww.DataColumn, pd.Series, np.ndarray*) – The classifier's predicted probabilities

- **threshold** (*float, optional*) – Threshold used to make a prediction. Defaults to 0.5.
- **X** (*ww.DataTable, pd.DataFrame, optional*) – Any extra columns that are needed from training data.

**Returns** predictions

### `evalml.objectives.Recall.is_defined_for_problem_type`

**classmethod** `Recall.is_defined_for_problem_type(problem_type)`

### `evalml.objectives.Recall.objective_function`

`Recall.objective_function(y_true, y_predicted, X=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

**Returns** Numerical value used to calculate score

### `evalml.objectives.Recall.optimize_threshold`

`Recall.optimize_threshold(ypred_proba, y_true, X=None)`

Learn a binary classification threshold which optimizes the current objective.

#### **Parameters**

- **ypred\_proba** (*ww.DataColumn, pd.Series*) – The classifier’s predicted probabilities
- **y\_true** (*ww.DataColumn, pd.Series*) – The ground truth for the predictions.
- **X** (*ww.DataTable, pd.DataFrame, optional*) – Any extra columns that are needed from training data.

**Returns** Optimal threshold for this objective

### `evalml.objectives.Recall.score`

`Recall.score(y_true, y_predicted, X=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

**Returns** score

### evalml.objectives.Recall.validate\_inputs

`Recall.validate_inputs(y_true, y_predicted)`

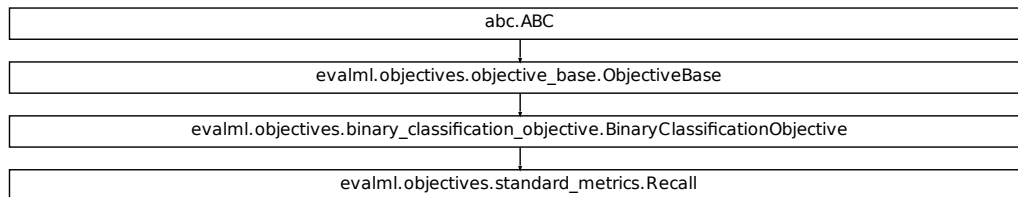
Validates the input based on a few simple checks.

#### Parameters

- **y\_predicted** (*ww.DataColumn, ww.DataTable, pd.Series, or pd.DataFrame*) – Predicted values of length [n\_samples]
- **y\_true** (*ww.DataColumn, pd.Series*) – Actual class labels of length [n\_samples]

**Returns** None

### Class Inheritance



### evalml.objectives.RecallMicro

**class** evalml.objectives.RecallMicro

Recall score for multiclass classification using micro averaging.

**name** = 'Recall Micro'

**greater\_is\_better** = True

**perfect\_score** = 1.0

**positive\_only** = False

**problem\_types** = [<ProblemTypes.MULTICLASS: 'multiclass'>, <ProblemTypes.TIME\_SERIES\_MULTICLASS: 'multiclass'>]

**score\_needs\_proba** = False

## Methods

<code>__init__</code>	Initialize self.
<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>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.

### `evalml.objectives.RecallMicro.__init__`

`RecallMicro.__init__()`

Initialize self. See `help(type(self))` for accurate signature.

### `evalml.objectives.RecallMicro.calculate_percent_difference`

**classmethod** `RecallMicro.calculate_percent_difference` (*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

### `evalml.objectives.RecallMicro.is_defined_for_problem_type`

**classmethod** `RecallMicro.is_defined_for_problem_type` (*problem\_type*)

**evalml.objectives.RecallMicro.objective\_function**

`RecallMicro.objective_function(y_true, y_predicted, X=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

**Returns** Numerical value used to calculate score

**evalml.objectives.RecallMicro.score**

`RecallMicro.score(y_true, y_predicted, X=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

**Returns** score

**evalml.objectives.RecallMicro.validate\_inputs**

`RecallMicro.validate_inputs(y_true, y_predicted)`

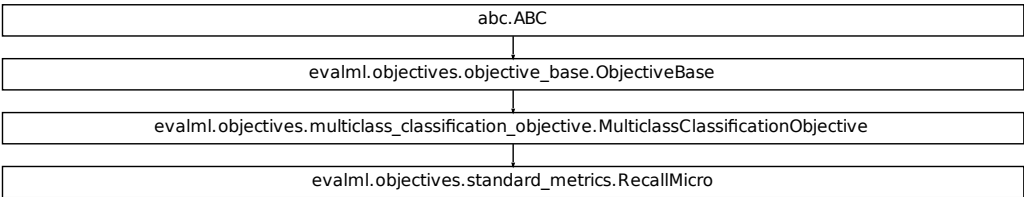
Validates the input based on a few simple checks.

**Parameters**

- **y\_predicted** (`ww.DataColumn`, `ww.DataTable`, `pd.Series`, or `pd.DataFrame`) – Predicted values of length `[n_samples]`
- **y\_true** (`ww.DataColumn`, `pd.Series`) – Actual class labels of length `[n_samples]`

**Returns** None

Class Inheritance



evalml.objectives.RecallMacro

```
class evalml.objectives.RecallMacro
    Recall score for multiclass classification using macro averaging.

    name = 'Recall Macro'
    greater_is_better = True
    perfect_score = 1.0
    positive_only = False
    problem_types = [<ProblemTypes.MULTICLASS: 'multiclass'>, <ProblemTypes.TIME_SERIES_MULTICLASS: 'multiclass'>]
    score_needs_proba = False
```

Methods

<code>__init__</code>	Initialize self.
<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>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.



**evalml.objectives.RecallMacro.\_\_init\_\_**`RecallMacro.__init__()`

Initialize self. See help(type(self)) for accurate signature.

**evalml.objectives.RecallMacro.calculate\_percent\_difference**`classmethod RecallMacro.calculate_percent_difference(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

**evalml.objectives.RecallMacro.is\_defined\_for\_problem\_type**`classmethod RecallMacro.is_defined_for_problem_type(problem_type)`**evalml.objectives.RecallMacro.objective\_function**`RecallMacro.objective_function(y_true, y_predicted, X=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

**Returns** Numerical value used to calculate score

**evalml.objectives.RecallMacro.score**`RecallMacro.score(y_true, y_predicted, X=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

**Returns** score

### `evalml.objectives.RecallMacro.validate_inputs`

`RecallMacro.validate_inputs(y_true, y_predicted)`

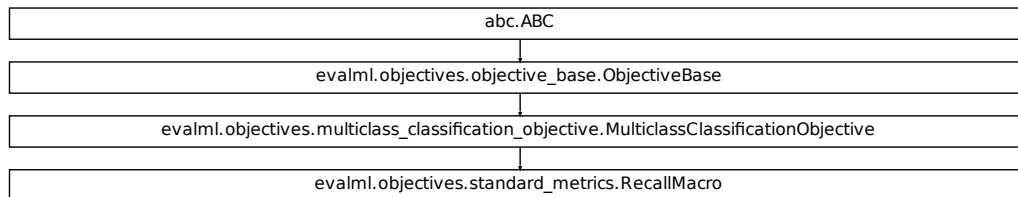
Validates the input based on a few simple checks.

#### Parameters

- **y\_predicted** (*ww.DataColumn, ww.DataTable, pd.Series, or pd.DataFrame*) – Predicted values of length [n\_samples]
- **y\_true** (*ww.DataColumn, pd.Series*) – Actual class labels of length [n\_samples]

**Returns** None

### Class Inheritance



### `evalml.objectives.RecallWeighted`

**class** `evalml.objectives.RecallWeighted`

Recall score for multiclass classification using weighted averaging.

**name** = 'Recall Weighted'

**greater\_is\_better** = True

**perfect\_score** = 1.0

**positive\_only** = False

**problem\_types** = [`<ProblemTypes.MULTICLASS: 'multiclass'>`, `<ProblemTypes.TIME_SERIES_MULTICLASS: 'multiclass'>`]

**score\_needs\_proba** = False

## Methods

<code>__init__</code>	Initialize self.
<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>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.

### `evalml.objectives.RecallWeighted.__init__`

`RecallWeighted.__init__()`  
Initialize self. See `help(type(self))` for accurate signature.

### `evalml.objectives.RecallWeighted.calculate_percent_difference`

**classmethod** `RecallWeighted.calculate_percent_difference` (*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`

### `evalml.objectives.RecallWeighted.is_defined_for_problem_type`

**classmethod** `RecallWeighted.is_defined_for_problem_type` (*problem\_type*)

### `evalml.objectives.RecallWeighted.objective_function`

`RecallWeighted.objective_function` (*y\_true*, *y\_predicted*, *X=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

**Returns** Numerical value used to calculate score

### `evalml.objectives.RecallWeighted.score`

`RecallWeighted.score` (*y\_true*, *y\_predicted*, *X=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

**Returns** score

### `evalml.objectives.RecallWeighted.validate_inputs`

`RecallWeighted.validate_inputs` (*y\_true*, *y\_predicted*)

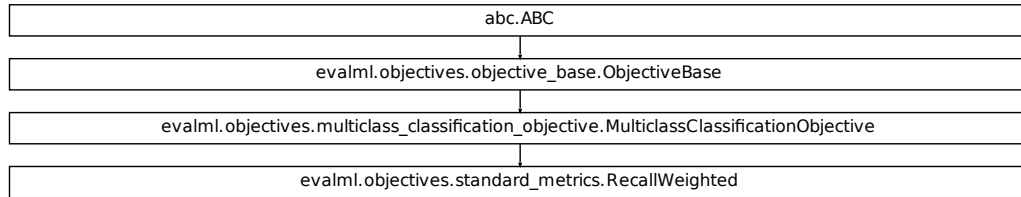
Validates the input based on a few simple checks.

#### Parameters

- ***y\_predicted*** (`ww.DataColumn`, `ww.DataTable`, `pd.Series`, or `pd.DataFrame`) – Predicted values of length [*n\_samples*]
- ***y\_true*** (`ww.DataColumn`, `pd.Series`) – Actual class labels of length [*n\_samples*]

**Returns** None

## Class Inheritance



### 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.
<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.

#### evalml.objectives.R2

**class** evalml.objectives.R2

Coefficient of determination for regression.

**name** = 'R2'

**greater\_is\_better** = True

**perfect\_score** = 1

**positive\_only** = False

**problem\_types** = [<ProblemTypes.REGRESSION: 'regression'>, <ProblemTypes.TIME\_SERIES\_RE

**score\_needs\_proba** = False

## Methods

<code>__init__</code>	Initialize self.
<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>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.

### `evalml.objectives.R2.__init__`

`R2.__init__()`

Initialize self. See `help(type(self))` for accurate signature.

### `evalml.objectives.R2.calculate_percent_difference`

**classmethod** `R2.calculate_percent_difference(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

### `evalml.objectives.R2.is_defined_for_problem_type`

**classmethod** `R2.is_defined_for_problem_type(problem_type)`

**evalml.objectives.R2.objective\_function****R2.objective\_function** (*y\_true*, *y\_predicted*, *X=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

**Returns** Numerical value used to calculate score

**evalml.objectives.R2.score****R2.score** (*y\_true*, *y\_predicted*, *X=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

**Returns** score

**evalml.objectives.R2.validate\_inputs****R2.validate\_inputs** (*y\_true*, *y\_predicted*)

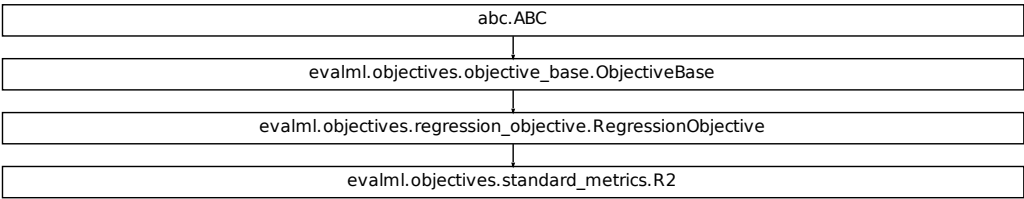
Validates the input based on a few simple checks.

**Parameters**

- **y\_predicted** (*ww.DataColumn*, *ww.DataTable*, *pd.Series*, or *pd.DataFrame*) – Predicted values of length [*n\_samples*]
- **y\_true** (*ww.DataColumn*, *pd.Series*) – Actual class labels of length [*n\_samples*]

**Returns** None

Class Inheritance



evalml.objectives.MAE

```
class evalml.objectives.MAE
    Mean absolute error for regression.

    name = 'MAE'

    greater_is_better = False

    perfect_score = 0.0

    positive_only = False

    problem_types = [<ProblemTypes.REGRESSION: 'regression'>, <ProblemTypes.TIME_SERIES_RE

    score_needs_proba = False
```

Methods

<code>__init__</code>	Initialize self.
<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 predic- tions compared to the actual labels, according a spec- ified metric
<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.



**evalml.objectives.MAE.\_\_init\_\_****MAE.\_\_init\_\_()**

Initialize self. See help(type(self)) for accurate signature.

**evalml.objectives.MAE.calculate\_percent\_difference****classmethod MAE.calculate\_percent\_difference** (*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**evalml.objectives.MAE.is\_defined\_for\_problem\_type****classmethod MAE.is\_defined\_for\_problem\_type** (*problem\_type*)**evalml.objectives.MAE.objective\_function****MAE.objective\_function** (*y\_true, y\_predicted, X=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

**Returns** Numerical value used to calculate score**evalml.objectives.MAE.score****MAE.score** (*y\_true, y\_predicted, X=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

**Returns** score

### `evalml.objectives.MAE.validate_inputs`

`MAE.validate_inputs(y_true, y_predicted)`

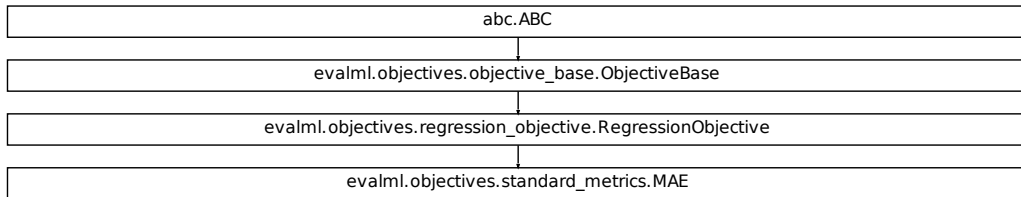
Validates the input based on a few simple checks.

#### Parameters

- **y\_predicted** (*ww.DataColumn, ww.DataTable, pd.Series, or pd.DataFrame*) – Predicted values of length [n\_samples]
- **y\_true** (*ww.DataColumn, pd.Series*) – Actual class labels of length [n\_samples]

**Returns** None

### Class Inheritance



### `evalml.objectives.MAPE`

**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`

**name** = 'Mean Absolute Percentage Error'

**greater\_is\_better** = False

**perfect\_score** = 0.0

**positive\_only** = True

**problem\_types** = [`ProblemTypes.TIME_SERIES_REGRESSION`: 'time series regression']

**score\_needs\_proba** = False

## Methods

<code>__init__</code>	Initialize self.
<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>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.

### `evalml.objectives.MAPE.__init__`

`MAPE.__init__()`

Initialize self. See `help(type(self))` for accurate signature.

### `evalml.objectives.MAPE.calculate_percent_difference`

**classmethod** `MAPE.calculate_percent_difference` (*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

### `evalml.objectives.MAPE.is_defined_for_problem_type`

**classmethod** `MAPE.is_defined_for_problem_type` (*problem\_type*)

### `evalml.objectives.MAPE.objective_function`

`MAPE.objective_function` (*y\_true*, *y\_predicted*, *X=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

**Returns** Numerical value used to calculate score

### `evalml.objectives.MAPE.score`

`MAPE.score` (*y\_true*, *y\_predicted*, *X=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

**Returns** score

### `evalml.objectives.MAPE.validate_inputs`

`MAPE.validate_inputs` (*y\_true*, *y\_predicted*)

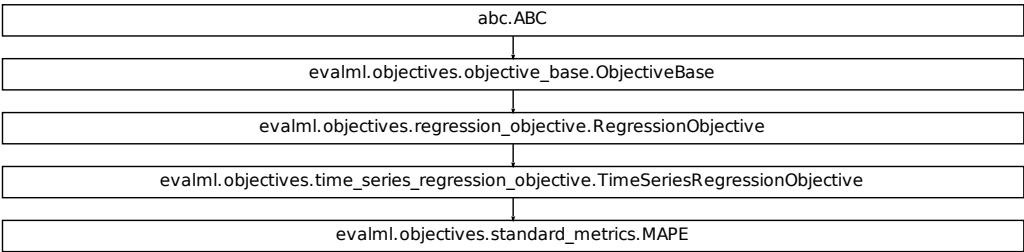
Validates the input based on a few simple checks.

#### Parameters

- ***y\_predicted*** (`ww.DataColumn`, `ww.DataTable`, `pd.Series`, or `pd.DataFrame`) – Predicted values of length [*n\_samples*]
- ***y\_true*** (`ww.DataColumn`, `pd.Series`) – Actual class labels of length [*n\_samples*]

**Returns** None

Class Inheritance



evalml.objectives.MSE

```
class evalml.objectives.MSE
    Mean squared error for regression.
    name = 'MSE'
    greater_is_better = False
    perfect_score = 0.0
    positive_only = False
    problem_types = [<ProblemTypes.REGRESSION: 'regression'>, <ProblemTypes.TIME_SERIES_RE
    score_needs_proba = False
```

Methods

<code>__init__</code>	Initialize self.
<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 predic- tions compared to the actual labels, according a spec- ified metric
<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.

**evalml.objectives.MSE.\_\_init\_\_****MSE.\_\_init\_\_()**

Initialize self. See help(type(self)) for accurate signature.

**evalml.objectives.MSE.calculate\_percent\_difference****classmethod MSE.calculate\_percent\_difference** (*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**evalml.objectives.MSE.is\_defined\_for\_problem\_type****classmethod MSE.is\_defined\_for\_problem\_type** (*problem\_type*)**evalml.objectives.MSE.objective\_function****MSE.objective\_function** (*y\_true, y\_predicted, X=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

**Returns** Numerical value used to calculate score**evalml.objectives.MSE.score****MSE.score** (*y\_true, y\_predicted, X=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

**Returns** score

### evalml.objectives.MSE.validate\_inputs

**MSE.validate\_inputs** (*y\_true*, *y\_predicted*)

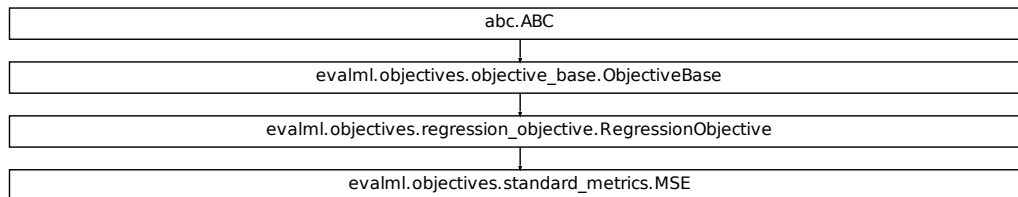
Validates the input based on a few simple checks.

#### Parameters

- **y\_predicted** (*ww.DataColumn*, *ww.DataTable*, *pd.Series*, or *pd.DataFrame*) – Predicted values of length [n\_samples]
- **y\_true** (*ww.DataColumn*, *pd.Series*) – Actual class labels of length [n\_samples]

**Returns** None

### Class Inheritance



### evalml.objectives.MeanSquaredLogError

**class** evalml.objectives.MeanSquaredLogError

Mean squared log error for regression.

Only valid for nonnegative inputs. Otherwise, will throw a ValueError

**name** = 'Mean Squared Log Error'

**greater\_is\_better** = False

**perfect\_score** = 0.0

**positive\_only** = True

**problem\_types** = [<ProblemTypes.REGRESSION: 'regression'>, <ProblemTypes.TIME\_SERIES\_RE

**score\_needs\_proba** = False

## Methods

<code>__init__</code>	Initialize self.
<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>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.

### `evalml.objectives.MeanSquaredLogError.__init__`

`MeanSquaredLogError.__init__()`

Initialize self. See `help(type(self))` for accurate signature.

### `evalml.objectives.MeanSquaredLogError.calculate_percent_difference`

**classmethod** `MeanSquaredLogError.calculate_percent_difference` (*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`

### `evalml.objectives.MeanSquaredLogError.is_defined_for_problem_type`

**classmethod** `MeanSquaredLogError.is_defined_for_problem_type` (*problem\_type*)



**evalml.objectives.MeanSquaredLogError.objective\_function**

`MeanSquaredLogError.objective_function(y_true, y_predicted, X=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

**Returns** Numerical value used to calculate score

**evalml.objectives.MeanSquaredLogError.score**

`MeanSquaredLogError.score(y_true, y_predicted, X=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

**Returns** score

**evalml.objectives.MeanSquaredLogError.validate\_inputs**

`MeanSquaredLogError.validate_inputs(y_true, y_predicted)`

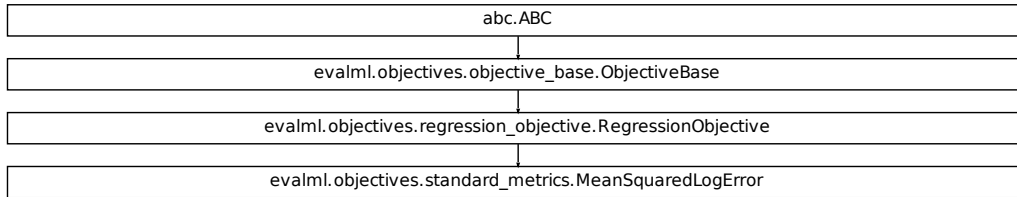
Validates the input based on a few simple checks.

**Parameters**

- **y\_predicted** (`ww.DataColumn`, `ww.DataTable`, `pd.Series`, or `pd.DataFrame`) – Predicted values of length `[n_samples]`
- **y\_true** (`ww.DataColumn`, `pd.Series`) – Actual class labels of length `[n_samples]`

**Returns** None

## Class Inheritance



### evalml.objectives.MedianAE

**class** evalml.objectives.MedianAE

Median absolute error for regression.

**name** = 'MedianAE'

**greater\_is\_better** = False

**perfect\_score** = 0.0

**positive\_only** = False

**problem\_types** = [<ProblemTypes.REGRESSION: 'regression'>, <ProblemTypes.TIME\_SERIES\_RE

**score\_needs\_proba** = False

#### Methods

<code>__init__</code>	Initialize self.
<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>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.

**evalml.objectives.MedianAE.\_\_init\_\_****MedianAE.\_\_init\_\_()**

Initialize self. See help(type(self)) for accurate signature.

**evalml.objectives.MedianAE.calculate\_percent\_difference****classmethod MedianAE.calculate\_percent\_difference** (*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**evalml.objectives.MedianAE.is\_defined\_for\_problem\_type****classmethod MedianAE.is\_defined\_for\_problem\_type** (*problem\_type*)**evalml.objectives.MedianAE.objective\_function****MedianAE.objective\_function** (*y\_true, y\_predicted, X=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

**Returns** Numerical value used to calculate score**evalml.objectives.MedianAE.score****MedianAE.score** (*y\_true, y\_predicted, X=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

**Returns** score

### `evalml.objectives.MedianAE.validate_inputs`

`MedianAE.validate_inputs(y_true, y_predicted)`

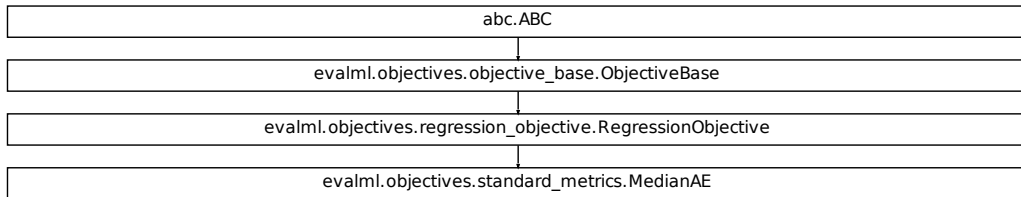
Validates the input based on a few simple checks.

#### Parameters

- **y\_predicted** (*ww.DataColumn*, *ww.DataTable*, *pd.Series*, or *pd.DataFrame*) – Predicted values of length [n\_samples]
- **y\_true** (*ww.DataColumn*, *pd.Series*) – Actual class labels of length [n\_samples]

**Returns** None

### Class Inheritance



### `evalml.objectives.MaxError`

**class** `evalml.objectives.MaxError`

Maximum residual error for regression.

**name** = 'MaxError'

**greater\_is\_better** = False

**perfect\_score** = 0.0

**positive\_only** = False

**problem\_types** = [`<ProblemTypes.REGRESSION: 'regression'>`, `<ProblemTypes.TIME_SERIES_RE`]

**score\_needs\_proba** = False

## Methods

<code>__init__</code>	Initialize self.
<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>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.

### `evalml.objectives.MaxError.__init__`

`MaxError.__init__()`

Initialize self. See help(type(self)) for accurate signature.

### `evalml.objectives.MaxError.calculate_percent_difference`

**classmethod** `MaxError.calculate_percent_difference` (*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

### `evalml.objectives.MaxError.is_defined_for_problem_type`

**classmethod** `MaxError.is_defined_for_problem_type` (*problem\_type*)

### `evalml.objectives.MaxError.objective_function`

`MaxError.objective_function` (*y\_true*, *y\_predicted*, *X=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

**Returns** Numerical value used to calculate score

### `evalml.objectives.MaxError.score`

`MaxError.score` (*y\_true*, *y\_predicted*, *X=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

**Returns** score

### `evalml.objectives.MaxError.validate_inputs`

`MaxError.validate_inputs` (*y\_true*, *y\_predicted*)

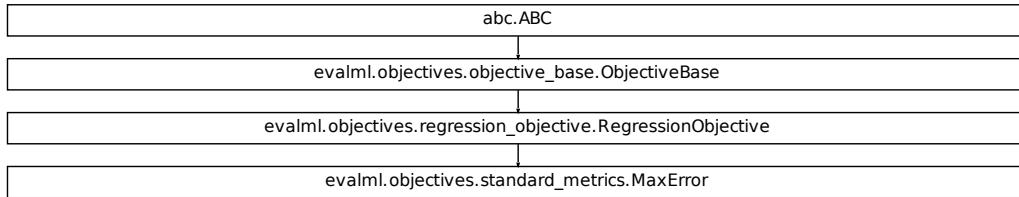
Validates the input based on a few simple checks.

#### Parameters

- ***y\_predicted*** (`ww.DataColumn`, `ww.DataTable`, `pd.Series`, or `pd.DataFrame`) – Predicted values of length [*n\_samples*]
- ***y\_true*** (`ww.DataColumn`, `pd.Series`) – Actual class labels of length [*n\_samples*]

**Returns** None

## Class Inheritance



### evalml.objectives.ExpVariance

**class** evalml.objectives.**ExpVariance**

Explained variance score for regression.

**name** = 'ExpVariance'

**greater\_is\_better** = True

**perfect\_score** = 1.0

**positive\_only** = False

**problem\_types** = [<ProblemTypes.REGRESSION: 'regression'>, <ProblemTypes.TIME\_SERIES\_RE

**score\_needs\_proba** = False

#### Methods

<code>__init__</code>	Initialize self.
<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>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.

### `evalml.objectives.ExpVariance.__init__`

`ExpVariance.__init__()`

Initialize self. See `help(type(self))` for accurate signature.

### `evalml.objectives.ExpVariance.calculate_percent_difference`

**classmethod** `ExpVariance.calculate_percent_difference(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`

### `evalml.objectives.ExpVariance.is_defined_for_problem_type`

**classmethod** `ExpVariance.is_defined_for_problem_type(problem_type)`

### `evalml.objectives.ExpVariance.objective_function`

`ExpVariance.objective_function(y_true, y_predicted, X=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

**Returns** Numerical value used to calculate score

### `evalml.objectives.ExpVariance.score`

`ExpVariance.score(y_true, y_predicted, X=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



**Returns** score

### `evalml.objectives.ExpVariance.validate_inputs`

`ExpVariance.validate_inputs(y_true, y_predicted)`

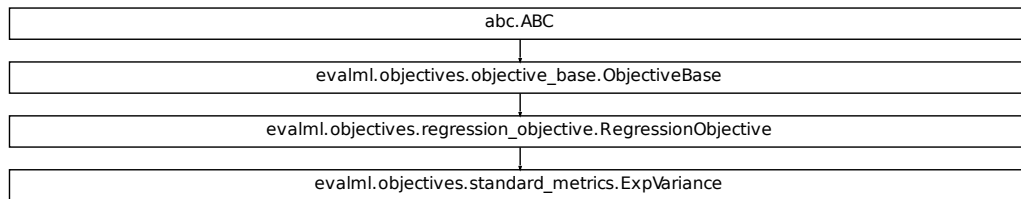
Validates the input based on a few simple checks.

#### **Parameters**

- **y\_predicted** (*ww.DataColumn*, *ww.DataTable*, *pd.Series*, or *pd.DataFrame*) – Predicted values of length [n\_samples]
- **y\_true** (*ww.DataColumn*, *pd.Series*) – Actual class labels of length [n\_samples]

**Returns** None

### Class Inheritance



### `evalml.objectives.RootMeanSquaredError`

**class** `evalml.objectives.RootMeanSquaredError`

Root mean squared error for regression.

**name** = 'Root Mean Squared Error'

**greater\_is\_better** = False

**perfect\_score** = 0.0

**positive\_only** = False

**problem\_types** = [`<ProblemTypes.REGRESSION: 'regression'>`, `<ProblemTypes.TIME_SERIES_RE`]

**score\_needs\_proba** = False

## Methods

<code>__init__</code>	Initialize self.
<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>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.

### `evalml.objectives.RootMeanSquaredError.__init__`

`RootMeanSquaredError.__init__()`  
Initialize self. See `help(type(self))` for accurate signature.

### `evalml.objectives.RootMeanSquaredError.calculate_percent_difference`

**classmethod** `RootMeanSquaredError.calculate_percent_difference` (*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`

### `evalml.objectives.RootMeanSquaredError.is_defined_for_problem_type`

**classmethod** `RootMeanSquaredError.is_defined_for_problem_type` (*problem\_type*)

**evalml.objectives.RootMeanSquaredError.objective\_function**

`RootMeanSquaredError.objective_function(y_true, y_predicted, X=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

**Returns** Numerical value used to calculate score

**evalml.objectives.RootMeanSquaredError.score**

`RootMeanSquaredError.score(y_true, y_predicted, X=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

**Returns** score

**evalml.objectives.RootMeanSquaredError.validate\_inputs**

`RootMeanSquaredError.validate_inputs(y_true, y_predicted)`

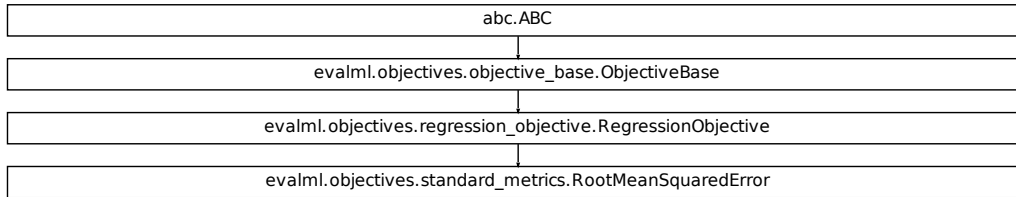
Validates the input based on a few simple checks.

**Parameters**

- **y\_predicted** (`ww.DataColumn`, `ww.DataTable`, `pd.Series`, or `pd.DataFrame`) – Predicted values of length `[n_samples]`
- **y\_true** (`ww.DataColumn`, `pd.Series`) – Actual class labels of length `[n_samples]`

**Returns** None

## Class Inheritance



### evalml.objectives.RootMeanSquaredLogError

**class** evalml.objectives.RootMeanSquaredLogError

Root mean squared log error for regression.

Only valid for nonnegative inputs. Otherwise, will throw a ValueError.

**name** = 'Root Mean Squared Log Error'

**greater\_is\_better** = False

**perfect\_score** = 0.0

**positive\_only** = True

**problem\_types** = [<ProblemTypes.REGRESSION: 'regression'>, <ProblemTypes.TIME\_SERIES\_RE

**score\_needs\_proba** = False

#### Methods

<code>__init__</code>	Initialize self.
<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>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.

**evalml.objectives.RootMeanSquaredLogError.\_\_init\_\_**

`RootMeanSquaredLogError.__init__()`

Initialize self. See help(type(self)) for accurate signature.

**evalml.objectives.RootMeanSquaredLogError.calculate\_percent\_difference**

**classmethod** `RootMeanSquaredLogError.calculate_percent_difference` (*score*,  
*base-*  
*line\_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

**evalml.objectives.RootMeanSquaredLogError.is\_defined\_for\_problem\_type**

**classmethod** `RootMeanSquaredLogError.is_defined_for_problem_type` (*problem\_type*)

**evalml.objectives.RootMeanSquaredLogError.objective\_function**

`RootMeanSquaredLogError.objective_function` (*y\_true*, *y\_predicted*, *X=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

**Returns** Numerical value used to calculate score

**evalml.objectives.RootMeanSquaredLogError.score**

`RootMeanSquaredLogError.score` (*y\_true*, *y\_predicted*, *X=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

**Returns** score

### `evalml.objectives.RootMeanSquaredLogError.validate_inputs`

`RootMeanSquaredLogError.validate_inputs` (*y\_true*, *y\_predicted*)

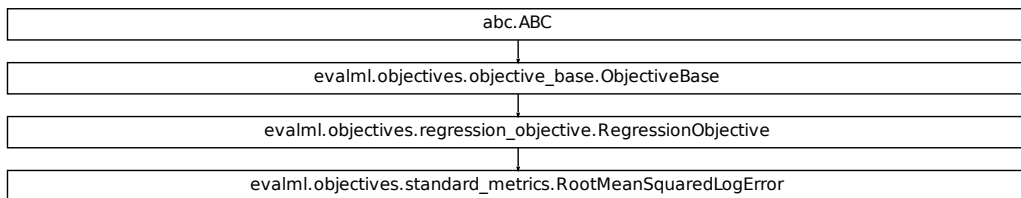
Validates the input based on a few simple checks.

#### Parameters

- **y\_predicted** (*ww.DataColumn*, *ww.DataTable*, *pd.Series*, or *pd.DataFrame*) – Predicted values of length [n\_samples]
- **y\_true** (*ww.DataColumn*, *pd.Series*) – Actual class labels of length [n\_samples]

**Returns** None

## Class Inheritance



## 5.8.5 Objective Utils

<code>get_all_objective_names</code>	Get a list of the names of all objectives.
<code>get_core_objectives</code>	Returns all core objective instances associated with the given problem type.
<code>get_core_objective_names</code>	Get a list of all valid core objectives.
<code>get_non_core_objectives</code>	Get non-core objective classes.
<code>get_objective</code>	Returns the Objective class corresponding to a given objective name.

### evalml.objectives.get\_all\_objective\_names

`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\_objectives

`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\_core\_objective\_names

`evalml.objectives.get_core_objective_names()`

Get a list of all valid core objectives.

**Returns** Objective names.

**Return type** list[str]

### evalml.objectives.get\_non\_core\_objectives

`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

`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.

## 5.9 Problem Types

<code>handle_problem_types</code>	Handles <code>problem_type</code> by either returning the <code>ProblemTypes</code> or converting from a str.
<code>detect_problem_type</code>	Determine the type of problem is being solved based on the targets (binary vs multiclass classification, regression)
<code>ProblemTypes</code>	Enum defining the supported types of machine learning problems.

### 5.9.1 `evalml.problem_types.handle_problem_types`

`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`

### 5.9.2 `evalml.problem_types.detect_problem_type`

`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
```

### 5.9.3 `evalml.problem_types.ProblemTypes`

**class** `evalml.problem_types.ProblemTypes(value)`

Enum defining the supported types of machine learning problems.



**Attributes**

BINARY	Binary classification problem.
MULTICLASS	Multiclass classification problem.
REGRESSION	Regression problem.
TIME_SERIES_BINARY	Time series binary classification problem.
TIME_SERIES_MULTICLASS	Time series multiclass classification problem.
TIME_SERIES_REGRESSION	Time series regression problem.

## 5.10 Model Family

<code>handle_model_family</code>	Handles <code>model_family</code> by either returning the <code>ModelFamily</code> or converting from a string
<code>ModelFamily</code>	Enum for family of machine learning models.

### 5.10.1 evalml.model\_family.handle\_model\_family

`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`

### 5.10.2 evalml.model\_family.ModelFamily

**class** `evalml.model_family.ModelFamily` (*value*)

Enum for family of machine learning models.

**Attributes**

ARIMA	ARIMA model family.
BASELINE	Baseline model family.
CATBOOST	CatBoost model family.
DECISION_TREE	Decision Tree model family.
ENSEMBLE	Ensemble model family.
EXTRA_TREES	Extra Trees model family.
K_NEIGHBORS	K Nearest Neighbors model family.
LIGHTGBM	LightGBM model family.
LINEAR_MODEL	Linear model family.
NONE	None
RANDOM_FOREST	Random Forest model family.
SVM	SVM model family.
XGBOOST	XGBoost model family.

## 5.11 Tuners

<i>Tuner</i>	Defines API for Tuners.
<i>SKOptTuner</i>	Bayesian Optimizer.
<i>GridSearchTuner</i>	Grid Search Optimizer.
<i>RandomSearchTuner</i>	Random Search Optimizer.

### 5.11.1 evalml.tuners.Tuner

**class** evalml.tuners.**Tuner** (*pipeline\_hyperparameter\_ranges*, *random\_seed=0*)

Defines API for Tuners.

Tuners implement different strategies for sampling from a search space. They're used in EvalML to search the space of pipeline hyperparameters.

#### Methods

<i>__init__</i>	Base Tuner class
<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.
<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.

#### evalml.tuners.Tuner.\_\_init\_\_

Tuner.**\_\_init\_\_** (*pipeline\_hyperparameter\_ranges*, *random\_seed=0*)

Base Tuner class

##### 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.

#### evalml.tuners.Tuner.add

**abstract** Tuner.**add** (*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

**evalml.tuners.Tuner.is\_search\_space\_exhausted****Tuner.is\_search\_space\_exhausted()**

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.

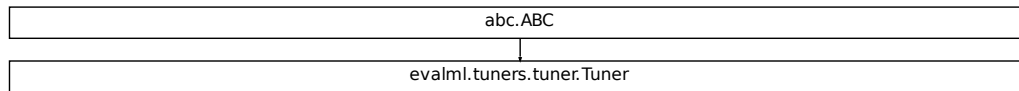
**Return type** bool

**evalml.tuners.Tuner.propose****abstract Tuner.propose()**

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 Inheritance****5.11.2 evalml.tuners.SKOptTuner**

**class** evalml.tuners.**SKOptTuner** (*pipeline\_hyperparameter\_ranges*, *random\_seed=0*)  
Bayesian Optimizer.

**Methods**

<code>__init__</code>	Init SKOptTuner
<code>add</code>	Add score to sample
<code>is_search_space_exhausted</code>	Optional.
<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.

**evalml.tuners.SKOptTuner.\_\_init\_\_**

`SKOptTuner.__init__(pipeline_hyperparameter_ranges, random_seed=0)`  
Init SKOptTuner

**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.

**evalml.tuners.SKOptTuner.add**

`SKOptTuner.add(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

**evalml.tuners.SKOptTuner.is\_search\_space\_exhausted**

`SKOptTuner.is_search_space_exhausted()`  
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

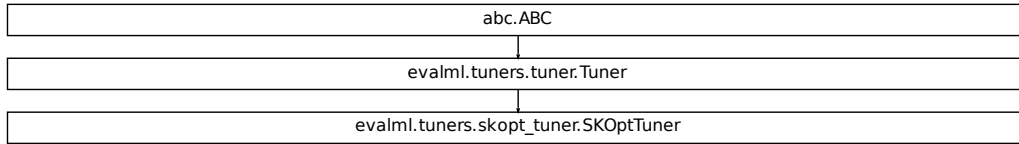
**evalml.tuners.SKOptTuner.propose**

`SKOptTuner.propose()`  
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 Inheritance



### 5.11.3 evalml.tuners.GridSearchTuner

**class** evalml.tuners.**GridSearchTuner** (*pipeline\_hyperparameter\_ranges*, *n\_points=10*, *random\_seed=0*)  
Grid Search Optimizer.

#### 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>__init__</code>	Generate all of the possible points to search for in the grid
<code>add</code>	Not applicable to grid search tuner as generated parameters are not dependent on scores of previous parameters.
<code>is_search_space_exhausted</code>	Checks if it is possible to generate a set of valid parameters.
<code>propose</code>	Returns parameters from <code>_grid_points</code> iterations

### `evalml.tuners.GridSearchTuner.__init__`

`GridSearchTuner.__init__(pipeline_hyperparameter_ranges, n_points=10, random_seed=0)`  
Generate all of the possible points to search for in the 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
- **`random_seed`** (*int*) – Seed for random number generator. Unused in this class, defaults to 0.

### `evalml.tuners.GridSearchTuner.add`

`GridSearchTuner.add(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

### `evalml.tuners.GridSearchTuner.is_search_space_exhausted`

`GridSearchTuner.is_search_space_exhausted()`  
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`

### `evalml.tuners.GridSearchTuner.propose`

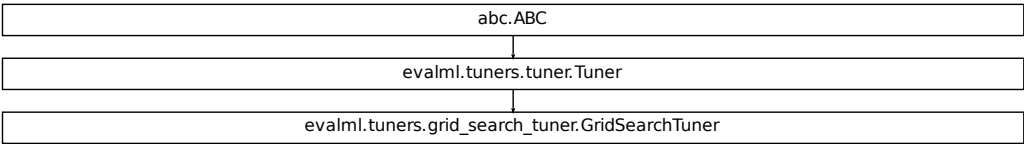
`GridSearchTuner.propose()`  
Returns parameters from `_grid_points` iterations

If all possible combinations of parameters have been scored, then `NoParamsException` is raised.

**Returns** proposed pipeline parameters

**Return type** `dict`

Class Inheritance



5.11.4 evalml.tuners.RandomSearchTuner

**class** evalml.tuners.**RandomSearchTuner** (*pipeline\_hyperparameter\_ranges*, *random\_seed=0*, *with\_replacement=False*, *replacement\_max\_attempts=10*)

Random Search Optimizer.

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>__init__</code>	Sets up check for duplication if needed.
<code>add</code>	Not applicable to random search tuner as generated parameters are not dependent on scores of previous parameters.
<code>is_search_space_exhausted</code>	Checks if it is possible to generate a set of valid parameters.
<code>propose</code>	Generate a unique set of parameters.

### `evalml.tuners.RandomSearchTuner.__init__`

`RandomSearchTuner.__init__(pipeline_hyperparameter_ranges, random_seed=0, with_replacement=False, replacement_max_attempts=10)`

Sets up check for duplication if needed.

#### Parameters

- **pipeline\_hyperparameter\_ranges** (*dict*) – a set of hyperparameter ranges corresponding to a pipeline’s parameters
- **random\_state** (*int*) – Unused in this class. Defaults to 0.
- **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.

### `evalml.tuners.RandomSearchTuner.add`

`RandomSearchTuner.add(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

### `evalml.tuners.RandomSearchTuner.is_search_space_exhausted`

`RandomSearchTuner.is_search_space_exhausted()`

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`

### `evalml.tuners.RandomSearchTuner.propose`

`RandomSearchTuner.propose()`

Generate a unique set of parameters.

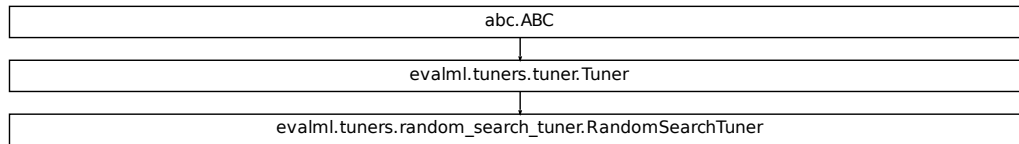
If tuner was initialized with `with_replacement=True` and the tuner is unable to generate a unique set of parameters after `replacement_max_attempts` tries, then `NoParamsException` is raised.

**Returns** Proposed pipeline parameters

**Return type** `dict`



## Class Inheritance



## 5.12 Data Checks

### 5.12.1 Data Check Classes

<i>DataCheck</i>	Base class for all data checks.
<i>InvalidTargetDataCheck</i>	Checks if the target data contains missing or invalid values.
<i>HighlyNullDataCheck</i>	Checks if there are any highly-null columns 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.
<i>NoVarianceDataCheck</i>	Check if the target or any of the features have no variance.
<i>ClassImbalanceDataCheck</i>	Checks if any target labels are imbalanced beyond a threshold.
<i>MulticollinearityDataCheck</i>	Check if any set features are likely to be multicollinear.
<i>DateTimeNaNDataCheck</i>	Checks if datetime columns contain NaN values.
<i>NaturalLanguageNaNDataCheck</i>	Checks if natural language columns contain NaN values.

#### evalml.data\_checks.DataCheck

**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.

**name** = 'DataCheck'

**Methods:**

<code>__init__</code>	Initialize self.
<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.

**evalml.data\_checks.DataCheck.\_\_init\_\_**

`DataCheck.__init__()`

Initialize self. See `help(type(self))` for accurate signature.

**evalml.data\_checks.DataCheck.validate**

**abstract** `DataCheck.validate(X, y=None)`

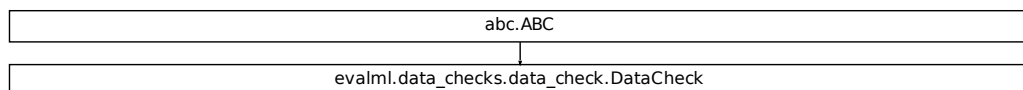
Inspects and validates the input data, runs any necessary calculations or algorithms, and returns a list of warnings and errors if applicable.

**Parameters**

- **X** (*ww.DataTable*, *pd.DataFrame*) – The input data of shape `[n_samples, n_features]`
- **y** (*ww.DataColumn*, *pd.Series*, *optional*) – The target data of length `[n_samples]`

**Returns** Dictionary of `DataCheckError` and `DataCheckWarning` messages

**Return type** dict (*DataCheckMessage*)

**Class Inheritance**

**evalml.data\_checks.InvalidTargetDataCheck**

**class** evalml.data\_checks.InvalidTargetDataCheck (*problem\_type*, *objective*,  
*n\_unique=100*)

Checks if the target data contains missing or invalid values.

**name** = 'InvalidTargetDataCheck'

**Methods:**

<code>__init__</code>	Check if the target is invalid for the specified problem type.
<code>validate</code>	Checks if the target data contains missing or invalid values.

**evalml.data\_checks.InvalidTargetDataCheck.\_\_init\_\_**

InvalidTargetDataCheck.**\_\_init\_\_** (*problem\_type*, *objective*, *n\_unique=100*)

Check if the target is invalid for the specified problem type.

**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. Defaults to 100. If None, stores all unique values.

**evalml.data\_checks.InvalidTargetDataCheck.validate**

InvalidTargetDataCheck.**validate** (*X*, *y*)

Checks if the target data contains missing or invalid values.

**Parameters**

- **X** (*ww.DataTable*, *pd.DataFrame*, *np.ndarray*) – Features. Ignored.
- **y** (*ww.DataColumn*, *pd.Series*, *np.ndarray*) – Target data to check for invalid values.

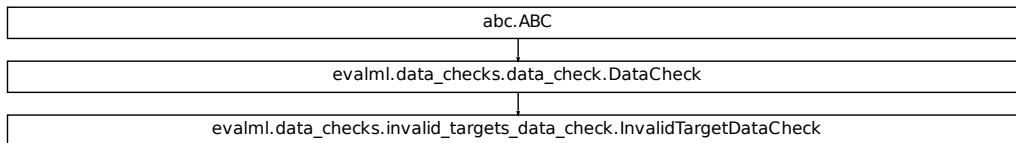
**Returns** List with DataCheckErrors if any invalid values are found in the target data.

**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 Inheritance



## evalml.data\_checks.HighlyNullDataCheck

**class** evalml.data\_checks.**HighlyNullDataCheck** (*pct\_null\_threshold=0.95*)

Checks if there are any highly-null columns in the input.

**name** = 'HighlyNullDataCheck'

### Methods:

<code>__init__</code>	Checks if there are any highly-null columns in the input.
<code>validate</code>	Checks if there are any highly-null columns in the input.

**evalml.data\_checks.HighlyNullDataCheck.\_\_init\_\_**

`HighlyNullDataCheck.__init__(pct_null_threshold=0.95)`

Checks if there are any highly-null columns in the input.

**Parameters** `pct_null_threshold` (*float*) – If the percentage of NaN values in an input feature exceeds this amount, that feature will be considered highly-null. Defaults to 0.95.

**evalml.data\_checks.HighlyNullDataCheck.validate**

`HighlyNullDataCheck.validate(X, y=None)`

Checks if there are any highly-null columns in the input.

**Parameters**

- **X** (*ww.DataTable*, *pd.DataFrame*, *np.ndarray*) – Features
- **y** (*ww.DataColumn*, *pd.Series*, *np.ndarray*) – Ignored.

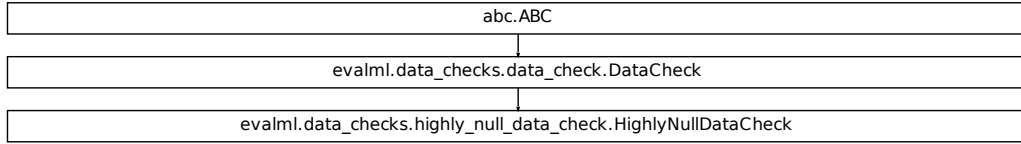
**Returns** dict with a `DataCheckWarning` if there are any highly-null columns.

**Return type** dict

**Example**

```
>>> import pandas as pd
>>> df = pd.DataFrame({
...     'lots_of_null': [None, None, None, None, 5],
...     'no_null': [1, 2, 3, 4, 5]
... })
>>> null_check = HighlyNullDataCheck(pct_null_threshold=0.8)
>>> assert null_check.validate(df) == {"errors": [],
    "warnings": [{"message": "Column 'lots_of_null' is
    ↳80.0% or more null",
    ↳     "data_check_name": "HighlyNullDataCheck",
    ↳     "level": "warning",
    ↳     "code": "HIGHLY_NULL",
    ↳     "details": {
    ↳ "column": "lots_of_null"}},
    ↳     "actions": [{"code": "DROP_COL",
    ↳     "metadata": {"column": "lots_of_null"}}]}
```

## Class Inheritance



### evalml.data\_checks.IDColumnsDataCheck

**class** evalml.data\_checks.IDColumnsDataCheck (*id\_threshold=1.0*)

Check if any of the features are likely to be ID columns.

**name** = 'IDColumnsDataCheck'

#### Methods:

<code>__init__</code>	Check if any of the features are likely to be ID columns.
<code>validate</code>	Check if any of the features are likely to be ID columns.

### evalml.data\_checks.IDColumnsDataCheck.\_\_init\_\_

IDColumnsDataCheck.**\_\_init\_\_** (*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.

### evalml.data\_checks.IDColumnsDataCheck.validate

IDColumnsDataCheck.**validate** (*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** (*ww.DataTable, pd.DataFrame, np.ndarray*) – The input features to check

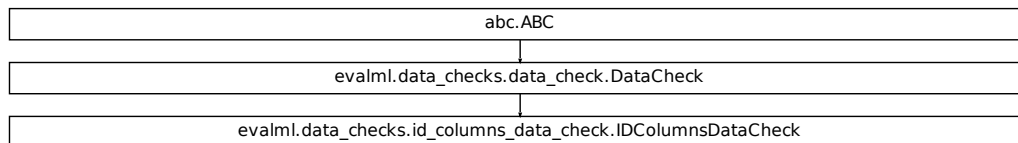
**Returns** A dictionary of features with column name or index and their probability of being ID columns

**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 Inheritance



### evalml.data\_checks.TargetLeakageDataCheck

**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.

**name** = 'TargetLeakageDataCheck'

**Methods:**

<code>__init__</code>	Check if any of the features are highly correlated with the target by using mutual information or Pearson correlation.
<code>validate</code>	Check if any of the features are highly correlated with the target by using mutual information or Pearson correlation.

**evalml.data\_checks.TargetLeakageDataCheck.\_\_init\_\_**

`TargetLeakageDataCheck.__init__(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’.

**evalml.data\_checks.TargetLeakageDataCheck.validate**

`TargetLeakageDataCheck.validate(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** (*ww.DataTable, pd.DataFrame, np.ndarray*) – The input features to check
- **y** (*ww.DataColumn, pd.Series, np.ndarray*) – The target data

**Returns** dict with a `DataCheckWarning` if target leakage is detected.

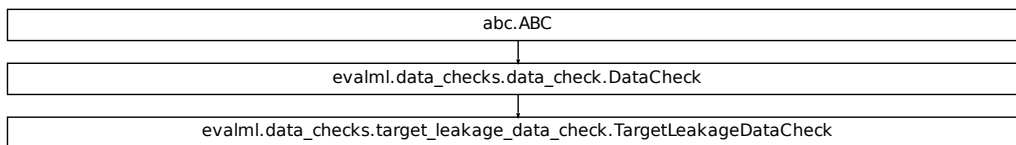
**Return type** dict (*DataCheckWarning*)



## 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 Inheritance



## evalml.data\_checks.OutliersDataCheck

**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.

**name** = 'OutliersDataCheck'

**Methods:**

<code>__init__</code>	Checks if there are any outliers in the input data.
<code>validate</code>	Checks if there are any outliers in a dataframe by using IQR to determine column anomalies.

**evalml.data\_checks.OutliersDataCheck.\_\_init\_\_**

`OutliersDataCheck.__init__()`

Checks if there are any outliers in the input data.

**evalml.data\_checks.OutliersDataCheck.validate**

`OutliersDataCheck.validate(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** (`ww.DataTable`, `pd.DataFrame`, `np.ndarray`) – Features
- **y** (`ww.DataColumn`, `pd.Series`, `np.ndarray`) – Ignored.

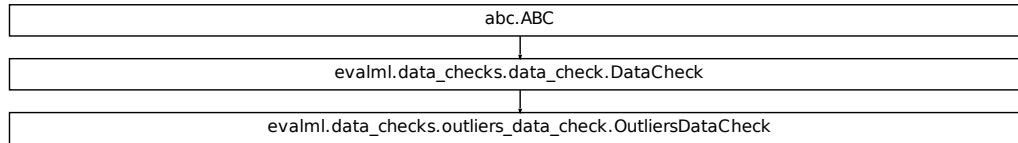
**Returns** A dictionary with warnings if any columns have outliers.

**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 Inheritance



### evalml.data\_checks.NoVarianceDataCheck

**class** evalml.data\_checks.NoVarianceDataCheck(*count\_nan\_as\_value=False*)

Check if the target or any of the features have no variance.

**name** = 'NoVarianceDataCheck'

#### Methods:

<code>__init__</code>	Check if the target or any of the features have no variance.
<code>validate</code>	Check if the target or any of the features have no variance (1 unique value).

### evalml.data\_checks.NoVarianceDataCheck.\_\_init\_\_

NoVarianceDataCheck.**\_\_init\_\_**(*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. If set to True, a feature that has one unique value and all other data is missing, a DataCheckWarning will be returned instead of an error. Defaults to False.

### evalml.data\_checks.NoVarianceDataCheck.validate

NoVarianceDataCheck.**validate**(*X, y*)

Check if the target or any of the features have no variance (1 unique value).

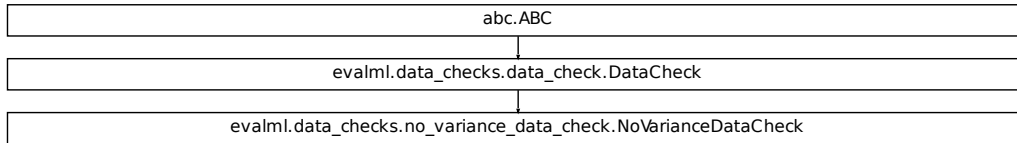
#### Parameters

- **X** (*ww.DataTable, pd.DataFrame, np.ndarray*) – The input features.
- **y** (*ww.DataColumn, pd.Series, np.ndarray*) – The target data.

**Returns** dict of warnings/errors corresponding to features or target with no variance.

**Return type** dict

## Class Inheritance



### evalml.data\_checks.ClassImbalanceDataCheck

```
class evalml.data_checks.ClassImbalanceDataCheck (threshold=0.1, min_samples=100,  
                                                  num_cv_folds=3)  
    Checks if any target labels are imbalanced beyond a threshold. Use for classification problems  
    name = 'ClassImbalanceDataCheck'
```

#### Methods:

<code>__init__</code>	Check if any of the target labels are imbalanced, or if the number of values for each target
<code>validate</code>	Checks if any target labels are imbalanced beyond a threshold for binary and multiclass problems

### evalml.data\_checks.ClassImbalanceDataCheck.\_\_init\_\_

```
ClassImbalanceDataCheck.__init__ (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

#### 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 multiclass 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.

**evalml.data\_checks.ClassImbalanceDataCheck.validate**

`ClassImbalanceDataCheck.validate(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** (`ww.DataTable`, `pd.DataFrame`, `np.ndarray`) – Features. Ignored.
- **y** (`ww.DataColumn`, `pd.Series`, `np.ndarray`) – Target labels to check for imbalanced data.

**Returns**

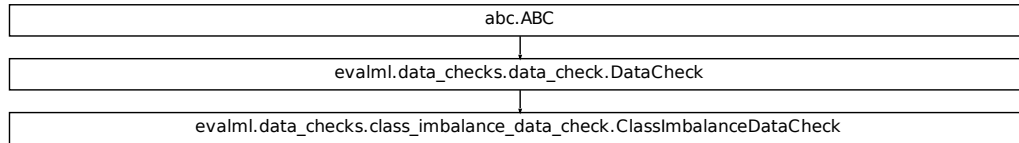
**Dictionary with `DataCheckWarnings` if imbalance in classes is less than the threshold, and `DataCheckErrors` if the number of values for each target is below  $2 * \text{num\_cv\_folds}$ .**

**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 Inheritance



### evalml.data\_checks.MulticollinearityDataCheck

**class** evalml.data\_checks.**MulticollinearityDataCheck** (*threshold=0.9*)

Check if any set features are likely to be multicollinear.

**name** = 'MulticollinearityDataCheck'

#### Methods:

<code>__init__</code>	Check if any set of features are likely to be multicollinear.
<code>validate</code>	Check if any set of features are likely to be multicollinear.

### evalml.data\_checks.MulticollinearityDataCheck.\_\_init\_\_

MulticollinearityDataCheck.**\_\_init\_\_** (*threshold=0.9*)

Check if any set of features are likely to be multicollinear.

**Parameters** **threshold** (*float*) – The threshold to be considered. Defaults to 0.9.

### evalml.data\_checks.MulticollinearityDataCheck.validate

MulticollinearityDataCheck.**validate** (*X, y=None*)

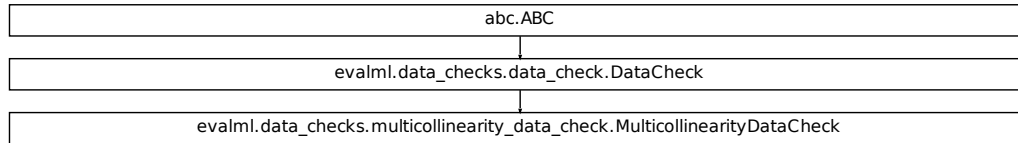
Check if any set of features are likely to be multicollinear.

**Parameters** **X** (*ww.DataTable, 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 Inheritance



### evalml.data\_checks.DateTimeNaNDataCheck

**class** evalml.data\_checks.DateTimeNaNDataCheck

Checks if datetime columns contain NaN values.

**name** = 'DateTimeNaNDataCheck'

#### Methods:

<code>__init__</code>	Checks each column in the input for datetime features and will issue an error if NaN values are present.
<code>validate</code>	Checks if any datetime columns contain NaN values.

### evalml.data\_checks.DateTimeNaNDataCheck.\_\_init\_\_

DateTimeNaNDataCheck.**\_\_init\_\_**()

Checks each column in the input for datetime features and will issue an error if NaN values are present.

### evalml.data\_checks.DateTimeNaNDataCheck.validate

DateTimeNaNDataCheck.**validate**(X, y=None)

Checks if any datetime columns contain NaN values.

#### Parameters

- **X** (*ww.DataTable*, *pd.DataFrame*, *np.ndarray*) – Features.
- **y** (*ww.DataColumn*, *pd.Series*, *np.ndarray*) – Ignored. Defaults to None.

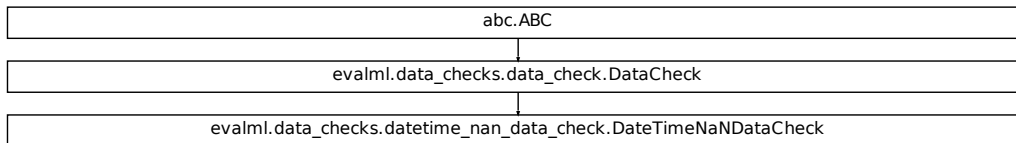
**Returns** dict with a DataCheckError if NaN values are present in datetime columns.

**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')
>>> ww_input = ww.DataTable(pd.DataFrame(dates, columns=['index']))
>>> dt_nan_check = DateTimeNaNDataCheck()
>>> assert dt_nan_check.validate(ww_input) == {"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 Inheritance



### evalml.data\_checks.NaturalLanguageNaNDataCheck

```
class evalml.data_checks.NaturalLanguageNaNDataCheck
    Checks if natural language columns contain NaN values.

    name = 'NaturalLanguageNaNDataCheck'
```



**Methods:**

<code>__init__</code>	Checks each column in the input for natural language features and will issue an error if NaN values are present.
<code>validate</code>	Checks if any natural language columns contain NaN values.

**evalml.data\_checks.NaturalLanguageNaNDataCheck.\_\_init\_\_**

`NaturalLanguageNaNDataCheck.__init__()`

Checks each column in the input for natural language features and will issue an error if NaN values are present.

**evalml.data\_checks.NaturalLanguageNaNDataCheck.validate**

`NaturalLanguageNaNDataCheck.validate(X, y=None)`

Checks if any natural language columns contain NaN values.

**Parameters**

- **X** (`ww.DataTable`, `pd.DataFrame`, `np.ndarray`) – Features.
- **y** (`ww.DataColumn`, `pd.Series`, `np.ndarray`) – Ignored. Defaults to None.

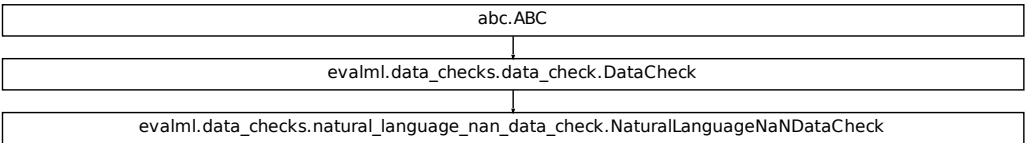
**Returns** dict with a `DataCheckError` if NaN values are present in natural language columns.

**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.DataTable(data, 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()]
... }
```

Class Inheritance



<i>DataChecks</i>	A collection of data checks.
<i>DefaultDataChecks</i>	A collection of basic data checks that is used by AutoML by default.

evalml.data\_checks.DataChecks

**class** evalml.data\_checks.DataChecks (*data\_checks=None, data\_check\_params=None*)  
A collection of data checks.

Methods

<i>__init__</i>	A collection of data checks.
<i>validate</i>	Inspects and validates the input data against data checks and returns a list of warnings and errors if applicable.

evalml.data\_checks.DataChecks.\_\_init\_\_

DataChecks.**\_\_init\_\_** (*data\_checks=None, data\_check\_params=None*)  
A collection of data checks.  
**Parameters** *data\_checks* (*list (DataCheck)*) – List of DataCheck objects

evalml.data\_checks.DataChecks.validate

DataChecks.**validate** (*X, y=None*)  
Inspects and validates the input data against data checks and returns a list of warnings and errors if applicable.  
**Parameters**

- **X** (*ww.DataTable, pd.DataFrame, np.ndarray*) – The input data of shape [n\_samples, n\_features]
- **y** (*ww.DataColumn, pd.Series, np.ndarray*) – The target data of length [n\_samples]

**Returns** Dictionary containing DataCheckMessage objects

**Return type** dict

## Class Inheritance

```
evalml.data_checks.data_checks.DataChecks
```

## evalml.data\_checks.DefaultDataChecks

**class** evalml.data\_checks.DefaultDataChecks (*problem\_type, objective, n\_splits=3*)

A collection of basic data checks that is used by AutoML by default. Includes:

- *HighlyNullDataCheck*
- *IDColumnsDataCheck*
- *TargetLeakageDataCheck*
- *InvalidTargetDataCheck*
- *NoVarianceDataCheck*
- *ClassImbalanceDataCheck* (for classification problem types)
- *DateTimeNaNDataCheck*
- *NaturalLanguageNaNDataCheck*

## Methods

<code>__init__</code>	A collection of basic data checks.
<code>validate</code>	Inspects and validates the input data against data checks and returns a list of warnings and errors if applicable.

## evalml.data\_checks.DefaultDataChecks.\_\_init\_\_

DefaultDataChecks.**\_\_init\_\_** (*problem\_type, objective, n\_splits=3*)

A collection of basic data checks.

### 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.

### `evalml.data_checks.DefaultDataChecks.validate`

`DefaultDataChecks.validate` (*X*, *y=None*)

Inspects and validates the input data against data checks and returns a list of warnings and errors if applicable.

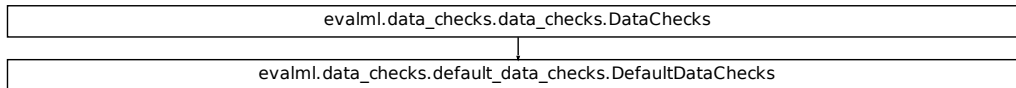
#### Parameters

- **X** (*ww.DataTable*, *pd.DataFrame*, *np.ndarray*) – The input data of shape [n\_samples, n\_features]
- **y** (*ww.DataColumn*, *pd.Series*, *np.ndarray*) – The target data of length [n\_samples]

**Returns** Dictionary containing `DataCheckMessage` objects

**Return type** dict

### Class Inheritance



## 5.12.2 Data Check Messages

<code>DataCheckMessage</code>	Base class for all <code>DataCheckMessages</code> .
<code>DataCheckError</code>	<code>DataCheckMessage</code> subclass for errors returned by data checks.
<code>DataCheckWarning</code>	<code>DataCheckMessage</code> subclass for warnings returned by data checks.

**evalml.data\_checks.DataCheckMessage**

**class** evalml.data\_checks.DataCheckMessage (*message*, *data\_check\_name*, *message\_code=None*, *details=None*)

Base class for all DataCheckMessages.

**message\_type** = None

**Methods:**

<code>__init__</code>	Message returned by a DataCheck, tagged by name.
<code>to_dict</code>	
<code>__str__</code>	String representation of data check message, equivalent to self.message attribute.
<code>__eq__</code>	Checks for equality.

**evalml.data\_checks.DataCheckMessage.\_\_init\_\_**

DataCheckMessage.**\_\_init\_\_** (*message*, *data\_check\_name*, *message\_code=None*, *details=None*)  
 Message returned by a DataCheck, tagged by name.

**Parameters**

- **message** (*str*) – Message string
- **data\_check\_name** (*str*) – Name of data check
- **message\_code** (*DataCheckMessageCode*, *optional*) – Message code associated with message.
- **details** (*dict*, *optional*) – Additional useful information associated with the message

**evalml.data\_checks.DataCheckMessage.to\_dict**

DataCheckMessage.**to\_dict** ()

**evalml.data\_checks.DataCheckMessage.\_\_str\_\_**

DataCheckMessage.**\_\_str\_\_** ()  
 String representation of data check message, equivalent to self.message attribute.

## evalml.data\_checks.DataCheckMessage.\_\_eq\_\_

DataCheckMessage.\_\_eq\_\_ (other)

Checks for equality. Two DataCheckMessage objs are considered equivalent if all of their attributes are equivalent.

## Class Inheritance

evalml.data_checks.data_check_message.DataCheckMessage
--

## evalml.data\_checks.DataCheckError

**class** evalml.data\_checks.DataCheckError (message, data\_check\_name, message\_code=None, details=None)

DataCheckMessage subclass for errors returned by data checks.

message\_type = 'error'

### Methods:

<code>__init__</code>	Message returned by a DataCheck, tagged by name.
<code>to_dict</code>	
<code>__str__</code>	String representation of data check message, equivalent to self.message attribute.
<code>__eq__</code>	Checks for equality.

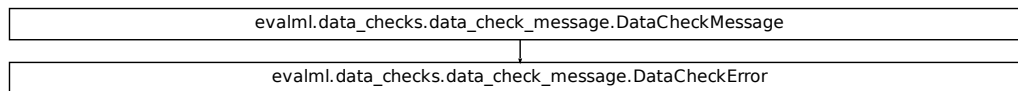
## evalml.data\_checks.DataCheckError.\_\_init\_\_

DataCheckError.\_\_init\_\_ (message, data\_check\_name, message\_code=None, details=None)

Message returned by a DataCheck, tagged by name.

### Parameters

- **message** (*str*) – Message string
- **data\_check\_name** (*str*) – Name of data check
- **message\_code** (*DataCheckMessageCode*, *optional*) – Message code associated with message.
- **details** (*dict*, *optional*) – Additional useful information associated with the message

**evalml.data\_checks.DataCheckError.to\_dict**`DataCheckError.to_dict()`**evalml.data\_checks.DataCheckError.\_\_str\_\_**`DataCheckError.__str__()`String representation of data check message, equivalent to `self.message` attribute.**evalml.data\_checks.DataCheckError.\_\_eq\_\_**`DataCheckError.__eq__(other)`Checks for equality. Two `DataCheckMessage` objs are considered equivalent if all of their attributes are equivalent.**Class Inheritance****evalml.data\_checks.DataCheckWarning**

**class** `evalml.data_checks.DataCheckWarning` (*message*, *data\_check\_name*, *message\_code=None*, *details=None*)

DataCheckMessage subclass for warnings returned by data checks.

`message_type = 'warning'`**Methods:**

<code>__init__</code>	Message returned by a DataCheck, tagged by name.
<code>to_dict</code>	
<code>__str__</code>	String representation of data check message, equivalent to <code>self.message</code> attribute.
<code>__eq__</code>	Checks for equality.

### `evalml.data_checks.DataCheckWarning.__init__`

`DataCheckWarning.__init__` (*message*, *data\_check\_name*, *message\_code=None*, *details=None*)  
Message returned by a `DataCheck`, tagged by name.

#### Parameters

- **message** (*str*) – Message string
- **data\_check\_name** (*str*) – Name of data check
- **message\_code** (`DataCheckMessageCode`, *optional*) – Message code associated with message.
- **details** (*dict*, *optional*) – Additional useful information associated with the message

### `evalml.data_checks.DataCheckWarning.to_dict`

`DataCheckWarning.to_dict` ()

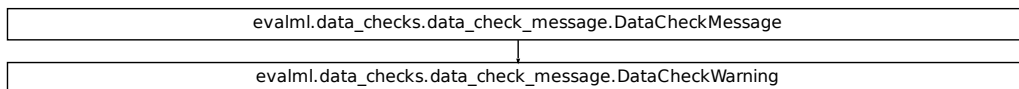
### `evalml.data_checks.DataCheckWarning.__str__`

`DataCheckWarning.__str__` ()  
String representation of data check message, equivalent to `self.message` attribute.

### `evalml.data_checks.DataCheckWarning.__eq__`

`DataCheckWarning.__eq__` (*other*)  
Checks for equality. Two `DataCheckMessage` objs are considered equivalent if all of their attributes are equivalent.

## Class Inheritance





### 5.12.3 Data Check Message Types

<i>DataCheckMessageType</i>	Enum for type of data check message: WARNING or ERROR.
-----------------------------	--

#### evalml.data\_checks.DataCheckMessageType

**class** evalml.data\_checks.DataCheckMessageType (*value*)  
Enum for type of data check message: WARNING or ERROR.

##### Attributes

ERROR	Error message returned by a data check.
WARNING	Warning message returned by a data check.

### 5.12.4 Data Check Message Codes

<i>DataCheckMessageCode</i>	Enum for data check message code.
-----------------------------	-----------------------------------

#### evalml.data\_checks.DataCheckMessageCode

**class** evalml.data\_checks.DataCheckMessageCode (*value*)  
Enum for data check message code.

##### Attributes

CLASS_IMBALANCE_BELOW_FOLDS	Message code for when the number of values for each target is below 2 * number of CV folds.
CLASS_IMBALANCE_BELOW_THRESHOLD	Message code for when balance in classes is less than the threshold.
CLASS_IMBALANCE_SEVERE	Message code for when balance in classes is less than the threshold and minimum class is less than minimum number of accepted samples.
DATETIME_HAS_NAN	Message code for when input datetime columns contain NaN values.
HAS_ID_COLUMN	Message code for data that has ID columns.
HAS_OUTLIERS	Message code for when outliers are detected.
HIGHLY_NULL	Message code for highly null columns.
HIGH_VARIANCE	Message code for when high variance is detected for cross-validation.
IS_MULTICOLLINEAR	Message code for when data is potentially multicollinear.
MISMATCHED_INDICES	Message code for when input target and features have mismatched indices.

continues on next page

Table 233 – continued from previous page

MISMATCHED_INDICES_ORDER	Message code for when input target and features have mismatched indices order.
MISMATCHED_LENGTHS	Message code for when input target and features have different lengths.
NATURAL_LANGUAGE_HAS_NAN	Message code for when input natural language columns contain NaN values.
NOT_UNIQUE_ENOUGH	Message code for when data does not possess enough unique values.
NO_VARIANCE	Message code for when data has no variance (1 unique value).
NO_VARIANCE_WITH_NULL	Message code for when data has one unique value and NaN values.
TARGET_BINARY_INVALID_VALUES	Message code for target data for a binary classification problem with numerical values not equal to {0, 1}.
TARGET_BINARY_NOT_TWO_UNIQUE_VALUES	Message code for target data for a binary classification problem that does not have two unique values.
TARGET_HAS_NULL	Message code for target data that has null values.
TARGET_INCOMPATIBLE_OBJECTIVE	Message code for target data that has incompatible values for the specified objective
TARGET_IS_EMPTY_OR_FULLY_NULL	Message code for target data that is empty or has all null values.
TARGET_IS_NONE	Message code for when target is None.
TARGET_LEAKAGE	Message code for when target leakage is detected.
TARGET_MULTICLASS_HIGH_UNIQUE_CLASS	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.
TARGET_MULTICLASS_NOT_ENOUGH_CLASSES	Message code for target data for a multi classification problem that does not have more than two unique classes.
TARGET_MULTICLASS_NOT_TWO_EXAMPLES_PER_CLASS	Message code for target data for a multi classification problem that does not have two examples per class.
TARGET_UNSUPPORTED_TYPE	Message code for target data that is of an unsupported type.
TOO_SPARSE	Message code for when multiclass data has values that are too sparsely populated.
TOO_UNIQUE	Message code for when data possesses too many unique values.

## 5.13 Utils

### 5.13.1 General Utils

<code>import_or_raise</code>	Attempts to import the requested library by name.
<code>convert_to_seconds</code>	Converts a string describing a length of time to its length in seconds.
<code>get_random_state</code>	Generates a <code>numpy.random.RandomState</code> instance using seed.

continues on next page

Table 234 – continued from previous page

<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.
<code>pad_with_nans</code>	Pad the beginning <code>num_to_pad</code> rows with nans.
<code>drop_rows_with_nans</code>	Drop rows that have any NaNs in all dataframes or series.
<code>infer_feature_types</code>	Create a Woodwork structure from the given list, pandas, or numpy input, with specified types for columns.
<code>save_plot</code>	Saves fig to filepath if specified, or to a default location if not.
<code>is_all_numeric</code>	Checks if the given <code>DataTable</code> contains only numeric values
<code>get_importable_subclasses</code>	Get importable subclasses of a base class.

### `evalml.utils.import_or_raise`

`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.convert_to_seconds`

`evalml.utils.convert_to_seconds(input_str)`

Converts a string describing a length of time to its length in seconds.

### `evalml.utils.get_random_state`

`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.get_random_seed`

`evalml.utils.get_random_seed(random_state, min_bound=0, max_bound=2147483647)`

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.pad\_with\_nans**

`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.drop\_rows\_with\_nans**

`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.infer\_feature\_types**

`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*) – 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.save\_plot

`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 is of type `plotly.Figure`. Defaults to 'png'.
- **interactive** (*bool*, *optional*) – If True and fig is of type `plotly.Figure`, saves the fig as interactive
- **static** (*bool*, *optional*) – If True, saves the fig as static
- **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.is\_all\_numeric

`evalml.utils.is_all_numeric` (*dt*)  
 Checks if the given DataTable contains only numeric values

**Parameters** *dt* (*ww.DataTable*) – The DataTable to check data types of.

**Returns** True if all the DataTable columns are numeric and are not missing any values, False otherwise.

## evalml.utils.get\_importable\_subclasses

`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.



## RELEASE NOTES

### Future Releases

- Enhancements
- Fixes
- Changes
- Documentation Changes
- Testing Changes

<b>Warning: Breaking Changes</b>
----------------------------------

### 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

- **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 `StackedEnsemble` 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](#)
- **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 `pyzmq` at less than 22.0.0 till we add support [#1756](#)
  - Remove `ProphetRegressor` from main as windows tests were flaky [#1764](#)
- **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 Featuretool 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 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 it's 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
  - Standardizing 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

## Symbols

<code>__eq__()</code> (evalml.data_checks.DataCheckError method), 539	<code>__init__()</code> (evalml.data_checks.TargetLeakageDataCheck method), 524
<code>__eq__()</code> (evalml.data_checks.DataCheckMessage method), 538	<code>__init__()</code> (evalml.objectives.AUC method), 423
<code>__eq__()</code> (evalml.data_checks.DataCheckWarning method), 540	<code>__init__()</code> (evalml.objectives.AUCMacro method), 426
<code>__init__()</code> (evalml.automl.AutoMLSearch method), 149	<code>__init__()</code> (evalml.objectives.AUCMicro method), 429
<code>__init__()</code> (evalml.automl.automl_algorithm.AutoMLAlgorithm method), 155	<code>__init__()</code> (evalml.objectives.AUCWeighted method), 431
<code>__init__()</code> (evalml.automl.automl_algorithm.IterativeAlgorithm method), 157	<code>__init__()</code> (evalml.objectives.AccuracyBinary method), 418
<code>__init__()</code> (evalml.data_checks.ClassImbalanceDataCheck method), 528	<code>__init__()</code> (evalml.objectives.AccuracyMulticlass method), 421
<code>__init__()</code> (evalml.data_checks.DataCheck method), 518	<code>__init__()</code> (evalml.objectives.BalancedAccuracyBinary method), 434
<code>__init__()</code> (evalml.data_checks.DataCheckError method), 538	<code>__init__()</code> (evalml.objectives.BalancedAccuracyMulticlass method), 437
<code>__init__()</code> (evalml.data_checks.DataCheckMessage method), 537	<code>__init__()</code> (evalml.objectives.BinaryClassificationObjective method), 398
<code>__init__()</code> (evalml.data_checks.DataCheckWarning method), 540	<code>__init__()</code> (evalml.objectives.CostBenefitMatrix method), 413
<code>__init__()</code> (evalml.data_checks.DataChecks method), 534	<code>__init__()</code> (evalml.objectives.ExpVariance method), 500
<code>__init__()</code> (evalml.data_checks.DateTimeNaNDataCheck method), 531	<code>__init__()</code> (evalml.objectives.F1 method), 439
<code>__init__()</code> (evalml.data_checks.DefaultDataChecks method), 535	<code>__init__()</code> (evalml.objectives.F1Macro method), 445
<code>__init__()</code> (evalml.data_checks.HighlyNullDataCheck method), 521	<code>__init__()</code> (evalml.objectives.F1Micro method), 442
<code>__init__()</code> (evalml.data_checks.IDColumnsDataCheck method), 522	<code>__init__()</code> (evalml.objectives.F1Weighted method), 447
<code>__init__()</code> (evalml.data_checks.InvalidTargetDataCheck method), 519	<code>__init__()</code> (evalml.objectives.FraudCost method), 407
<code>__init__()</code> (evalml.data_checks.MulticollinearityDataCheck method), 530	<code>__init__()</code> (evalml.objectives.LeadScoring method), 410
<code>__init__()</code> (evalml.data_checks.NaturalLanguageNaNDataCheck method), 533	<code>__init__()</code> (evalml.objectives.LogLossBinary method), 450
<code>__init__()</code> (evalml.data_checks.NoVarianceDataCheck method), 527	<code>__init__()</code> (evalml.objectives.LogLossMulticlass method), 453
<code>__init__()</code> (evalml.data_checks.OutliersDataCheck method), 526	<code>__init__()</code> (evalml.objectives.MAE method), 485
	<code>__init__()</code> (evalml.objectives.MAPE method), 487
	<code>__init__()</code> (evalml.objectives.MCCBinary method), 455
	<code>__init__()</code> (evalml.objectives.MCCMulticlass method), 456

- method*), 458
- `__init__()` (*evalml.objectives.MSE method*), 490
- `__init__()` (*evalml.objectives.MaxError method*), 497
- `__init__()` (*evalml.objectives.MeanSquaredLogError method*), 492
- `__init__()` (*evalml.objectives.MedianAE method*), 495
- `__init__()` (*evalml.objectives.MulticlassClassificationObjective method*), 401
- `__init__()` (*evalml.objectives.ObjectiveBase method*), 396
- `__init__()` (*evalml.objectives.Precision method*), 461
- `__init__()` (*evalml.objectives.PrecisionMacro method*), 466
- `__init__()` (*evalml.objectives.PrecisionMicro method*), 464
- `__init__()` (*evalml.objectives.PrecisionWeighted method*), 469
- `__init__()` (*evalml.objectives.R2 method*), 482
- `__init__()` (*evalml.objectives.Recall method*), 471
- `__init__()` (*evalml.objectives.RecallMacro method*), 477
- `__init__()` (*evalml.objectives.RecallMicro method*), 474
- `__init__()` (*evalml.objectives.RecallWeighted method*), 479
- `__init__()` (*evalml.objectives.RegressionObjective method*), 404
- `__init__()` (*evalml.objectives.RootMeanSquaredError method*), 502
- `__init__()` (*evalml.objectives.RootMeanSquaredLogError method*), 505
- `__init__()` (*evalml.pipelines.BaselineBinaryPipeline method*), 205
- `__init__()` (*evalml.pipelines.BaselineMulticlassPipeline method*), 210
- `__init__()` (*evalml.pipelines.BaselineRegressionPipeline method*), 225
- `__init__()` (*evalml.pipelines.BinaryClassificationPipeline method*), 170
- `__init__()` (*evalml.pipelines.ClassificationPipeline method*), 165
- `__init__()` (*evalml.pipelines.MeanBaselineRegressionPipeline method*), 230
- `__init__()` (*evalml.pipelines.ModeBaselineBinaryPipeline method*), 215
- `__init__()` (*evalml.pipelines.ModeBaselineMulticlassPipeline method*), 220
- `__init__()` (*evalml.pipelines.MulticlassClassificationPipeline method*), 175
- `__init__()` (*evalml.pipelines.PipelineBase method*), 160
- `__init__()` (*evalml.pipelines.RegressionPipeline method*), 180
- `__init__()` (*evalml.pipelines.TimeSeriesBaselineRegressionPipeline method*), 235
- `__init__()` (*evalml.pipelines.TimeSeriesBinaryClassificationPipeline method*), 189
- `__init__()` (*evalml.pipelines.TimeSeriesClassificationPipeline method*), 184
- `__init__()` (*evalml.pipelines.TimeSeriesMulticlassClassificationPipeline method*), 195
- `__init__()` (*evalml.pipelines.TimeSeriesRegressionPipeline method*), 200
- `__init__()` (*evalml.pipelines.components.ARIMARegressor method*), 344
- `__init__()` (*evalml.pipelines.components.BaselineClassifier method*), 327
- `__init__()` (*evalml.pipelines.components.BaselineRegressor method*), 365
- `__init__()` (*evalml.pipelines.components.CatBoostClassifier method*), 306
- `__init__()` (*evalml.pipelines.components.CatBoostRegressor method*), 347
- `__init__()` (*evalml.pipelines.components.ComponentBase method*), 241
- `__init__()` (*evalml.pipelines.components.DFSTransformer method*), 296
- `__init__()` (*evalml.pipelines.components.DateTimeFeaturizer method*), 286
- `__init__()` (*evalml.pipelines.components.DecisionTreeClassifier method*), 334
- `__init__()` (*evalml.pipelines.components.DecisionTreeRegressor method*), 375
- `__init__()` (*evalml.pipelines.components.DelayedFeatureTransformer method*), 293
- `__init__()` (*evalml.pipelines.components.DropColumns method*), 250
- `__init__()` (*evalml.pipelines.components.DropNullColumns method*), 283
- `__init__()` (*evalml.pipelines.components.ElasticNetClassifier method*), 309
- `__init__()` (*evalml.pipelines.components.ElasticNetRegressor method*), 350
- `__init__()` (*evalml.pipelines.components.Estimator method*), 246
- `__init__()` (*evalml.pipelines.components.ExtraTreesClassifier method*), 312
- `__init__()` (*evalml.pipelines.components.ExtraTreesRegressor method*), 356
- `__init__()` (*evalml.pipelines.components.Imputer method*), 267
- `__init__()` (*evalml.pipelines.components.KNeighborsClassifier method*), 337
- `__init__()` (*evalml.pipelines.components.LightGBMClassifier method*), 318
- `__init__()` (*evalml.pipelines.components.LightGBMRegressor method*), 318

method), 378  
 \_\_init\_\_() (evalml.pipelines.components.LinearRegressor method), 353  
 \_\_init\_\_() (evalml.pipelines.components.LogisticRegressionClassifier method), 321  
 \_\_init\_\_() (evalml.pipelines.components.OneHotEncoder method), 256  
 \_\_init\_\_() (evalml.pipelines.components.PerColumnImputer method), 264  
 \_\_init\_\_() (evalml.pipelines.components.PolynomialDetrender method), 299  
 \_\_init\_\_() (evalml.pipelines.components.RFClassifierSelectFromModel method), 280  
 \_\_init\_\_() (evalml.pipelines.components.RFRegressorSelectFromModel method), 277  
 \_\_init\_\_() (evalml.pipelines.components.RandomForestClassifier method), 315  
 \_\_init\_\_() (evalml.pipelines.components.RandomForestRegressor method), 359  
 \_\_init\_\_() (evalml.pipelines.components.SVMClassifier\_\_str\_\_() method), 340  
 \_\_init\_\_() (evalml.pipelines.components.SVMRegressor\_\_str\_\_() method), 381  
 \_\_init\_\_() (evalml.pipelines.components.SelectColumns\_\_str\_\_() method), 253  
 \_\_init\_\_() (evalml.pipelines.components.SimpleImputer method), 270  
 \_\_init\_\_() (evalml.pipelines.components.StackedEnsembleClassifier method), 331  
 \_\_init\_\_() (evalml.pipelines.components.StackedEnsembleRegressor method), 372  
 \_\_init\_\_() (evalml.pipelines.components.StandardScaler method), 273  
 \_\_init\_\_() (evalml.pipelines.components.TargetEncoder method), 260  
 \_\_init\_\_() (evalml.pipelines.components.TextFeaturizer method), 290  
 \_\_init\_\_() (evalml.pipelines.components.TimeSeriesBaselineEstimator method), 368  
 \_\_init\_\_() (evalml.pipelines.components.Transformer method), 243  
 \_\_init\_\_() (evalml.pipelines.components.Undersampler method), 302  
 \_\_init\_\_() (evalml.pipelines.components.XGBoostClassifier method), 324  
 \_\_init\_\_() (evalml.pipelines.components.XGBoostRegressor method), 362  
 \_\_init\_\_() (evalml.preprocessing.BalancedClassificationDataCVSplit method), 127  
 \_\_init\_\_() (evalml.preprocessing.BalancedClassificationDataTVSplit method), 125  
 \_\_init\_\_() (evalml.preprocessing.KMeansSMOTECVSplit method), 129  
 \_\_init\_\_() (evalml.preprocessing.KMeansSMOTETVSplit method), 131  
 \_\_init\_\_() (evalml.preprocessing.RandomUnderSamplerCVSplit method), 136  
 \_\_init\_\_() (evalml.preprocessing.RandomUnderSamplerTVSplit method), 138  
 \_\_init\_\_() (evalml.preprocessing.SMOTENCCVSplit method), 140  
 \_\_init\_\_() (evalml.preprocessing.SMOTENCTVSplit method), 141  
 \_\_init\_\_() (evalml.preprocessing.SMOTETomekCVSplit method), 132  
 \_\_init\_\_() (evalml.preprocessing.SMOTETomekTVSplit method), 134  
 \_\_init\_\_() (evalml.tuners.GridSearchTuner method), 514  
 \_\_init\_\_() (evalml.tuners.RandomSearchTuner method), 516  
 \_\_init\_\_() (evalml.tuners.SKOptTuner method), 512  
 \_\_init\_\_() (evalml.tuners.Tuner method), 510  
 \_\_str\_\_() (evalml.data\_checks.DataCheckError method), 539  
 \_\_str\_\_() (evalml.data\_checks.DataCheckMessage method), 537  
 \_\_str\_\_() (evalml.data\_checks.DataCheckWarning method), 540

## A

AccuracyBinary (class in evalml.objectives), 417  
 AccuracyMulticlass (class in evalml.objectives), 420  
 add() (evalml.tuners.GridSearchTuner method), 514  
 add() (evalml.tuners.RandomSearchTuner method), 516  
 add() (evalml.tuners.SKOptTuner method), 512  
 add() (evalml.tuners.Tuner method), 510  
 add\_result() (evalml.automl.automl\_algorithm.AutoMLAlgorithm method), 155  
 add\_to\_rankings() (evalml.automl.automl\_algorithm.IterativeAlgorithm method), 157  
 add\_to\_rankings() (evalml.automl.AutoMLSearch method), 151  
 allowed\_model\_families() (in module evalml.pipelines.components.utils), 248  
 ARIMARegressor (class in evalml.pipelines.components), 343  
 AUC (class in evalml.objectives), 422  
 AUCMacro (class in evalml.objectives), 425  
 AUCLogLoss (class in evalml.objectives), 428  
 AUCWeighted (class in evalml.objectives), 430  
 AutoMLAlgorithm (class in evalml.automl.automl\_algorithm), 154  
 AutoMLSearch (class in evalml.automl), 148  
 AutoMLSearchException (class in evalml.exceptions), 146

## B

BalancedAccuracyBinary (class in [evalml.objectives](#)), 433

BalancedAccuracyMulticlass (class in [evalml.objectives](#)), 436

BalancedClassificationDataCVSplit (class in [evalml.preprocessing](#)), 126

BalancedClassificationDataTVSplit (class in [evalml.preprocessing](#)), 125

BaselineBinaryPipeline (class in [evalml.pipelines](#)), 203

BaselineClassifier (class in [evalml.pipelines.components](#)), 326

BaselineMulticlassPipeline (class in [evalml.pipelines](#)), 209

BaselineRegressionPipeline (class in [evalml.pipelines](#)), 224

BaselineRegressor (class in [evalml.pipelines.components](#)), 364

binary\_objective\_vs\_threshold() (in module [evalml.model\\_understanding](#)), 386

BinaryClassificationObjective (class in [evalml.objectives](#)), 398

BinaryClassificationPipeline (class in [evalml.pipelines](#)), 169

calculate\_percent\_difference() (evalml.objectives.CostBenefitMatrix class method), 414

calculate\_percent\_difference() (evalml.objectives.ExpVariance class method), 500

calculate\_percent\_difference() (evalml.objectives.F1 class method), 439

calculate\_percent\_difference() (evalml.objectives.F1Macro class method), 445

calculate\_percent\_difference() (evalml.objectives.F1Micro class method), 442

calculate\_percent\_difference() (evalml.objectives.F1Weighted class method), 447

calculate\_percent\_difference() (evalml.objectives.FraudCost class method), 407

calculate\_percent\_difference() (evalml.objectives.LeadScoring class method), 410

calculate\_percent\_difference() (evalml.objectives.LogLossBinary class method), 450

## C

calculate\_percent\_difference() (evalml.objectives.AccuracyBinary class method), 418

calculate\_percent\_difference() (evalml.objectives.AccuracyMulticlass class method), 421

calculate\_percent\_difference() (evalml.objectives.AUC class method), 423

calculate\_percent\_difference() (evalml.objectives.AUCMacro class method), 426

calculate\_percent\_difference() (evalml.objectives.AUCMicro class method), 429

calculate\_percent\_difference() (evalml.objectives.AUCWeighted class method), 431

calculate\_percent\_difference() (evalml.objectives.BalancedAccuracyBinary class method), 434

calculate\_percent\_difference() (evalml.objectives.BalancedAccuracyMulticlass class method), 437

calculate\_percent\_difference() (evalml.objectives.BinaryClassificationObjective class method), 399

calculate\_percent\_difference() (evalml.objectives.LogLossMulticlass class method), 453

calculate\_percent\_difference() (evalml.objectives.MAE class method), 485

calculate\_percent\_difference() (evalml.objectives.MAPE class method), 487

calculate\_percent\_difference() (evalml.objectives.MaxError class method), 497

calculate\_percent\_difference() (evalml.objectives.MCCBinary class method), 455

calculate\_percent\_difference() (evalml.objectives.MCCMulticlass class method), 458

calculate\_percent\_difference() (evalml.objectives.MeanSquaredLogError class method), 492

calculate\_percent\_difference() (evalml.objectives.MedianAE class method), 495

calculate\_percent\_difference() (evalml.objectives.MSE class method), 490

calculate\_percent\_difference() (evalml.objectives.MulticlassClassificationObjective class method), 402



<code>calculate_percent_difference()</code> ( <i>evalml.objectives.ObjectiveBase</i> method), 396	class	<code>can_tune_threshold_with_objective()</code> ( <i>evalml.pipelines.MeanBaselineRegressionPipeline</i> method), 231
<code>calculate_percent_difference()</code> ( <i>evalml.objectives.Precision</i> class method), 461	class	<code>can_tune_threshold_with_objective()</code> ( <i>evalml.pipelines.ModeBaselineBinaryPipeline</i> method), 215
<code>calculate_percent_difference()</code> ( <i>evalml.objectives.PrecisionMacro</i> method), 466	class	<code>can_tune_threshold_with_objective()</code> ( <i>evalml.pipelines.ModeBaselineMulticlassPipeline</i> method), 220
<code>calculate_percent_difference()</code> ( <i>evalml.objectives.PrecisionMicro</i> method), 464	class	<code>can_tune_threshold_with_objective()</code> ( <i>evalml.pipelines.MulticlassClassificationPipeline</i> method), 175
<code>calculate_percent_difference()</code> ( <i>evalml.objectives.PrecisionWeighted</i> method), 469	class	<code>can_tune_threshold_with_objective()</code> ( <i>evalml.pipelines.PipelineBase</i> method), 161
<code>calculate_percent_difference()</code> ( <i>evalml.objectives.R2</i> class method), 482		<code>can_tune_threshold_with_objective()</code> ( <i>evalml.pipelines.RegressionPipeline</i> method), 180
<code>calculate_percent_difference()</code> ( <i>evalml.objectives.Recall</i> class method), 471		<code>can_tune_threshold_with_objective()</code> ( <i>evalml.pipelines.TimeSeriesBaselineRegressionPipeline</i> method), 236
<code>calculate_percent_difference()</code> ( <i>evalml.objectives.RecallMacro</i> class method), 477		<code>can_tune_threshold_with_objective()</code> ( <i>evalml.pipelines.TimeSeriesBinaryClassificationPipeline</i> method), 190
<code>calculate_percent_difference()</code> ( <i>evalml.objectives.RecallMicro</i> class method), 474		<code>can_tune_threshold_with_objective()</code> ( <i>evalml.pipelines.TimeSeriesClassificationPipeline</i> method), 185
<code>calculate_percent_difference()</code> ( <i>evalml.objectives.RecallWeighted</i> class method), 479	class	<code>can_tune_threshold_with_objective()</code> ( <i>evalml.pipelines.TimeSeriesMulticlassClassificationPipeline</i> method), 195
<code>calculate_percent_difference()</code> ( <i>evalml.objectives.RegressionObjective</i> class method), 404	class	<code>can_tune_threshold_with_objective()</code> ( <i>evalml.pipelines.TimeSeriesRegressionPipeline</i> method), 200
<code>calculate_percent_difference()</code> ( <i>evalml.objectives.RootMeanSquaredError</i> class method), 502		<code>CatBoostClassifier</code> (class in <i>evalml.pipelines.components</i> ), 305
<code>calculate_percent_difference()</code> ( <i>evalml.objectives.RootMeanSquaredLogError</i> class method), 505		<code>CatBoostRegressor</code> (class in <i>evalml.pipelines.components</i> ), 346
<code>calculate_permutation_importance()</code> (in <i>module evalml.model_understanding</i> ), 386		<code>categories()</code> ( <i>evalml.pipelines.components.OneHotEncoder</i> method), 257
<code>can_tune_threshold_with_objective()</code> ( <i>evalml.pipelines.BaselineBinaryPipeline</i> method), 205		<code>ClassificationPipeline</code> (class in <i>evalml.pipelines</i> ), 164
<code>can_tune_threshold_with_objective()</code> ( <i>evalml.pipelines.BaselineMulticlassPipeline</i> method), 210		<code>ClassImbalanceDataCheck</code> (class in <i>evalml.data_checks</i> ), 528
<code>can_tune_threshold_with_objective()</code> ( <i>evalml.pipelines.BaselineRegressionPipeline</i> method), 226		<code>clone()</code> ( <i>evalml.pipelines.BaselineBinaryPipeline</i> method), 205
<code>can_tune_threshold_with_objective()</code> ( <i>evalml.pipelines.BinaryClassificationPipeline</i> method), 170		<code>clone()</code> ( <i>evalml.pipelines.BaselineMulticlassPipeline</i> method), 210
<code>can_tune_threshold_with_objective()</code> ( <i>evalml.pipelines.ClassificationPipeline</i> method), 165		<code>clone()</code> ( <i>evalml.pipelines.BaselineRegressionPipeline</i> method), 226
		<code>clone()</code> ( <i>evalml.pipelines.BinaryClassificationPipeline</i> method), 170
		<code>clone()</code> ( <i>evalml.pipelines.ClassificationPipeline</i> method), 165
		<code>clone()</code> ( <i>evalml.pipelines.components.ARIMARegressor</i>

`method)`, 344  
`clone()` (`evalml.pipelines.components.BaselineClassifier` `method)`, 328  
`clone()` (`evalml.pipelines.components.BaselineRegressor` `method)`, 365  
`clone()` (`evalml.pipelines.components.CatBoostClassifier` `method)`, 306  
`clone()` (`evalml.pipelines.components.CatBoostRegressor` `method)`, 347  
`clone()` (`evalml.pipelines.components.ComponentBase` `method)`, 241  
`clone()` (`evalml.pipelines.components.DateTimeFeaturizer` `method)`, 287  
`clone()` (`evalml.pipelines.components.DecisionTreeClassifier` `method)`, 334  
`clone()` (`evalml.pipelines.components.DecisionTreeRegressor` `method)`, 375  
`clone()` (`evalml.pipelines.components.DelayedFeatureTransformer` `method)`, 293  
`clone()` (`evalml.pipelines.components.DFSTransformer` `method)`, 296  
`clone()` (`evalml.pipelines.components.DropColumns` `method)`, 251  
`clone()` (`evalml.pipelines.components.DropNullColumns` `method)`, 284  
`clone()` (`evalml.pipelines.components.ElasticNetClassifier` `method)`, 309  
`clone()` (`evalml.pipelines.components.ElasticNetRegressor` `method)`, 350  
`clone()` (`evalml.pipelines.components.Estimator` `method)`, 246  
`clone()` (`evalml.pipelines.components.ExtraTreesClassifier` `method)`, 312  
`clone()` (`evalml.pipelines.components.ExtraTreesRegressor` `method)`, 356  
`clone()` (`evalml.pipelines.components.Imputer` `method)`, 268  
`clone()` (`evalml.pipelines.components.KNeighborsClassifier` `method)`, 337  
`clone()` (`evalml.pipelines.components.LightGBMClassifier` `method)`, 318  
`clone()` (`evalml.pipelines.components.LightGBMRegressor` `method)`, 378  
`clone()` (`evalml.pipelines.components.LinearRegressor` `method)`, 353  
`clone()` (`evalml.pipelines.components.LogisticRegressionClassifier` `method)`, 321  
`clone()` (`evalml.pipelines.components.OneHotEncoder` `method)`, 257  
`clone()` (`evalml.pipelines.components.PerColumnImputer` `method)`, 264  
`clone()` (`evalml.pipelines.components.PolynomialDetrender` `method)`, 299  
`clone()` (`evalml.pipelines.components.RandomForestClassifier` `method)`, 315  
`clone()` (`evalml.pipelines.components.RandomForestRegressor` `method)`, 359  
`clone()` (`evalml.pipelines.components.RFClassifierSelectFromModel` `method)`, 280  
`clone()` (`evalml.pipelines.components.RFRegressorSelectFromModel` `method)`, 277  
`clone()` (`evalml.pipelines.components.SelectColumns` `method)`, 254  
`clone()` (`evalml.pipelines.components.SimpleImputer` `method)`, 271  
`clone()` (`evalml.pipelines.components.StackedEnsembleClassifier` `method)`, 331  
`clone()` (`evalml.pipelines.components.StackedEnsembleRegressor` `method)`, 372  
`clone()` (`evalml.pipelines.components.StandardScaler` `method)`, 274  
`clone()` (`evalml.pipelines.components.SVMClassifier` `method)`, 340  
`clone()` (`evalml.pipelines.components.SVMRegressor` `method)`, 381  
`clone()` (`evalml.pipelines.components.TargetEncoder` `method)`, 261  
`clone()` (`evalml.pipelines.components.TextFeaturizer` `method)`, 290  
`clone()` (`evalml.pipelines.components.TimeSeriesBaselineEstimator` `method)`, 369  
`clone()` (`evalml.pipelines.components.Transformer` `method)`, 243  
`clone()` (`evalml.pipelines.components.Undersampler` `method)`, 303  
`clone()` (`evalml.pipelines.components.XGBoostClassifier` `method)`, 324  
`clone()` (`evalml.pipelines.components.XGBoostRegressor` `method)`, 362  
`clone()` (`evalml.pipelines.components.MeanBaselineRegressionPipeline` `method)`, 231  
`clone()` (`evalml.pipelines.components.ModeBaselineBinaryPipeline` `method)`, 215  
`clone()` (`evalml.pipelines.components.ModeBaselineMulticlassPipeline` `method)`, 221  
`clone()` (`evalml.pipelines.components.MulticlassClassificationPipeline` `method)`, 175  
`clone()` (`evalml.pipelines.PipelineBase` `method)`, 161  
`clone()` (`evalml.pipelines.RegistrationPipeline` `method)`, 180  
`clone()` (`evalml.pipelines.TimeSeriesBaselineRegressionPipeline` `method)`, 236  
`clone()` (`evalml.pipelines.TimeSeriesBinaryClassificationPipeline` `method)`, 190  
`clone()` (`evalml.pipelines.TimeSeriesClassificationPipeline` `method)`, 185  
`clone()` (`evalml.pipelines.TimeSeriesMulticlassClassificationPipeline` `method)`, 195



<code>clone()</code> ( <i>evalml.pipelines.TimeSeriesRegressionPipeline</i> method), 200	<code>clone()</code> ( <i>evalml.pipelines.TimeSeriesBaselineRegressionPipeline</i> method), 236
<code>component_graph</code> ( <i>evalml.pipelines.BaselineBinaryPipeline</i> attribute), 203	<code>compute_estimator_features()</code> ( <i>evalml.pipelines.TimeSeriesBinaryClassificationPipeline</i> method), 190
<code>component_graph</code> ( <i>evalml.pipelines.BaselineMulticlassPipeline</i> attribute), 209	<code>compute_estimator_features()</code> ( <i>evalml.pipelines.TimeSeriesClassificationPipeline</i> method), 185
<code>component_graph</code> ( <i>evalml.pipelines.BaselineRegressionPipeline</i> attribute), 224	<code>compute_estimator_features()</code> ( <i>evalml.pipelines.TimeSeriesMulticlassClassificationPipeline</i> method), 195
<code>component_graph</code> ( <i>evalml.pipelines.MeanBaselineRegressionPipeline</i> attribute), 229	<code>compute_estimator_features()</code> ( <i>evalml.pipelines.TimeSeriesRegressionPipeline</i> method), 200
<code>component_graph</code> ( <i>evalml.pipelines.ModeBaselineBinaryPipeline</i> attribute), 214	<code>convert_to_seconds()</code> (in module <i>evalml.model_understanding</i> ), 384
<code>component_graph</code> ( <i>evalml.pipelines.ModeBaselineMulticlassPipeline</i> attribute), 219	<code>CostBenefitMatrix</code> (class in <i>evalml.objectives</i> ), 413
<code>component_graph</code> ( <i>evalml.pipelines.TimeSeriesBaselineRegressionPipeline</i> attribute), 234	<code>create_objectives()</code> ( <i>evalml.pipelines.BaselineBinaryPipeline</i> static method), 206
<code>ComponentBase</code> (class in <i>evalml.pipelines.components</i> ), 241	<code>create_objectives()</code> ( <i>evalml.pipelines.BaselineMulticlassPipeline</i> static method), 211
<code>ComponentNotYetFittedError</code> (class in <i>evalml.exceptions</i> ), 145	<code>create_objectives()</code> ( <i>evalml.pipelines.BaselineRegressionPipeline</i> static method), 226
<code>compute_estimator_features()</code> ( <i>evalml.pipelines.BaselineBinaryPipeline</i> method), 205	<code>create_objectives()</code> ( <i>evalml.pipelines.BinaryClassificationPipeline</i> static method), 171
<code>compute_estimator_features()</code> ( <i>evalml.pipelines.BaselineMulticlassPipeline</i> method), 210	<code>create_objectives()</code> ( <i>evalml.pipelines.ClassificationPipeline</i> static method), 166
<code>compute_estimator_features()</code> ( <i>evalml.pipelines.BaselineRegressionPipeline</i> method), 226	<code>create_objectives()</code> ( <i>evalml.pipelines.MeanBaselineRegressionPipeline</i> static method), 231
<code>compute_estimator_features()</code> ( <i>evalml.pipelines.BinaryClassificationPipeline</i> method), 170	<code>create_objectives()</code> ( <i>evalml.pipelines.ModeBaselineBinaryPipeline</i> static method), 216
<code>compute_estimator_features()</code> ( <i>evalml.pipelines.ClassificationPipeline</i> method), 165	<code>create_objectives()</code> ( <i>evalml.pipelines.ModeBaselineMulticlassPipeline</i> static method), 221
<code>compute_estimator_features()</code> ( <i>evalml.pipelines.MeanBaselineRegressionPipeline</i> method), 231	<code>create_objectives()</code> ( <i>evalml.pipelines.MulticlassClassificationPipeline</i> static method), 175
<code>compute_estimator_features()</code> ( <i>evalml.pipelines.ModeBaselineBinaryPipeline</i> method), 216	<code>create_objectives()</code> ( <i>evalml.pipelines.PipelineBase</i> static method), 161
<code>compute_estimator_features()</code> ( <i>evalml.pipelines.ModeBaselineMulticlassPipeline</i> method), 221	<code>create_objectives()</code> ( <i>evalml.pipelines.RegressionPipeline</i> static method), 181
<code>compute_estimator_features()</code> ( <i>evalml.pipelines.MulticlassClassificationPipeline</i> method), 175	<code>create_objectives()</code>
<code>compute_estimator_features()</code> ( <i>evalml.pipelines.PipelineBase</i> method), 161	
<code>compute_estimator_features()</code> ( <i>evalml.pipelines.RegressionPipeline</i> method), 180	
<code>compute_estimator_features()</code>	

(*evalml.pipelines.TimeSeriesBaselineRegressionPipeline* attribute), 159  
 static method), 236  
 custom\_name (*evalml.pipelines.RegressionPipeline* attribute), 179  
 create\_objectives() (*evalml.pipelines.TimeSeriesBinaryClassificationPipeline* static method), 190  
 custom\_name (*evalml.pipelines.TimeSeriesBaselineRegressionPipeline* attribute), 234  
 create\_objectives() (*evalml.pipelines.TimeSeriesClassificationPipeline* static method), 185  
 custom\_name (*evalml.pipelines.TimeSeriesBinaryClassificationPipeline* attribute), 188  
 create\_objectives() (*evalml.pipelines.TimeSeriesMulticlassClassificationPipeline* static method), 196  
 custom\_name (*evalml.pipelines.TimeSeriesMulticlassClassificationPipeline* attribute), 194  
 create\_objectives() (*evalml.pipelines.TimeSeriesRegressionPipeline* static method), 201  
 custom\_name (*evalml.pipelines.TimeSeriesRegressionPipeline* attribute), 199  
 custom\_hyperparameters (*evalml.pipelines.BaselineBinaryPipeline* attribute), 204  
 custom\_hyperparameters (*evalml.pipelines.BaselineMulticlassPipeline* attribute), 209  
 custom\_hyperparameters (*evalml.pipelines.BaselineRegressionPipeline* attribute), 224  
 custom\_hyperparameters (*evalml.pipelines.MeanBaselineRegressionPipeline* attribute), 229  
 custom\_hyperparameters (*evalml.pipelines.ModeBaselineBinaryPipeline* attribute), 214  
 custom\_hyperparameters (*evalml.pipelines.ModeBaselineMulticlassPipeline* attribute), 219  
 custom\_hyperparameters (*evalml.pipelines.TimeSeriesBaselineRegressionPipeline* attribute), 234  
 custom\_name (*evalml.pipelines.BaselineBinaryPipeline* attribute), 203  
 custom\_name (*evalml.pipelines.BaselineMulticlassPipeline* attribute), 209  
 custom\_name (*evalml.pipelines.BaselineRegressionPipeline* attribute), 224  
 custom\_name (*evalml.pipelines.BinaryClassificationPipeline* attribute), 169  
 custom\_name (*evalml.pipelines.ClassificationPipeline* attribute), 164  
 custom\_name (*evalml.pipelines.MeanBaselineRegressionPipeline* attribute), 229  
 custom\_name (*evalml.pipelines.ModeBaselineBinaryPipeline* attribute), 214  
 custom\_name (*evalml.pipelines.ModeBaselineMulticlassPipeline* attribute), 219  
 custom\_name (*evalml.pipelines.MulticlassClassificationPipeline* attribute), 174  
 custom\_name (*evalml.pipelines.PipelineBase* attribute), 159

**D**  
 DataCheck (class in *evalml.data\_checks*), 517  
 DataCheckError (class in *evalml.data\_checks*), 538  
 DataCheckInitError (class in *evalml.exceptions*), 147  
 DataCheckMessage (class in *evalml.data\_checks*), 537  
 DataCheckMessageCode (class in *evalml.data\_checks*), 541  
 DataCheckMessageType (class in *evalml.data\_checks*), 541  
 DataChecks (class in *evalml.data\_checks*), 534  
 DataCheckWarning (class in *evalml.data\_checks*), 539  
 DateTimeFeaturizer (class in *evalml.pipelines.components*), 286  
 DateTimeNaNDataCheck (class in *evalml.data\_checks*), 531  
 decision\_function() (*evalml.objectives.AccuracyBinary* method), 418  
 decision\_function() (*evalml.objectives.AUC* method), 423  
 decision\_function() (*evalml.objectives.BalancedAccuracyBinary* method), 434  
 decision\_function() (*evalml.objectives.BinaryClassificationObjective* method), 399  
 decision\_function() (*evalml.objectives.CostBenefitMatrix* method), 414  
 decision\_function() (*evalml.objectives.F1* method), 439  
 decision\_function() (*evalml.objectives.FraudCost* method), 407  
 decision\_function() (*evalml.objectives.LeadScoring* method), 411  
 decision\_function()

(*evalml.objectives.LogLossBinary* method), 450  
 decision\_function() (*evalml.objectives.MCCBinary* method), 455  
 decision\_function() (*evalml.objectives.Precision* method), 461  
 decision\_function() (*evalml.objectives.Recall* method), 471  
 DecisionTreeClassifier (class in *evalml.pipelines.components*), 333  
 DecisionTreeRegressor (class in *evalml.pipelines.components*), 374  
 default\_parameters (*evalml.pipelines.BaselineBinaryPipeline* attribute), 204  
 default\_parameters (*evalml.pipelines.BaselineMulticlassPipeline* attribute), 209  
 default\_parameters (*evalml.pipelines.BaselineRegressionPipeline* attribute), 224  
 default\_parameters (*evalml.pipelines.components.ARIMARegressor* attribute), 343  
 default\_parameters (*evalml.pipelines.components.BaselineClassifier* attribute), 326  
 default\_parameters (*evalml.pipelines.components.BaselineRegressor* attribute), 364  
 default\_parameters (*evalml.pipelines.components.CatBoostClassifier* attribute), 305  
 default\_parameters (*evalml.pipelines.components.CatBoostRegressor* attribute), 346  
 default\_parameters (*evalml.pipelines.components.DateTimeFeaturizer* attribute), 286  
 default\_parameters (*evalml.pipelines.components.DecisionTreeClassifier* attribute), 333  
 default\_parameters (*evalml.pipelines.components.DecisionTreeRegressor* attribute), 374  
 default\_parameters (*evalml.pipelines.components.DelayedFeatureTransformer* attribute), 292  
 default\_parameters (*evalml.pipelines.components.DFSTransformer* attribute), 295  
 default\_parameters (*evalml.pipelines.components.DropColumns* attribute), 250  
 default\_parameters (*evalml.pipelines.components.DropNullColumns* attribute), 283  
 default\_parameters (*evalml.pipelines.components.ElasticNetClassifier* attribute), 308  
 default\_parameters (*evalml.pipelines.components.ElasticNetRegressor* attribute), 349  
 default\_parameters (*evalml.pipelines.components.ExtraTreesClassifier* attribute), 311  
 default\_parameters (*evalml.pipelines.components.ExtraTreesRegressor* attribute), 355  
 default\_parameters (*evalml.pipelines.components.Imputer* attribute), 266  
 default\_parameters (*evalml.pipelines.components.KNeighborsClassifier* attribute), 336  
 default\_parameters (*evalml.pipelines.components.LightGBMClassifier* attribute), 317  
 default\_parameters (*evalml.pipelines.components.LightGBMRegressor* attribute), 377  
 default\_parameters (*evalml.pipelines.components.LinearRegressor* attribute), 352  
 default\_parameters (*evalml.pipelines.components.LogisticRegressionClassifier* attribute), 320  
 default\_parameters (*evalml.pipelines.components.OneHotEncoder* attribute), 256  
 default\_parameters (*evalml.pipelines.components.PerColumnImputer* attribute), 263  
 default\_parameters (*evalml.pipelines.components.PolynomialDetrender* attribute), 298  
 default\_parameters (*evalml.pipelines.components.RandomForestClassifier* attribute), 314  
 default\_parameters (*evalml.pipelines.components.RandomForestRegressor* attribute), 358  
 default\_parameters (*evalml.pipelines.components.RFClassifierSelectFromModel* attribute), 279  
 default\_parameters (*evalml.pipelines.components.RFRegressorSelectFromModel* attribute), 279

- attribute*), 276
- default\_parameters (*evalml.pipelines.components.SelectColumns attribute*), 253
- default\_parameters (*evalml.pipelines.components.SimpleImputer attribute*), 270
- default\_parameters (*evalml.pipelines.components.StackedEnsembleClassifier attribute*), 330
- default\_parameters (*evalml.pipelines.components.StackedEnsembleRegressor attribute*), 371
- default\_parameters (*evalml.pipelines.components.StandardScaler attribute*), 273
- default\_parameters (*evalml.pipelines.components.SVMClassifier attribute*), 339
- default\_parameters (*evalml.pipelines.components.SVMRegressor attribute*), 380
- default\_parameters (*evalml.pipelines.components.TargetEncoder attribute*), 260
- default\_parameters (*evalml.pipelines.components.TextFeaturizer attribute*), 289
- default\_parameters (*evalml.pipelines.components.TimeSeriesBaselineEstimator attribute*), 367
- default\_parameters (*evalml.pipelines.components.Undersampler attribute*), 302
- default\_parameters (*evalml.pipelines.components.XGBoostClassifier attribute*), 323
- default\_parameters (*evalml.pipelines.components.XGBoostRegressor attribute*), 361
- default\_parameters (*evalml.pipelines.ModeBaselineRegressionPipeline attribute*), 229
- default\_parameters (*evalml.pipelines.ModeBaselineBinaryPipeline attribute*), 214
- default\_parameters (*evalml.pipelines.ModeBaselineMulticlassPipeline attribute*), 219
- default\_parameters (*evalml.pipelines.TimeSeriesBaselineRegressionPipeline attribute*), 234
- DefaultDataChecks (class in *evalml.data\_checks*), 535
- DelayedFeatureTransformer (class in *evalml.pipelines.components*), 292
- describe() (*evalml.pipelines.BaselineBinaryPipeline method*), 206
- describe() (*evalml.pipelines.BaselineMulticlassPipeline method*), 211
- describe() (*evalml.pipelines.BaselineRegressionPipeline method*), 226
- describe() (*evalml.pipelines.BinaryClassificationPipeline method*), 171
- describe() (*evalml.pipelines.ClassificationPipeline method*), 166
- describe() (*evalml.pipelines.components.ARIMAREgressor method*), 344
- describe() (*evalml.pipelines.components.BaselineClassifier method*), 328
- describe() (*evalml.pipelines.components.BaselineRegressor method*), 365
- describe() (*evalml.pipelines.components.CatBoostClassifier method*), 306
- describe() (*evalml.pipelines.components.CatBoostRegressor method*), 347
- describe() (*evalml.pipelines.components.ComponentBase method*), 241
- describe() (*evalml.pipelines.components.DateTimeFeaturizer method*), 287
- describe() (*evalml.pipelines.components.DecisionTreeClassifier method*), 334
- describe() (*evalml.pipelines.components.DecisionTreeRegressor method*), 375
- describe() (*evalml.pipelines.components.DelayedFeatureTransformer method*), 293
- describe() (*evalml.pipelines.components.DFSTransformer method*), 296
- describe() (*evalml.pipelines.components.DropColumns method*), 251
- describe() (*evalml.pipelines.components.DropNullColumns method*), 284
- describe() (*evalml.pipelines.components.ElasticNetClassifier method*), 309
- describe() (*evalml.pipelines.components.ElasticNetRegressor method*), 350
- describe() (*evalml.pipelines.components.Estimator method*), 246
- describe() (*evalml.pipelines.components.ExtraTreesClassifier method*), 312
- describe() (*evalml.pipelines.components.ExtraTreesRegressor method*), 356
- describe() (*evalml.pipelines.components.Imputer method*), 268
- describe() (*evalml.pipelines.components.KNeighborsClassifier method*), 337
- describe() (*evalml.pipelines.components.LightGBMClassifier method*), 318

[describe\(\) \(evalml.pipelines.components.LightGBMRegressor method\), 379](#)  
[describe\(\) \(evalml.pipelines.components.LinearRegressor method\), 353](#)  
[describe\(\) \(evalml.pipelines.components.LogisticRegressionClassifier method\), 321](#)  
[describe\(\) \(evalml.pipelines.components.OneHotEncoder method\), 257](#)  
[describe\(\) \(evalml.pipelines.components.PerColumnImputer method\), 264](#)  
[describe\(\) \(evalml.pipelines.components.PolynomialDegreeReducer method\), 299](#)  
[describe\(\) \(evalml.pipelines.components.RandomForestClassifier method\), 315](#)  
[describe\(\) \(evalml.pipelines.components.RandomForestRegressor method\), 359](#)  
[describe\(\) \(evalml.pipelines.components.RFClassifierSelectFromModel method\), 280](#)  
[describe\(\) \(evalml.pipelines.components.RFRegressorSelectFromModel method\), 277](#)  
[describe\(\) \(evalml.pipelines.components.SelectColumns method\), 254](#)  
[describe\(\) \(evalml.pipelines.components.SimpleImputer method\), 271](#)  
[describe\(\) \(evalml.pipelines.components.StackedEnsembleClassifier method\), 331](#)  
[describe\(\) \(evalml.pipelines.components.StackedEnsembleRegressor method\), 372](#)  
[describe\(\) \(evalml.pipelines.components.StandardScaler method\), 274](#)  
[describe\(\) \(evalml.pipelines.components.SVMClassifier method\), 340](#)  
[describe\(\) \(evalml.pipelines.components.SVMRegressor method\), 381](#)  
[describe\(\) \(evalml.pipelines.components.TargetEncoder method\), 261](#)  
[describe\(\) \(evalml.pipelines.components.TextFeaturizer method\), 290](#)  
[describe\(\) \(evalml.pipelines.components.TimeSeriesBaselineEstimator method\), 369](#)  
[describe\(\) \(evalml.pipelines.components.Transformer method\), 243](#)  
[describe\(\) \(evalml.pipelines.components.Undersampler method\), 303](#)  
[describe\(\) \(evalml.pipelines.components.XGBoostClassifier method\), 324](#)  
[describe\(\) \(evalml.pipelines.components.XGBoostRegressor method\), 362](#)  
[describe\(\) \(evalml.pipelines.ModeBaselineRegressionPipeline method\), 231](#)  
[describe\(\) \(evalml.pipelines.ModeBaselineBinaryPipeline method\), 216](#)  
[describe\(\) \(evalml.pipelines.ModeBaselineMulticlassPipeline method\), 221](#)  
[describe\(\) \(evalml.pipelines.MulticlassClassificationPipeline method\), 176](#)  
[describe\(\) \(evalml.pipelines.PipelineBase method\), 161](#)  
[describe\(\) \(evalml.pipelines.RegressionPipeline method\), 181](#)  
[describe\(\) \(evalml.pipelines.TimeSeriesBaselineRegressionPipeline method\), 236](#)  
[describe\(\) \(evalml.pipelines.TimeSeriesBinaryClassificationPipeline method\), 190](#)  
[describe\(\) \(evalml.pipelines.TimeSeriesClassificationPipeline method\), 185](#)  
[describe\(\) \(evalml.pipelines.TimeSeriesMulticlassClassificationPipeline method\), 196](#)  
[describe\(\) \(evalml.pipelines.TimeSeriesRegressionPipeline method\), 201](#)  
[describe\(\) \(evalml.pipelines.TimeSeriesRegressionPipeline method\), 201](#)  
[describe\(\) \(evalml.pipelines.AutoMLSearch method\), 508](#)  
[detect\\_problem\\_type\(\) \(in module evalml.problem\\_types\), 508](#)  
[DFSTransformer \(class in evalml.pipelines.components\), 295](#)  
[drop\\_nan\\_target\\_rows\(\) \(in module evalml.preprocessing\), 123](#)  
[drop\\_rows\\_with\\_nans\(\) \(in module evalml.utils\), 544](#)  
[DropColumns \(class in evalml.pipelines.components\), 250](#)  
[DropNullColumns \(class in evalml.pipelines.components\), 283](#)  

## E

[ElasticNetClassifier \(class in evalml.pipelines.components\), 308](#)  
[ElasticNetRegressor \(class in evalml.pipelines.components\), 349](#)  
[EnsembleMissingPipelinesError \(class in evalml.exceptions\), 146](#)  
[Estimator \(class in evalml.pipelines.components\), 245](#)  
[explain\\_predictions\(\) \(in module evalml.model\\_understanding.prediction\\_explanations\), 394](#)  
[explain\\_predictions\\_best\\_worst\(\) \(in module evalml.model\\_understanding.prediction\\_explanations\), 395](#)  
[ExpVariance \(class in evalml.objectives\), 499](#)  
[ExtraTreesClassifier \(class in evalml.pipelines.components\), 311](#)  
[ExtraTreesRegressor \(class in evalml.pipelines.components\), 355](#)  

## F

[F1 \(class in evalml.objectives\), 438](#)



F1Macro (class in evalml.objectives), 444  
 F1Micro (class in evalml.objectives), 441  
 F1Weighted (class in evalml.objectives), 446  
 fit () (evalml.pipelines.BaselineBinaryPipeline method), 206  
 fit () (evalml.pipelines.BaselineMulticlassPipeline method), 211  
 fit () (evalml.pipelines.BaselineRegressionPipeline method), 227  
 fit () (evalml.pipelines.BinaryClassificationPipeline method), 171  
 fit () (evalml.pipelines.ClassificationPipeline method), 166  
 fit () (evalml.pipelines.components.ARIMARegressor method), 344  
 fit () (evalml.pipelines.components.BaselineClassifier method), 328  
 fit () (evalml.pipelines.components.BaselineRegressor method), 366  
 fit () (evalml.pipelines.components.CatBoostClassifier method), 306  
 fit () (evalml.pipelines.components.CatBoostRegressor method), 347  
 fit () (evalml.pipelines.components.ComponentBase method), 242  
 fit () (evalml.pipelines.components.DateTimeFeaturizer method), 287  
 fit () (evalml.pipelines.components.DecisionTreeClassifier method), 335  
 fit () (evalml.pipelines.components.DecisionTreeRegressor method), 376  
 fit () (evalml.pipelines.components.DelayedFeatureTransformer method), 294  
 fit () (evalml.pipelines.components.DFSTransformer method), 297  
 fit () (evalml.pipelines.components.DropColumns method), 251  
 fit () (evalml.pipelines.components.DropNullColumns method), 284  
 fit () (evalml.pipelines.components.ElasticNetClassifier method), 309  
 fit () (evalml.pipelines.components.ElasticNetRegressor method), 350  
 fit () (evalml.pipelines.components.Estimator method), 246  
 fit () (evalml.pipelines.components.ExtraTreesClassifier method), 312  
 fit () (evalml.pipelines.components.ExtraTreesRegressor method), 356  
 fit () (evalml.pipelines.components.Imputer method), 268  
 fit () (evalml.pipelines.components.KNeighborsClassifier method), 338  
 fit () (evalml.pipelines.components.LightGBMClassifier method), 319  
 fit () (evalml.pipelines.components.LightGBMRegressor method), 379  
 fit () (evalml.pipelines.components.LinearRegressor method), 353  
 fit () (evalml.pipelines.components.LogisticRegressionClassifier method), 322  
 fit () (evalml.pipelines.components.OneHotEncoder method), 258  
 fit () (evalml.pipelines.components.PerColumnImputer method), 265  
 fit () (evalml.pipelines.components.PolynomialDetrender method), 300  
 fit () (evalml.pipelines.components.RandomForestClassifier method), 315  
 fit () (evalml.pipelines.components.RandomForestRegressor method), 359  
 fit () (evalml.pipelines.components.RFClassifierSelectFromModel method), 281  
 fit () (evalml.pipelines.components.RFRegressorSelectFromModel method), 277  
 fit () (evalml.pipelines.components.SelectColumns method), 254  
 fit () (evalml.pipelines.components.SimpleImputer method), 271  
 fit () (evalml.pipelines.components.StackedEnsembleClassifier method), 332  
 fit () (evalml.pipelines.components.StackedEnsembleRegressor method), 373  
 fit () (evalml.pipelines.components.StandardScaler method), 274  
 fit () (evalml.pipelines.components.SVMClassifier method), 341  
 fit () (evalml.pipelines.components.SVMRegressor method), 382  
 fit () (evalml.pipelines.components.TargetEncoder method), 261  
 fit () (evalml.pipelines.components.TextFeaturizer method), 290  
 fit () (evalml.pipelines.components.TimeSeriesBaselineEstimator method), 369  
 fit () (evalml.pipelines.components.Transformer method), 244  
 fit () (evalml.pipelines.components.Undersampler method), 303  
 fit () (evalml.pipelines.components.XGBoostClassifier method), 325  
 fit () (evalml.pipelines.components.XGBoostRegressor method), 362  
 fit () (evalml.pipelines.MeanBaselineRegressionPipeline method), 232  
 fit () (evalml.pipelines.ModeBaselineBinaryPipeline method), 216  
 fit () (evalml.pipelines.ModeBaselineMulticlassPipeline

- method), 221
- fit() (evalml.pipelines.MulticlassClassificationPipeline method), 176
- fit() (evalml.pipelines.PipelineBase method), 162
- fit() (evalml.pipelines.RegressionPipeline method), 181
- fit() (evalml.pipelines.TimeSeriesBaselineRegressionPipeline method), 237
- fit() (evalml.pipelines.TimeSeriesBinaryClassificationPipeline method), 191
- fit() (evalml.pipelines.TimeSeriesClassificationPipeline method), 186
- fit() (evalml.pipelines.TimeSeriesMulticlassClassificationPipeline method), 196
- fit() (evalml.pipelines.TimeSeriesRegressionPipeline method), 201
- fit\_transform() (evalml.pipelines.components.DateTimeFeaturizer method), 287
- fit\_transform() (evalml.pipelines.components.DelayedFeatureTransformer method), 294
- fit\_transform() (evalml.pipelines.components.DFSTransformer method), 297
- fit\_transform() (evalml.pipelines.components.DropColumns method), 251
- fit\_transform() (evalml.pipelines.components.DropNullColumns method), 284
- fit\_transform() (evalml.pipelines.components.Imputer method), 268
- fit\_transform() (evalml.pipelines.components.OneHotEncoder method), 258
- fit\_transform() (evalml.pipelines.components.PerColumnImputer method), 265
- fit\_transform() (evalml.pipelines.components.PolynomialDetrender method), 300
- fit\_transform() (evalml.pipelines.components.RFClassifierSelectFromModel method), 281
- fit\_transform() (evalml.pipelines.components.RFRegressorSelectFromModel method), 278
- fit\_transform() (evalml.pipelines.components.SelectColumns method), 254
- fit\_transform() (evalml.pipelines.components.SimpleImputer method), 271
- fit\_transform() (evalml.pipelines.components.StandardScaler method), 274
- fit\_transform() (evalml.pipelines.components.TargetEncoder method), 262
- fit\_transform() (evalml.pipelines.components.TextFeaturizer method), 290
- fit\_transform() (evalml.pipelines.components.Transformer method), 244
- fit\_transform() (evalml.pipelines.components.Undersampler method), 303
- FraudCost (class in evalml.objectives), 406
- G**
- generate\_component\_code() (in module evalml.pipelines.components.utils), 249
- generate\_pipeline\_code() (in module evalml.pipelines.utils), 240
- get\_all\_objective\_names() (in module evalml.objectives), 507
- get\_component() (evalml.pipelines.BaselineBinaryPipeline method), 206
- get\_component() (evalml.pipelines.BaselineMulticlassPipeline method), 211
- get\_component() (evalml.pipelines.BaselineRegressionPipeline method), 227
- get\_component() (evalml.pipelines.BinaryClassificationPipeline method), 171
- get\_component() (evalml.pipelines.ClassificationPipeline method), 166
- get\_component() (evalml.pipelines.MeanBaselineRegressionPipeline method), 252
- get\_component() (evalml.pipelines.ModeBaselineBinaryPipeline method), 216
- get\_component() (evalml.pipelines.ModeBaselineMulticlassPipeline method), 222
- get\_component() (evalml.pipelines.MulticlassClassificationPipeline method), 176
- get\_component() (evalml.pipelines.PipelineBase method), 162
- get\_component() (evalml.pipelines.RegressionPipeline method), 181
- get\_component() (evalml.pipelines.TimeSeriesBaselineRegressionPipeline method), 237
- get\_component() (evalml.pipelines.TimeSeriesBinaryClassificationPipeline method), 191
- get\_component() (evalml.pipelines.TimeSeriesClassificationPipeline method), 186
- get\_component() (evalml.pipelines.TimeSeriesMulticlassClassificationPipeline method), 196
- get\_component() (evalml.pipelines.TimeSeriesRegressionPipeline method), 201
- get\_core\_objective\_names() (in module evalml.objectives), 507
- get\_core\_objectives() (in module evalml.objectives), 507
- get\_default\_primary\_search\_objective() (in module evalml.automl), 153
- get\_estimators() (in module evalml.pipelines.components.utils), 248
- get\_feature\_names() (evalml.pipelines.components.DateTimeFeaturizer method), 288
- get\_feature\_names() (evalml.pipelines.components.OneHotEncoder method), 258
- get\_feature\_names()

(*evalml.pipelines.components.TargetEncoder* method), 262

*get\_importable\_subclasses()* (in module *evalml.utils*), 545

*get\_linear\_coefficients()* (in module *evalml.model\_understanding*), 388

*get\_n\_splits()* (*evalml.preprocessing.BalancedClassificationDataCVSplit* method), 127

*get\_n\_splits()* (*evalml.preprocessing.BalancedClassificationDataTVSplit* method), 126

*get\_n\_splits()* (*evalml.preprocessing.KMeansSMOTENCVSplit* method), 129

*get\_n\_splits()* (*evalml.preprocessing.KMeansSMOTECTVSplit* method), 131

*get\_n\_splits()* (*evalml.preprocessing.RandomUnderSamplerCVSplit* method), 136

*get\_n\_splits()* (*evalml.preprocessing.RandomUnderSamplerTVSplit* method), 138

*get\_n\_splits()* (*evalml.preprocessing.SMOTENCCVSplit* method), 140

*get\_n\_splits()* (*evalml.preprocessing.SMOTENCTVSplit* method), 142

*get\_n\_splits()* (*evalml.preprocessing.SMOTETomekCVSplit* method), 133

*get\_n\_splits()* (*evalml.preprocessing.SMOTETomekTVSplit* method), 135

*get\_names()* (*evalml.pipelines.components.RFClassifierSelectFromModel* method), 281

*get\_names()* (*evalml.pipelines.components.RFRegressorSelectFromModel* method), 278

*get\_non\_core\_objectives()* (in module *evalml.objectives*), 507

*get\_objective()* (in module *evalml.objectives*), 507

*get\_pipeline()* (*evalml.automl.AutoMLSearch* method), 151

*get\_prediction\_vs\_actual\_data()* (in module *evalml.model\_understanding*), 388

*get\_prediction\_vs\_actual\_over\_time\_data()* (in module *evalml.model\_understanding*), 387

*get\_random\_seed()* (in module *evalml.utils*), 543

*get\_random\_state()* (in module *evalml.utils*), 543

*graph()* (*evalml.pipelines.BaselineBinaryPipeline* method), 206

*graph()* (*evalml.pipelines.BaselineMulticlassPipeline* method), 211

*graph()* (*evalml.pipelines.BaselineRegressionPipeline* method), 227

*graph()* (*evalml.pipelines.BinaryClassificationPipeline* method), 171

*graph()* (*evalml.pipelines.ClassificationPipeline* method), 166

*graph()* (*evalml.pipelines.MeanBaselineRegressionPipeline* method), 232

*graph()* (*evalml.pipelines.ModeBaselineBinaryPipeline* method), 217

*graph()* (*evalml.pipelines.ModeBaselineMulticlassPipeline* method), 222

*graph()* (*evalml.pipelines.MulticlassClassificationPipeline* method), 176

*graph()* (*evalml.pipelines.PipelineBase* method), 162

*graph()* (*evalml.pipelines.RegressionPipeline* method), 181

*graph()* (*evalml.pipelines.TimeSeriesBaselineRegressionPipeline* method), 237

*graph()* (*evalml.pipelines.TimeSeriesBinaryClassificationPipeline* method), 191

*graph()* (*evalml.pipelines.TimeSeriesClassificationPipeline* method), 186

*graph()* (*evalml.pipelines.TimeSeriesMulticlassClassificationPipeline* method), 196

*graph()* (*evalml.pipelines.TimeSeriesRegressionPipeline* method), 201

*graph\_binary\_objective\_vs\_threshold()* (in module *evalml.model\_understanding*), 391

*graph\_confusion\_matrix()* (in module *evalml.model\_understanding*), 390

*graph\_feature\_importance()* (*evalml.pipelines.BaselineBinaryPipeline* method), 207

*graph\_feature\_importance()* (*evalml.pipelines.BaselineMulticlassPipeline* method), 212

*graph\_feature\_importance()* (*evalml.pipelines.BaselineRegressionPipeline* method), 227

*graph\_feature\_importance()* (*evalml.pipelines.BinaryClassificationPipeline* method), 172

*graph\_feature\_importance()* (*evalml.pipelines.ClassificationPipeline* method), 167

*graph\_feature\_importance()* (*evalml.pipelines.MeanBaselineRegressionPipeline* method), 232

*graph\_feature\_importance()* (*evalml.pipelines.ModeBaselineBinaryPipeline* method), 217

*graph\_feature\_importance()* (*evalml.pipelines.ModeBaselineMulticlassPipeline* method), 222

*graph\_feature\_importance()* (*evalml.pipelines.MulticlassClassificationPipeline* method), 177

*graph\_feature\_importance()* (*evalml.pipelines.PipelineBase* method), 162

*graph\_feature\_importance()* (*evalml.pipelines.RegressionPipeline* method),



182

`graph_feature_importance()`  
(`evalml.pipelines.TimeSeriesBaselineRegressionPipeline`  
`method`), 237

`graph_feature_importance()`  
(`evalml.pipelines.TimeSeriesBinaryClassificationPipeline`  
`method`), 191

`graph_feature_importance()`  
(`evalml.pipelines.TimeSeriesClassificationPipeline`  
`method`), 186

`graph_feature_importance()`  
(`evalml.pipelines.TimeSeriesMulticlassClassificationPipeline`  
`method`), 197

`graph_feature_importance()`  
(`evalml.pipelines.TimeSeriesRegressionPipeline`  
`method`), 202

`graph_partial_dependence()` (in module  
`evalml.model_understanding`), 392

`graph_permutation_importance()` (in module  
`evalml.model_understanding`), 390

`graph_precision_recall_curve()` (in module  
`evalml.model_understanding`), 389

`graph_prediction_vs_actual()` (in module  
`evalml.model_understanding`), 391

`graph_prediction_vs_actual_over_time()`  
(in module `evalml.model_understanding`), 392

`graph_roc_curve()` (in module  
`evalml.model_understanding`), 390

`graph_t_sne()` (in module  
`evalml.model_understanding`), 393

`greater_is_better`  
(`evalml.objectives.AccuracyBinary` attribute),  
417

`greater_is_better`  
(`evalml.objectives.AccuracyMulticlass` at-  
tribute), 420

`greater_is_better` (`evalml.objectives.AUC` at-  
tribute), 422

`greater_is_better` (`evalml.objectives.AUCMacro`  
attribute), 425

`greater_is_better` (`evalml.objectives.AUCMicro`  
attribute), 428

`greater_is_better`  
(`evalml.objectives.AUCWeighted` attribute),  
430

`greater_is_better`  
(`evalml.objectives.BalancedAccuracyBinary`  
attribute), 433

`greater_is_better`  
(`evalml.objectives.BalancedAccuracyMulticlass`  
attribute), 436

`greater_is_better`  
(`evalml.objectives.CostBenefitMatrix` at-  
tribute), 413

`greater_is_better` (`evalml.objectives.ExpVariance`  
attribute), 499

`greater_is_better` (`evalml.objectives.F1` at-  
tribute), 438

`greater_is_better` (`evalml.objectives.F1Macro` at-  
tribute), 444

`greater_is_better` (`evalml.objectives.F1Micro` at-  
tribute), 441

`greater_is_better` (`evalml.objectives.F1Weighted`  
attribute), 446

`greater_is_better` (`evalml.objectives.FraudCost`  
attribute), 406

`greater_is_better` (`evalml.objectives.LeadScoring`  
attribute), 409

`greater_is_better`  
(`evalml.objectives.LogLossBinary` attribute),  
449

`greater_is_better`  
(`evalml.objectives.LogLossMulticlass` at-  
tribute), 452

`greater_is_better` (`evalml.objectives.MAE` at-  
tribute), 484

`greater_is_better` (`evalml.objectives.MAPE` at-  
tribute), 486

`greater_is_better` (`evalml.objectives.MaxError`  
attribute), 496

`greater_is_better` (`evalml.objectives.MCCBinary`  
attribute), 454

`greater_is_better`  
(`evalml.objectives.MCCMulticlass` attribute),  
457

`greater_is_better`  
(`evalml.objectives.MeanSquaredLogError`  
attribute), 491

`greater_is_better` (`evalml.objectives.MedianAE`  
attribute), 494

`greater_is_better` (`evalml.objectives.MSE` at-  
tribute), 489

`greater_is_better` (`evalml.objectives.Precision` at-  
tribute), 460

`greater_is_better`  
(`evalml.objectives.PrecisionMacro` attribute),  
465

`greater_is_better`  
(`evalml.objectives.PrecisionMicro` attribute),  
463

`greater_is_better`  
(`evalml.objectives.PrecisionWeighted` at-  
tribute), 468

`greater_is_better` (`evalml.objectives.R2` at-  
tribute), 481

`greater_is_better` (`evalml.objectives.Recall` at-  
tribute), 470

`greater_is_better`

(*evalml.objectives.RecallMacro* attribute), 476  
 greater\_is\_better (*evalml.objectives.RecallMicro* attribute), 473  
 greater\_is\_better (*evalml.objectives.RecallWeighted* attribute), 478  
 greater\_is\_better (*evalml.objectives.RootMeanSquaredError* attribute), 501  
 greater\_is\_better (*evalml.objectives.RootMeanSquaredLogError* attribute), 504  
 GridSearchTuner (class in *evalml.tuners*), 513

## H

handle\_model\_family() (in module *evalml.model\_family*), 509  
 handle\_problem\_types() (in module *evalml.problem\_types*), 508  
 HighlyNullDataCheck (class in *evalml.data\_checks*), 520  
 hyperparameter\_ranges (*evalml.pipelines.components.ARIMARegressor* attribute), 343  
 hyperparameter\_ranges (*evalml.pipelines.components.BaselineClassifier* attribute), 326  
 hyperparameter\_ranges (*evalml.pipelines.components.BaselineRegressor* attribute), 364  
 hyperparameter\_ranges (*evalml.pipelines.components.CatBoostClassifier* attribute), 305  
 hyperparameter\_ranges (*evalml.pipelines.components.CatBoostRegressor* attribute), 346  
 hyperparameter\_ranges (*evalml.pipelines.components.DateTimeFeaturizer* attribute), 286  
 hyperparameter\_ranges (*evalml.pipelines.components.DecisionTreeClassifier* attribute), 333  
 hyperparameter\_ranges (*evalml.pipelines.components.DecisionTreeRegressor* attribute), 374  
 hyperparameter\_ranges (*evalml.pipelines.components.DelayedFeatureTransformer* attribute), 292  
 hyperparameter\_ranges (*evalml.pipelines.components.DFSTransformer* attribute), 295  
 hyperparameter\_ranges (*evalml.pipelines.components.DropColumns* attribute), 250  
 hyperparameter\_ranges (*evalml.pipelines.components.DropNullColumns* attribute), 283  
 hyperparameter\_ranges (*evalml.pipelines.components.ElasticNetClassifier* attribute), 308  
 hyperparameter\_ranges (*evalml.pipelines.components.ElasticNetRegressor* attribute), 349  
 hyperparameter\_ranges (*evalml.pipelines.components.ExtraTreesClassifier* attribute), 311  
 hyperparameter\_ranges (*evalml.pipelines.components.ExtraTreesRegressor* attribute), 355  
 hyperparameter\_ranges (*evalml.pipelines.components.Imputer* attribute), 266  
 hyperparameter\_ranges (*evalml.pipelines.components.KNeighborsClassifier* attribute), 336  
 hyperparameter\_ranges (*evalml.pipelines.components.LightGBMClassifier* attribute), 317  
 hyperparameter\_ranges (*evalml.pipelines.components.LightGBMRegressor* attribute), 377  
 hyperparameter\_ranges (*evalml.pipelines.components.LinearRegressor* attribute), 352  
 hyperparameter\_ranges (*evalml.pipelines.components.LogisticRegressionClassifier* attribute), 320  
 hyperparameter\_ranges (*evalml.pipelines.components.OneHotEncoder* attribute), 256  
 hyperparameter\_ranges (*evalml.pipelines.components.PerColumnImputer* attribute), 263  
 hyperparameter\_ranges (*evalml.pipelines.components.PolynomialDetrender* attribute), 298  
 hyperparameter\_ranges (*evalml.pipelines.components.RandomForestClassifier* attribute), 314  
 hyperparameter\_ranges (*evalml.pipelines.components.RandomForestRegressor* attribute), 358  
 hyperparameter\_ranges (*evalml.pipelines.components.RFClassifierSelectFromModel* attribute), 279  
 hyperparameter\_ranges (*evalml.pipelines.components.RFRegressorSelectFromModel* attribute), 279

[attribute](#)), 276  
[hyperparameter\\_ranges](#) ([evalml.pipelines.components.SelectColumns](#) [attribute](#)), 253  
[hyperparameter\\_ranges](#) ([evalml.pipelines.components.SimpleImputer](#) [attribute](#)), 270  
[hyperparameter\\_ranges](#) ([evalml.pipelines.components.StackedEnsembleClassifier](#) [attribute](#)), 330  
[hyperparameter\\_ranges](#) ([evalml.pipelines.components.StackedEnsembleRegressor](#) [attribute](#)), 371  
[hyperparameter\\_ranges](#) ([evalml.pipelines.components.StandardScaler](#) [attribute](#)), 273  
[hyperparameter\\_ranges](#) ([evalml.pipelines.components.SVMClassifier](#) [attribute](#)), 339  
[hyperparameter\\_ranges](#) ([evalml.pipelines.components.SVMRegressor](#) [attribute](#)), 380  
[hyperparameter\\_ranges](#) ([evalml.pipelines.components.TargetEncoder](#) [attribute](#)), 260  
[hyperparameter\\_ranges](#) ([evalml.pipelines.components.TextFeaturizer](#) [attribute](#)), 289  
[hyperparameter\\_ranges](#) ([evalml.pipelines.components.TimeSeriesBaselineEstimator](#) [attribute](#)), 367  
[hyperparameter\\_ranges](#) ([evalml.pipelines.components.Undersampler](#) [attribute](#)), 302  
[hyperparameter\\_ranges](#) ([evalml.pipelines.components.XGBoostClassifier](#) [attribute](#)), 323  
[hyperparameter\\_ranges](#) ([evalml.pipelines.components.XGBoostRegressor](#) [attribute](#)), 361  
[hyperparameters](#) ([evalml.pipelines.BaselineBinaryPipeline](#) [attribute](#)), 204  
[hyperparameters](#) ([evalml.pipelines.BaselineMulticlassPipeline](#) [attribute](#)), 209  
[hyperparameters](#) ([evalml.pipelines.BaselineRegressionPipeline](#) [attribute](#)), 224  
[hyperparameters](#) ([evalml.pipelines.MeanBaselineRegressionPipeline](#) [attribute](#)), 229  
[hyperparameters](#) ([evalml.pipelines.ModeBaselineBinaryPipeline](#) [attribute](#)), 214  
[hyperparameters](#) ([evalml.pipelines.ModeBaselineMulticlassPipeline](#) [attribute](#)), 219  
[hyperparameters](#) ([evalml.pipelines.TimeSeriesBaselineRegressionPipeline](#) [attribute](#)), 234

**I**  
[IDColumnsDataCheck](#) (class in [evalml.data\\_checks](#)), 522  
[IllFormattedClassNameError](#) (class in [evalml.exceptions](#)), 144  
[import\\_or\\_raise\(\)](#) (in module [evalml.utils](#)), 543  
[Imputer](#) (class in [evalml.pipelines.components](#)), 266  
[infer\\_feature\\_types\(\)](#) (in module [evalml.utils](#)), 544  
[InvalidTargetDataCheck](#) (class in [evalml.data\\_checks](#)), 519  
[inverse\\_transform\(\)](#) ([evalml.pipelines.components.PolynomialDetrender](#) [method](#)), 300  
[is\\_all\\_numeric\(\)](#) (in module [evalml.utils](#)), 545  
[is\\_defined\\_for\\_problem\\_type\(\)](#) ([evalml.objectives.AccuracyBinary](#) [class method](#)), 418  
[is\\_defined\\_for\\_problem\\_type\(\)](#) ([evalml.objectives.AccuracyMulticlass](#) [class method](#)), 421  
[is\\_defined\\_for\\_problem\\_type\(\)](#) ([evalml.objectives.AUC](#) [class method](#)), 424  
[is\\_defined\\_for\\_problem\\_type\(\)](#) ([evalml.objectives.AUCMacro](#) [class method](#)), 426  
[is\\_defined\\_for\\_problem\\_type\(\)](#) ([evalml.objectives.AUCMicro](#) [class method](#)), 429  
[is\\_defined\\_for\\_problem\\_type\(\)](#) ([evalml.objectives.AUCWeighted](#) [class method](#)), 431  
[is\\_defined\\_for\\_problem\\_type\(\)](#) ([evalml.objectives.BalancedAccuracyBinary](#) [class method](#)), 434  
[is\\_defined\\_for\\_problem\\_type\(\)](#) ([evalml.objectives.BalancedAccuracyMulticlass](#) [class method](#)), 437  
[is\\_defined\\_for\\_problem\\_type\(\)](#) ([evalml.objectives.BinaryClassificationObjective](#) [class method](#)), 399  
[is\\_defined\\_for\\_problem\\_type\(\)](#) ([evalml.objectives.CostBenefitMatrix](#) [class method](#)), 414  
[is\\_defined\\_for\\_problem\\_type\(\)](#) ([evalml.objectives.ExpVariance](#) [class method](#)), 440  
[is\\_defined\\_for\\_problem\\_type\(\)](#) ([evalml.objectives.F1](#) [class method](#)), 440  
[is\\_defined\\_for\\_problem\\_type\(\)](#) ([evalml.objectives.F1Macro](#) [class method](#)), 445  
[is\\_defined\\_for\\_problem\\_type\(\)](#) ([evalml.objectives.F1Micro](#) [class method](#)), 445

<a href="#">442</a>		<a href="#">(evalml.objectives.PrecisionWeighted class method), 469</a>	
<code>is_defined_for_problem_type()</code>	<a href="#">(evalml.objectives.F1Weighted class method), 447</a>	<code>is_defined_for_problem_type()</code>	<a href="#">(evalml.objectives.R2 class method), 482</a>
<code>is_defined_for_problem_type()</code>	<a href="#">(evalml.objectives.FraudCost class method), 408</a>	<code>is_defined_for_problem_type()</code>	<a href="#">(evalml.objectives.Recall class method), 472</a>
<code>is_defined_for_problem_type()</code>	<a href="#">(evalml.objectives.LeadScoring class method), 411</a>	<code>is_defined_for_problem_type()</code>	<a href="#">(evalml.objectives.RecallMacro class method), 477</a>
<code>is_defined_for_problem_type()</code>	<a href="#">(evalml.objectives.LogLossBinary class method), 450</a>	<code>is_defined_for_problem_type()</code>	<a href="#">(evalml.objectives.RecallMicro class method), 474</a>
<code>is_defined_for_problem_type()</code>	<a href="#">(evalml.objectives.LogLossMulticlass class method), 453</a>	<code>is_defined_for_problem_type()</code>	<a href="#">(evalml.objectives.RecallWeighted class method), 479</a>
<code>is_defined_for_problem_type()</code>	<a href="#">(evalml.objectives.MAE class method), 485</a>	<code>is_defined_for_problem_type()</code>	<a href="#">(evalml.objectives.RegressionObjective class method), 404</a>
<code>is_defined_for_problem_type()</code>	<a href="#">(evalml.objectives.MAPE class method), 487</a>	<code>is_defined_for_problem_type()</code>	<a href="#">(evalml.objectives.RootMeanSquaredError class method), 502</a>
<code>is_defined_for_problem_type()</code>	<a href="#">(evalml.objectives.MaxError class method), 497</a>	<code>is_defined_for_problem_type()</code>	<a href="#">(evalml.objectives.RootMeanSquaredLogError class method), 505</a>
<code>is_defined_for_problem_type()</code>	<a href="#">(evalml.objectives.MCCBinary class method), 456</a>	<code>is_search_space_exhausted()</code>	<a href="#">(evalml.tuners.GridSearchTuner method), 514</a>
<code>is_defined_for_problem_type()</code>	<a href="#">(evalml.objectives.MCCMulticlass class method), 458</a>	<code>is_search_space_exhausted()</code>	<a href="#">(evalml.tuners.RandomSearchTuner method), 516</a>
<code>is_defined_for_problem_type()</code>	<a href="#">(evalml.objectives.MeanSquaredLogError class method), 492</a>	<code>is_search_space_exhausted()</code>	<a href="#">(evalml.tuners.SKOptTuner method), 512</a>
<code>is_defined_for_problem_type()</code>	<a href="#">(evalml.objectives.MedianAE class method), 495</a>	<code>is_search_space_exhausted()</code>	<a href="#">(evalml.tuners.Tuner method), 511</a>
<code>is_defined_for_problem_type()</code>	<a href="#">(evalml.objectives.MSE class method), 490</a>	<code>IterativeAlgorithm</code>	<a href="#">(class in evalml.automl.automl_algorithm), 156</a>
<code>is_defined_for_problem_type()</code>	<a href="#">(evalml.objectives.MulticlassClassificationObjective class method), 402</a>	<b>K</b>	
<code>is_defined_for_problem_type()</code>	<a href="#">(evalml.objectives.ObjectiveBase class method), 397</a>	<code>KMeansSMOTECVSplit</code>	<a href="#">(class in evalml.preprocessing), 128</a>
<code>is_defined_for_problem_type()</code>	<a href="#">(evalml.objectives.Precision class method), 461</a>	<code>KMeansSMOTETVSplit</code>	<a href="#">(class in evalml.preprocessing), 130</a>
<code>is_defined_for_problem_type()</code>	<a href="#">(evalml.objectives.PrecisionMacro class method), 466</a>	<code>KNeighborsClassifier</code>	<a href="#">(class in evalml.pipelines.components), 336</a>
<code>is_defined_for_problem_type()</code>	<a href="#">(evalml.objectives.PrecisionMicro class method), 464</a>	<b>L</b>	
<code>is_defined_for_problem_type()</code>	<a href="#">(evalml.objectives.PrecisionWeighted class method), 469</a>	<code>LeadScoring</code>	<a href="#">(class in evalml.objectives), 409</a>
<code>is_defined_for_problem_type()</code>	<a href="#">(evalml.objectives.R2 class method), 482</a>	<code>LightGBMClassifier</code>	<a href="#">(class in evalml.pipelines.components), 317</a>
<code>is_defined_for_problem_type()</code>	<a href="#">(evalml.objectives.Recall class method), 472</a>	<code>LightGBMRegressor</code>	<a href="#">(class in evalml.pipelines.components), 377</a>
<code>is_defined_for_problem_type()</code>	<a href="#">(evalml.objectives.RecallMacro class method), 477</a>	<code>LinearRegressor</code>	<a href="#">(class in evalml.pipelines.components), 352</a>
<code>is_defined_for_problem_type()</code>	<a href="#">(evalml.objectives.RecallMicro class method), 474</a>		
<code>is_defined_for_problem_type()</code>	<a href="#">(evalml.objectives.RecallWeighted class method), 479</a>		
<code>is_defined_for_problem_type()</code>	<a href="#">(evalml.objectives.RegressionObjective class method), 404</a>		
<code>is_defined_for_problem_type()</code>	<a href="#">(evalml.objectives.RootMeanSquaredError class method), 502</a>		
<code>is_defined_for_problem_type()</code>	<a href="#">(evalml.objectives.RootMeanSquaredLogError class method), 505</a>		
<code>is_search_space_exhausted()</code>	<a href="#">(evalml.tuners.GridSearchTuner method), 514</a>		
<code>is_search_space_exhausted()</code>	<a href="#">(evalml.tuners.RandomSearchTuner method), 516</a>		
<code>is_search_space_exhausted()</code>	<a href="#">(evalml.tuners.SKOptTuner method), 512</a>		
<code>is_search_space_exhausted()</code>	<a href="#">(evalml.tuners.Tuner method), 511</a>		
<code>IterativeAlgorithm</code>	<a href="#">(class in evalml.automl.automl_algorithm), 156</a>		

<code>load()</code> ( <i>evalml.automl.AutoMLSearch</i> static method), 151	<code>load()</code> ( <i>evalml.pipelines.components.LightGBMRegressor</i> static method), 379
<code>load()</code> ( <i>evalml.pipelines.BaselineBinaryPipeline</i> static method), 207	<code>load()</code> ( <i>evalml.pipelines.components.LinearRegressor</i> static method), 354
<code>load()</code> ( <i>evalml.pipelines.BaselineMulticlassPipeline</i> static method), 212	<code>load()</code> ( <i>evalml.pipelines.components.LogisticRegressionClassifier</i> static method), 322
<code>load()</code> ( <i>evalml.pipelines.BaselineRegressionPipeline</i> static method), 228	<code>load()</code> ( <i>evalml.pipelines.components.OneHotEncoder</i> static method), 259
<code>load()</code> ( <i>evalml.pipelines.BinaryClassificationPipeline</i> static method), 172	<code>load()</code> ( <i>evalml.pipelines.components.PerColumnImputer</i> static method), 265
<code>load()</code> ( <i>evalml.pipelines.ClassificationPipeline</i> static method), 167	<code>load()</code> ( <i>evalml.pipelines.components.PolynomialDetrender</i> static method), 301
<code>load()</code> ( <i>evalml.pipelines.components.ARIMARegressor</i> static method), 345	<code>load()</code> ( <i>evalml.pipelines.components.RandomForestClassifier</i> static method), 316
<code>load()</code> ( <i>evalml.pipelines.components.BaselineClassifier</i> static method), 328	<code>load()</code> ( <i>evalml.pipelines.components.RandomForestRegressor</i> static method), 360
<code>load()</code> ( <i>evalml.pipelines.components.BaselineRegressor</i> static method), 366	<code>load()</code> ( <i>evalml.pipelines.components.RFClassifierSelectFromModel</i> static method), 281
<code>load()</code> ( <i>evalml.pipelines.components.CatBoostClassifier</i> static method), 307	<code>load()</code> ( <i>evalml.pipelines.components.RFRegressorSelectFromModel</i> static method), 278
<code>load()</code> ( <i>evalml.pipelines.components.CatBoostRegressor</i> static method), 348	<code>load()</code> ( <i>evalml.pipelines.components.SelectColumns</i> static method), 255
<code>load()</code> ( <i>evalml.pipelines.components.ComponentBase</i> static method), 242	<code>load()</code> ( <i>evalml.pipelines.components.SimpleImputer</i> static method), 272
<code>load()</code> ( <i>evalml.pipelines.components.DateTimeFeaturizer</i> static method), 288	<code>load()</code> ( <i>evalml.pipelines.components.StackedEnsembleClassifier</i> static method), 332
<code>load()</code> ( <i>evalml.pipelines.components.DecisionTreeClassifier</i> static method), 335	<code>load()</code> ( <i>evalml.pipelines.components.StackedEnsembleRegressor</i> static method), 373
<code>load()</code> ( <i>evalml.pipelines.components.DecisionTreeRegressor</i> static method), 376	<code>load()</code> ( <i>evalml.pipelines.components.StandardScaler</i> static method), 275
<code>load()</code> ( <i>evalml.pipelines.components.DelayedFeatureTransformer</i> static method), 294	<code>load()</code> ( <i>evalml.pipelines.components.SVMClassifier</i> static method), 341
<code>load()</code> ( <i>evalml.pipelines.components.DFSTransformer</i> static method), 297	<code>load()</code> ( <i>evalml.pipelines.components.SVMRegressor</i> static method), 382
<code>load()</code> ( <i>evalml.pipelines.components.DropColumns</i> static method), 252	<code>load()</code> ( <i>evalml.pipelines.components.TargetEncoder</i> static method), 262
<code>load()</code> ( <i>evalml.pipelines.components.DropNullColumns</i> static method), 285	<code>load()</code> ( <i>evalml.pipelines.components.TextFeaturizer</i> static method), 291
<code>load()</code> ( <i>evalml.pipelines.components.ElasticNetClassifier</i> static method), 310	<code>load()</code> ( <i>evalml.pipelines.components.TimeSeriesBaselineEstimator</i> static method), 369
<code>load()</code> ( <i>evalml.pipelines.components.ElasticNetRegressor</i> static method), 351	<code>load()</code> ( <i>evalml.pipelines.components.Transformer</i> static method), 244
<code>load()</code> ( <i>evalml.pipelines.components.Estimator</i> static method), 247	<code>load()</code> ( <i>evalml.pipelines.components.Undersampler</i> static method), 304
<code>load()</code> ( <i>evalml.pipelines.components.ExtraTreesClassifier</i> static method), 313	<code>load()</code> ( <i>evalml.pipelines.components.XGBoostClassifier</i> static method), 325
<code>load()</code> ( <i>evalml.pipelines.components.ExtraTreesRegressor</i> static method), 357	<code>load()</code> ( <i>evalml.pipelines.components.XGBoostRegressor</i> static method), 363
<code>load()</code> ( <i>evalml.pipelines.components.Imputer</i> static method), 269	<code>load()</code> ( <i>evalml.pipelines.MeanBaselineRegressionPipeline</i> static method), 233
<code>load()</code> ( <i>evalml.pipelines.components.KNeighborsClassifier</i> static method), 338	<code>load()</code> ( <i>evalml.pipelines.ModeBaselineBinaryPipeline</i> static method), 217
<code>load()</code> ( <i>evalml.pipelines.components.LightGBMClassifier</i> static method), 319	<code>load()</code> ( <i>evalml.pipelines.ModeBaselineMulticlassPipeline</i> static method), 222



`load()` (*evalml.pipelines.MulticlassClassificationPipeline* *MethodPropertyNotFoundError* (class in static method), 177 *evalml.exceptions*), 143  
`load()` (*evalml.pipelines.PipelineBase* static method), *MissingComponentError* (class in 163 *evalml.exceptions*), 145  
`load()` (*evalml.pipelines.RegressionPipeline* static *ModeBaselineBinaryPipeline* (class in method), 182 *evalml.pipelines*), 214  
`load()` (*evalml.pipelines.TimeSeriesBaselineRegressionPipeline* *ModelBaselineMulticlassPipeline* (class in static method), 238 *evalml.pipelines*), 219  
`load()` (*evalml.pipelines.TimeSeriesBinaryClassificationPipeline* *PipelineFamily* (*evalml.pipelines.BaselineBinaryPipeline* static method), 192 attribute), 203  
`load()` (*evalml.pipelines.TimeSeriesClassificationPipeline* *model\_family* (*evalml.pipelines.BaselineMulticlassPipeline* static method), 187 attribute), 209  
`load()` (*evalml.pipelines.TimeSeriesMulticlassClassificationPipeline* *model\_family* (*evalml.pipelines.BaselineRegressionPipeline* static method), 197 attribute), 224  
`load()` (*evalml.pipelines.TimeSeriesRegressionPipeline* *model\_family* (*evalml.pipelines.components.ARIMARegressor* static method), 202 attribute), 343  
`load_breast_cancer()` (in module *evalml.demos*), *model\_family* (*evalml.pipelines.components.BaselineClassifier* 122 attribute), 326  
`load_churn()` (in module *evalml.demos*), 122 *model\_family* (*evalml.pipelines.components.BaselineRegressor* attribute), 364  
`load_data()` (in module *evalml.preprocessing*), 123 *model\_family* (*evalml.pipelines.components.CatBoostClassifier* attribute), 305  
`load_diabetes()` (in module *evalml.demos*), 122 *model\_family* (*evalml.pipelines.components.CatBoostRegressor* attribute), 346  
`load_fraud()` (in module *evalml.demos*), 121 *model\_family* (*evalml.pipelines.components.DateTimeFeaturizer* attribute), 286  
`load_wine()` (in module *evalml.demos*), 121 *model\_family* (*evalml.pipelines.components.DecisionTreeClassifier* attribute), 333  
`log_error_callback()` (in module *evalml.automl.callbacks*), 159 *model\_family* (*evalml.pipelines.components.DecisionTreeRegressor* attribute), 374  
`LogisticRegressionClassifier` (class in *evalml.pipelines.components*), 320 *model\_family* (*evalml.pipelines.components.DelayedFeatureTransformer* attribute), 292  
`LogLossBinary` (class in *evalml.objectives*), 449 *model\_family* (*evalml.pipelines.components.DFSTransformer* attribute), 295  
`LogLossMulticlass` (class in *evalml.objectives*), 452 *model\_family* (*evalml.pipelines.components.DropColumns* attribute), 250  
*model\_family* (*evalml.pipelines.components.DropNullColumns* attribute), 283  
*model\_family* (*evalml.pipelines.components.ElasticNetClassifier* attribute), 308  
*model\_family* (*evalml.pipelines.components.ElasticNetRegressor* attribute), 349  
*model\_family* (*evalml.pipelines.components.ExtraTreesClassifier* attribute), 311  
*model\_family* (*evalml.pipelines.components.ExtraTreesRegressor* attribute), 355  
*model\_family* (*evalml.pipelines.components.Imputer* attribute), 266  
*model\_family* (*evalml.pipelines.components.KNeighborsClassifier* attribute), 336  
*model\_family* (*evalml.pipelines.components.LightGBMClassifier* attribute), 317  
*model\_family* (*evalml.pipelines.components.LightGBMRegressor* attribute), 377

## M

`MAE` (class in *evalml.objectives*), 484  
`make_data_splitter()` (in module *evalml.automl*), 154  
`make_pipeline()` (in module *evalml.pipelines.utils*), 239  
`make_pipeline_from_components()` (in module *evalml.pipelines.utils*), 240  
`MAPE` (class in *evalml.objectives*), 486  
`MaxError` (class in *evalml.objectives*), 496  
`MCCBinary` (class in *evalml.objectives*), 454  
`MCCMulticlass` (class in *evalml.objectives*), 457  
`MeanBaselineRegressionPipeline` (class in *evalml.pipelines*), 229  
`MeanSquaredLogError` (class in *evalml.objectives*), 491  
`MedianAE` (class in *evalml.objectives*), 494  
`message_type` (*evalml.data\_checks.DataCheckError* attribute), 538  
`message_type` (*evalml.data\_checks.DataCheckMessage* attribute), 537  
`message_type` (*evalml.data\_checks.DataCheckWarning* attribute), 539

`model_family` (`evalml.pipelines.components.LinearRegression` attribute), 352  
`model_family` (`evalml.pipelines.components.LogisticRegression` attribute), 320  
`model_family` (`evalml.pipelines.components.OneHotEncoder` attribute), 256  
`model_family` (`evalml.pipelines.components.PerColumnImputer` attribute), 263  
`model_family` (`evalml.pipelines.components.PolynomialDegree` attribute), 298  
`model_family` (`evalml.pipelines.components.RandomForestClassifier` attribute), 314  
`model_family` (`evalml.pipelines.components.RandomForestRegressor` attribute), 358  
`model_family` (`evalml.pipelines.components.RFClassifierSelectFromModel` attribute), 279  
`model_family` (`evalml.pipelines.components.RFRegressorSelectFromModel` attribute), 276  
`model_family` (`evalml.pipelines.components.SelectColumns` attribute), 253  
`model_family` (`evalml.pipelines.components.SimpleImputer` attribute), 270  
`model_family` (`evalml.pipelines.components.StackedEnsembleClassifier` attribute), 330  
`model_family` (`evalml.pipelines.components.StackedEnsembleRegressor` attribute), 371  
`model_family` (`evalml.pipelines.components.StandardScaler` attribute), 273  
`model_family` (`evalml.pipelines.components.SVMClassifier` attribute), 339  
`model_family` (`evalml.pipelines.components.SVMRegressor` attribute), 380  
`model_family` (`evalml.pipelines.components.TargetEncoder` attribute), 260  
`model_family` (`evalml.pipelines.components.TextFeaturizer` attribute), 289  
`model_family` (`evalml.pipelines.components.TimeSeriesBaselineEstimator` attribute), 367  
`model_family` (`evalml.pipelines.components.Undersampler` attribute), 302  
`model_family` (`evalml.pipelines.components.XGBoostClassifier` attribute), 323  
`model_family` (`evalml.pipelines.components.XGBoostRegressor` attribute), 361  
`model_family` (`evalml.pipelines.components.MeanBaselineRegressionPipeline` attribute), 229  
`model_family` (`evalml.pipelines.components.ModeBaselineBinaryPipeline` attribute), 214  
`model_family` (`evalml.pipelines.components.ModeBaselineMulticlassPipeline` attribute), 219  
`model_family` (`evalml.pipelines.components.TimeSeriesBaselineRegressionPipeline` attribute), 234  
`ModelFamily` (class in `evalml.model_family`), 509  
`MSE` (class in `evalml.objectives`), 489  
`MultiClassClassificationObjective` (class in `evalml.objectives`), 401  
`MultiClassClassificationPipeline` (class in `evalml.pipelines`), 174  
`MulticollinearityDataCheck` (class in `evalml.data_checks`), 530  
`Name`  
`ClassImbalanceDataCheck` (class in `evalml.data_checks`), 528  
`DataCheck` (class in `evalml.data_checks`), 517  
`DateTimeNaNDataCheck` (class in `evalml.data_checks`), 531  
`HighlyNullDataCheck` (class in `evalml.data_checks`), 520  
`IDColumnsDataCheck` (class in `evalml.data_checks`), 522  
`InvalidTargetDataCheck` (class in `evalml.data_checks`), 519  
`MulticollinearityDataCheck` (class in `evalml.data_checks`), 530  
`NaturalLanguageNaNDataCheck` (class in `evalml.data_checks`), 532  
`NoVarianceDataCheck` (class in `evalml.data_checks`), 527  
`OutliersDataCheck` (class in `evalml.data_checks`), 525  
`TargetLeakageDataCheck` (class in `evalml.data_checks`), 523  
`AccuracyBinary` (class in `evalml.objectives`), 417  
`AccuracyMulticlass` (class in `evalml.objectives`), 420  
`AUC` (class in `evalml.objectives`), 422  
`AUCMacro` (class in `evalml.objectives`), 425  
`AUCMicro` (class in `evalml.objectives`), 428  
`AUCWeighted` (class in `evalml.objectives`), 430  
`BalancedAccuracyBinary` (class in `evalml.objectives`), 433  
`BalancedAccuracyMulticlass` (class in `evalml.objectives`), 436  
`CostBenefitMatrix` (class in `evalml.objectives`), 413  
`ExpVariance` (class in `evalml.objectives`), 499  
`F1` (class in `evalml.objectives`), 438  
`F1Macro` (class in `evalml.objectives`), 444  
`F1Micro` (class in `evalml.objectives`), 441  
`F1Weighted` (class in `evalml.objectives`), 446  
`FraudCost` (class in `evalml.objectives`), 406  
`LeadScoring` (class in `evalml.objectives`), 409  
`LogLossBinary` (class in `evalml.objectives`), 449  
`LogLossMulticlass` (class in `evalml.objectives`), 452  
`MAE` (class in `evalml.objectives`), 484  
`MAPE` (class in `evalml.objectives`), 486

- name (*evalml.objectives.MaxError* attribute), 496
- name (*evalml.objectives.MCCBinary* attribute), 454
- name (*evalml.objectives.MCCMulticlass* attribute), 457
- name (*evalml.objectives.MeanSquaredLogError* attribute), 491
- name (*evalml.objectives.MedianAE* attribute), 494
- name (*evalml.objectives.MSE* attribute), 489
- name (*evalml.objectives.Precision* attribute), 460
- name (*evalml.objectives.PrecisionMacro* attribute), 465
- name (*evalml.objectives.PrecisionMicro* attribute), 463
- name (*evalml.objectives.PrecisionWeighted* attribute), 468
- name (*evalml.objectives.R2* attribute), 481
- name (*evalml.objectives.Recall* attribute), 470
- name (*evalml.objectives.RecallMacro* attribute), 476
- name (*evalml.objectives.RecallMicro* attribute), 473
- name (*evalml.objectives.RecallWeighted* attribute), 478
- name (*evalml.objectives.RootMeanSquaredError* attribute), 501
- name (*evalml.objectives.RootMeanSquaredLogError* attribute), 504
- name (*evalml.pipelines.BaselineBinaryPipeline* attribute), 203
- name (*evalml.pipelines.BaselineMulticlassPipeline* attribute), 209
- name (*evalml.pipelines.BaselineRegressionPipeline* attribute), 224
- name (*evalml.pipelines.BinaryClassificationPipeline* attribute), 169
- name (*evalml.pipelines.ClassificationPipeline* attribute), 164
- name (*evalml.pipelines.components.ARIMARegressor* attribute), 343
- name (*evalml.pipelines.components.BaselineClassifier* attribute), 326
- name (*evalml.pipelines.components.BaselineRegressor* attribute), 364
- name (*evalml.pipelines.components.CatBoostClassifier* attribute), 305
- name (*evalml.pipelines.components.CatBoostRegressor* attribute), 346
- name (*evalml.pipelines.components.DateTimeFeaturizer* attribute), 286
- name (*evalml.pipelines.components.DecisionTreeClassifier* attribute), 333
- name (*evalml.pipelines.components.DecisionTreeRegressor* attribute), 374
- name (*evalml.pipelines.components.DelayedFeatureTransformer* attribute), 292
- name (*evalml.pipelines.components.DFSTransformer* attribute), 295
- name (*evalml.pipelines.components.DropColumns* attribute), 250
- name (*evalml.pipelines.components.DropNullColumns* attribute), 283
- name (*evalml.pipelines.components.ElasticNetClassifier* attribute), 308
- name (*evalml.pipelines.components.ElasticNetRegressor* attribute), 349
- name (*evalml.pipelines.components.ExtraTreesClassifier* attribute), 311
- name (*evalml.pipelines.components.ExtraTreesRegressor* attribute), 355
- name (*evalml.pipelines.components.Imputer* attribute), 266
- name (*evalml.pipelines.components.KNeighborsClassifier* attribute), 336
- name (*evalml.pipelines.components.LightGBMClassifier* attribute), 317
- name (*evalml.pipelines.components.LightGBMRegressor* attribute), 377
- name (*evalml.pipelines.components.LinearRegressor* attribute), 352
- name (*evalml.pipelines.components.LogisticRegressionClassifier* attribute), 320
- name (*evalml.pipelines.components.OneHotEncoder* attribute), 256
- name (*evalml.pipelines.components.PerColumnImputer* attribute), 263
- name (*evalml.pipelines.components.PolynomialDetrender* attribute), 298
- name (*evalml.pipelines.components.RandomForestClassifier* attribute), 314
- name (*evalml.pipelines.components.RandomForestRegressor* attribute), 358
- name (*evalml.pipelines.components.RFClassifierSelectFromModel* attribute), 279
- name (*evalml.pipelines.components.RFRegressorSelectFromModel* attribute), 276
- name (*evalml.pipelines.components.SelectColumns* attribute), 253
- name (*evalml.pipelines.components.SimpleImputer* attribute), 270
- name (*evalml.pipelines.components.StackedEnsembleClassifier* attribute), 330
- name (*evalml.pipelines.components.StackedEnsembleRegressor* attribute), 371
- name (*evalml.pipelines.components.StandardScaler* attribute), 273
- name (*evalml.pipelines.components.SVMClassifier* attribute), 339
- name (*evalml.pipelines.components.SVMRegressor* attribute), 380
- name (*evalml.pipelines.components.TargetEncoder* attribute), 260
- name (*evalml.pipelines.components.TextFeaturizer* attribute), 289
- name (*evalml.pipelines.components.TimeSeriesBaselineEstimator* attribute), 283



[attribute](#)), 367  
 name ([evalml.pipelines.components.Undersampler attribute](#)), 302  
 name ([evalml.pipelines.components.XGBoostClassifier attribute](#)), 323  
 name ([evalml.pipelines.components.XGBoostRegressor attribute](#)), 361  
 name ([evalml.pipelines.MeanBaselineRegressionPipeline attribute](#)), 229  
 name ([evalml.pipelines.ModeBaselineBinaryPipeline attribute](#)), 214  
 name ([evalml.pipelines.ModeBaselineMulticlassPipeline attribute](#)), 219  
 name ([evalml.pipelines.MulticlassClassificationPipeline attribute](#)), 174  
 name ([evalml.pipelines.PipelineBase attribute](#)), 159  
 name ([evalml.pipelines.RegressionPipeline attribute](#)), 179  
 name ([evalml.pipelines.TimeSeriesBaselineRegressionPipeline attribute](#)), 234  
 name ([evalml.pipelines.TimeSeriesBinaryClassificationPipeline attribute](#)), 188  
 name ([evalml.pipelines.TimeSeriesClassificationPipeline attribute](#)), 183  
 name ([evalml.pipelines.TimeSeriesMulticlassClassificationPipeline attribute](#)), 194  
 name ([evalml.pipelines.TimeSeriesRegressionPipeline attribute](#)), 199  
 NaturalLanguageNaNDataCheck (class in [evalml.data\\_checks](#)), 532  
 next\_batch() ([evalml.automl.automl\\_algorithm.AutoMLAlgorithm](#) method), 156  
 next\_batch() ([evalml.automl.automl\\_algorithm.IterativeAlgorithm](#) method), 158  
 normalize\_confusion\_matrix() (in module [evalml.model\\_understanding](#)), 384  
 NoVarianceDataCheck (class in [evalml.data\\_checks](#)), 527  
 NullsInColumnWarning (class in [evalml.exceptions](#)), 147  
 number\_of\_features() (in module [evalml.preprocessing](#)), 123  
 O  
 objective\_function() ([evalml.objectives.AccuracyBinary](#) method), 419  
 objective\_function() ([evalml.objectives.AccuracyMulticlass](#) method), 421  
 objective\_function() ([evalml.objectives.AUC](#) method), 424  
 objective\_function() ([evalml.objectives.AUCMacro](#) method), 427  
 objective\_function() ([evalml.objectives.AUCMicro](#) method), 429  
 objective\_function() ([evalml.objectives.AUCWeighted](#) method), 432  
 objective\_function() ([evalml.objectives.BalancedAccuracyBinary](#) method), 435  
 objective\_function() ([evalml.objectives.BalancedAccuracyMulticlass](#) method), 437  
 objective\_function() ([evalml.objectives.BinaryClassificationObjective](#) class method), 399  
 objective\_function() ([evalml.objectives.CostBenefitMatrix](#) method), 414  
 objective\_function() ([evalml.objectives.ExpVariance](#) method), 500  
 objective\_function() ([evalml.objectives.F1](#) method), 440  
 objective\_function() ([evalml.objectives.F1Macro](#) method), 445  
 objective\_function() ([evalml.objectives.F1Micro](#) method), 443  
 objective\_function() ([evalml.objectives.F1Weighted](#) method), 448  
 objective\_function() ([evalml.objectives.FraudCost](#) method), 408  
 objective\_function() ([evalml.objectives.LeadScoring](#) method), 411  
 objective\_function() ([evalml.objectives.LogLossBinary](#) method), 451  
 objective\_function() ([evalml.objectives.LogLossMulticlass](#) method), 453  
 objective\_function() ([evalml.objectives.MAE](#) method), 485  
 objective\_function() ([evalml.objectives.MAPE](#) method), 488  
 objective\_function() ([evalml.objectives.MaxError](#) method), 498  
 objective\_function() ([evalml.objectives.MCCBinary](#) method), 456  
 objective\_function() ([evalml.objectives.MCCMulticlass](#) method), 459  
 objective\_function() ([evalml.objectives.MeanSquaredLogError](#)

[method](#)), 493  
[objective\\_function\(\)](#) ([evalml.objectives.MedianAE method](#)), 495  
[objective\\_function\(\)](#) ([evalml.objectives.MSE method](#)), 490  
[objective\\_function\(\)](#) ([evalml.objectives.MulticlassClassificationObjective class method](#)), 402  
[objective\\_function\(\)](#) ([evalml.objectives.ObjectiveBase class method](#)), 397  
[objective\\_function\(\)](#) ([evalml.objectives.Precision method](#)), 462  
[objective\\_function\(\)](#) ([evalml.objectives.PrecisionMacro method](#)), 467  
[objective\\_function\(\)](#) ([evalml.objectives.PrecisionMicro method](#)), 464  
[objective\\_function\(\)](#) ([evalml.objectives.PrecisionWeighted method](#)), 469  
[objective\\_function\(\)](#) ([evalml.objectives.R2 method](#)), 483  
[objective\\_function\(\)](#) ([evalml.objectives.Recall method](#)), 472  
[objective\\_function\(\)](#) ([evalml.objectives.RecallMacro method](#)), 477  
[objective\\_function\(\)](#) ([evalml.objectives.RecallMicro method](#)), 475  
[objective\\_function\(\)](#) ([evalml.objectives.RecallWeighted method](#)), 480  
[objective\\_function\(\)](#) ([evalml.objectives.RegressionObjective class method](#)), 404  
[objective\\_function\(\)](#) ([evalml.objectives.RootMeanSquaredError method](#)), 503  
[objective\\_function\(\)](#) ([evalml.objectives.RootMeanSquaredLogError method](#)), 505  
[ObjectiveBase](#) ([class in evalml.objectives](#)), 396  
[ObjectiveNotFoundError](#) ([class in evalml.exceptions](#)), 144  
[OneHotEncoder](#) ([class in evalml.pipelines.components](#)), 256  
[optimize\\_threshold\(\)](#) ([evalml.objectives.AccuracyBinary method](#)), 419  
[optimize\\_threshold\(\)](#) ([evalml.objectives.AUC method](#)), 424  
[optimize\\_threshold\(\)](#) ([evalml.objectives.BalancedAccuracyBinary method](#)), 435  
[optimize\\_threshold\(\)](#) ([evalml.objectives.BinaryClassificationObjective method](#)), 400  
[optimize\\_threshold\(\)](#) ([evalml.objectives.CostBenefitMatrix method](#)), 415  
[optimize\\_threshold\(\)](#) ([evalml.objectives.F1 method](#)), 440  
[optimize\\_threshold\(\)](#) ([evalml.objectives.FraudCost method](#)), 408  
[optimize\\_threshold\(\)](#) ([evalml.objectives.LeadScoring method](#)), 411  
[optimize\\_threshold\(\)](#) ([evalml.objectives.LogLossBinary method](#)), 451  
[optimize\\_threshold\(\)](#) ([evalml.objectives.MCCBinary method](#)), 456  
[optimize\\_threshold\(\)](#) ([evalml.objectives.Precision method](#)), 462  
[optimize\\_threshold\(\)](#) ([evalml.objectives.Recall method](#)), 472  
[optimize\\_threshold\(\)](#) ([evalml.pipelines.BaselineBinaryPipeline method](#)), 207  
[optimize\\_threshold\(\)](#) ([evalml.pipelines.BinaryClassificationPipeline method](#)), 172  
[optimize\\_threshold\(\)](#) ([evalml.pipelines.ModeBaselineBinaryPipeline method](#)), 217  
[optimize\\_threshold\(\)](#) ([evalml.pipelines.TimeSeriesBinaryClassificationPipeline method](#)), 192  
[OutliersDataCheck](#) ([class in evalml.data\\_checks](#)), 525

## P

[pad\\_with\\_nans\(\)](#) ([in module evalml.utils](#)), 544  
[partial\\_dependence\(\)](#) ([in module evalml.model\\_understanding](#)), 387  
[PerColumnImputer](#) ([class in evalml.pipelines.components](#)), 263  
[perfect\\_score](#) ([evalml.objectives.AccuracyBinary attribute](#)), 417  
[perfect\\_score](#) ([evalml.objectives.AccuracyMulticlass attribute](#)), 420  
[perfect\\_score](#) ([evalml.objectives.AUC attribute](#)), 422

[perfect\\_score \(evalml.objectives.AUCMacro attribute\), 425](#)  
[perfect\\_score \(evalml.objectives.AUCMicro attribute\), 428](#)  
[perfect\\_score \(evalml.objectives.AUCWeighted attribute\), 430](#)  
[perfect\\_score \(evalml.objectives.BalancedAccuracyBinary attribute\), 433](#)  
[perfect\\_score \(evalml.objectives.BalancedAccuracyMulticlass attribute\), 436](#)  
[perfect\\_score \(evalml.objectives.CostBenefitMatrix attribute\), 413](#)  
[perfect\\_score \(evalml.objectives.ExpVariance attribute\), 499](#)  
[perfect\\_score \(evalml.objectives.F1 attribute\), 438](#)  
[perfect\\_score \(evalml.objectives.F1Macro attribute\), 444](#)  
[perfect\\_score \(evalml.objectives.F1Micro attribute\), 441](#)  
[perfect\\_score \(evalml.objectives.F1Weighted attribute\), 446](#)  
[perfect\\_score \(evalml.objectives.FraudCost attribute\), 406](#)  
[perfect\\_score \(evalml.objectives.LeadScoring attribute\), 409](#)  
[perfect\\_score \(evalml.objectives.LogLossBinary attribute\), 449](#)  
[perfect\\_score \(evalml.objectives.LogLossMulticlass attribute\), 452](#)  
[perfect\\_score \(evalml.objectives.MAE attribute\), 484](#)  
[perfect\\_score \(evalml.objectives.MAPE attribute\), 486](#)  
[perfect\\_score \(evalml.objectives.MaxError attribute\), 496](#)  
[perfect\\_score \(evalml.objectives.MCCBinary attribute\), 454](#)  
[perfect\\_score \(evalml.objectives.MCCMulticlass attribute\), 457](#)  
[perfect\\_score \(evalml.objectives.MeanSquaredLogError attribute\), 491](#)  
[perfect\\_score \(evalml.objectives.MedianAE attribute\), 494](#)  
[perfect\\_score \(evalml.objectives.MSE attribute\), 489](#)  
[perfect\\_score \(evalml.objectives.Precision attribute\), 460](#)  
[perfect\\_score \(evalml.objectives.PrecisionMacro attribute\), 465](#)  
[perfect\\_score \(evalml.objectives.PrecisionMicro attribute\), 463](#)  
[perfect\\_score \(evalml.objectives.PrecisionWeighted attribute\), 468](#)  
[perfect\\_score \(evalml.objectives.R2 attribute\), 481](#)  
[perfect\\_score \(evalml.objectives.Recall attribute\), 470](#)  
[perfect\\_score \(evalml.objectives.RecallMacro attribute\), 476](#)  
[perfect\\_score \(evalml.objectives.RecallMicro attribute\), 473](#)  
[perfect\\_score \(evalml.objectives.RecallWeighted attribute\), 478](#)  
[perfect\\_score \(evalml.objectives.RootMeanSquaredError attribute\), 501](#)  
[perfect\\_score \(evalml.objectives.RootMeanSquaredLogError attribute\), 504](#)  
[PipelineBase \(class in evalml.pipelines\), 159](#)  
[PipelineNotFoundError \(class in evalml.exceptions\), 144](#)  
[PipelineNotYetFittedError \(class in evalml.exceptions\), 145](#)  
[PipelineScoreError \(class in evalml.exceptions\), 146](#)  
[PolynomialDetrender \(class in evalml.pipelines.components\), 298](#)  
[positive\\_only \(evalml.objectives.AccuracyBinary attribute\), 417](#)  
[positive\\_only \(evalml.objectives.AccuracyMulticlass attribute\), 420](#)  
[positive\\_only \(evalml.objectives.AUC attribute\), 422](#)  
[positive\\_only \(evalml.objectives.AUCMacro attribute\), 425](#)  
[positive\\_only \(evalml.objectives.AUCMicro attribute\), 428](#)  
[positive\\_only \(evalml.objectives.AUCWeighted attribute\), 430](#)  
[positive\\_only \(evalml.objectives.BalancedAccuracyBinary attribute\), 433](#)  
[positive\\_only \(evalml.objectives.BalancedAccuracyMulticlass attribute\), 436](#)  
[positive\\_only \(evalml.objectives.CostBenefitMatrix attribute\), 413](#)  
[positive\\_only \(evalml.objectives.ExpVariance attribute\), 499](#)  
[positive\\_only \(evalml.objectives.F1 attribute\), 438](#)  
[positive\\_only \(evalml.objectives.F1Macro attribute\), 444](#)  
[positive\\_only \(evalml.objectives.F1Micro attribute\), 441](#)  
[positive\\_only \(evalml.objectives.F1Weighted attribute\), 446](#)  
[positive\\_only \(evalml.objectives.FraudCost attribute\), 406](#)  
[positive\\_only \(evalml.objectives.LeadScoring attribute\), 409](#)  
[positive\\_only \(evalml.objectives.LogLossBinary attribute\), 449](#)

- `positive_only` (*evalml.objectives.LogLossMulticlass attribute*), 452
- `positive_only` (*evalml.objectives.MAE attribute*), 484
- `positive_only` (*evalml.objectives.MAPE attribute*), 486
- `positive_only` (*evalml.objectives.MaxError attribute*), 496
- `positive_only` (*evalml.objectives.MCCBinary attribute*), 454
- `positive_only` (*evalml.objectives.MCCMulticlass attribute*), 457
- `positive_only` (*evalml.objectives.MeanSquaredLogError attribute*), 491
- `positive_only` (*evalml.objectives.MedianAE attribute*), 494
- `positive_only` (*evalml.objectives.MSE attribute*), 489
- `positive_only` (*evalml.objectives.Precision attribute*), 460
- `positive_only` (*evalml.objectives.PrecisionMacro attribute*), 465
- `positive_only` (*evalml.objectives.PrecisionMicro attribute*), 463
- `positive_only` (*evalml.objectives.PrecisionWeighted attribute*), 468
- `positive_only` (*evalml.objectives.R2 attribute*), 481
- `positive_only` (*evalml.objectives.Recall attribute*), 470
- `positive_only` (*evalml.objectives.RecallMacro attribute*), 476
- `positive_only` (*evalml.objectives.RecallMicro attribute*), 473
- `positive_only` (*evalml.objectives.RecallWeighted attribute*), 478
- `positive_only` (*evalml.objectives.RootMeanSquaredError attribute*), 501
- `positive_only` (*evalml.objectives.RootMeanSquaredLogError attribute*), 504
- `Precision` (class in *evalml.objectives*), 460
- `precision_recall_curve()` (in module *evalml.model\_understanding*), 385
- `PrecisionMacro` (class in *evalml.objectives*), 465
- `PrecisionMicro` (class in *evalml.objectives*), 463
- `PrecisionWeighted` (class in *evalml.objectives*), 468
- `predict()` (*evalml.pipelines.BaselineBinaryPipeline method*), 207
- `predict()` (*evalml.pipelines.BaselineMulticlassPipeline method*), 212
- `predict()` (*evalml.pipelines.BaselineRegressionPipeline method*), 228
- `predict()` (*evalml.pipelines.BinaryClassificationPipeline method*), 172
- `predict()` (*evalml.pipelines.ClassificationPipeline method*), 167
- `predict()` (*evalml.pipelines.components.ARIMAREgressor method*), 345
- `predict()` (*evalml.pipelines.components.BaselineClassifier method*), 329
- `predict()` (*evalml.pipelines.components.BaselineRegressor method*), 366
- `predict()` (*evalml.pipelines.components.CatBoostClassifier method*), 307
- `predict()` (*evalml.pipelines.components.CatBoostRegressor method*), 348
- `predict()` (*evalml.pipelines.components.DecisionTreeClassifier method*), 335
- `predict()` (*evalml.pipelines.components.DecisionTreeRegressor method*), 376
- `predict()` (*evalml.pipelines.components.ElasticNetClassifier method*), 310
- `predict()` (*evalml.pipelines.components.ElasticNetRegressor method*), 351
- `predict()` (*evalml.pipelines.components.Estimator method*), 247
- `predict()` (*evalml.pipelines.components.ExtraTreesClassifier method*), 313
- `predict()` (*evalml.pipelines.components.ExtraTreesRegressor method*), 357
- `predict()` (*evalml.pipelines.components.KNeighborsClassifier method*), 338
- `predict()` (*evalml.pipelines.components.LightGBMClassifier method*), 319
- `predict()` (*evalml.pipelines.components.LightGBMRegressor method*), 379
- `predict()` (*evalml.pipelines.components.LinearRegressor method*), 354
- `predict()` (*evalml.pipelines.components.LogisticRegressionClassifier method*), 322
- `predict()` (*evalml.pipelines.components.RandomForestClassifier method*), 316
- `predict()` (*evalml.pipelines.components.RandomForestRegressor method*), 360
- `predict()` (*evalml.pipelines.components.StackedEnsembleClassifier method*), 332
- `predict()` (*evalml.pipelines.components.StackedEnsembleRegressor method*), 373
- `predict()` (*evalml.pipelines.components.SVMClassifier method*), 341
- `predict()` (*evalml.pipelines.components.SVMRegressor method*), 382
- `predict()` (*evalml.pipelines.components.TimeSeriesBaselineEstimator method*), 370
- `predict()` (*evalml.pipelines.components.XGBoostClassifier method*), 325
- `predict()` (*evalml.pipelines.components.XGBoostRegressor method*), 363

`predict()` (`evalml.pipelines.MeanBaselineRegressionPipeline` method), 233  
`predict_proba()` (`evalml.pipelines.components.KNeighborsClassifier` method), 338  
`predict()` (`evalml.pipelines.ModeBaselineBinaryPipeline` method), 218  
`predict_proba()` (`evalml.pipelines.components.LightGBMClassifier` method), 319  
`predict()` (`evalml.pipelines.ModeBaselineMulticlassPipeline` method), 222  
`predict_proba()` (`evalml.pipelines.components.LightGBMRegressor` method), 380  
`predict()` (`evalml.pipelines.MulticlassClassificationPipeline` method), 177  
`predict_proba()` (`evalml.pipelines.components.LinearRegressor` method), 354  
`predict()` (`evalml.pipelines.PipelineBase` method), 163  
`predict_proba()` (`evalml.pipelines.components.LogisticRegressionClassifier` method), 322  
`predict()` (`evalml.pipelines.RegressionPipeline` method), 182  
`predict_proba()` (`evalml.pipelines.components.RandomForestClassifier` method), 316  
`predict()` (`evalml.pipelines.TimeSeriesBaselineRegressionPipeline` method), 238  
`predict_proba()` (`evalml.pipelines.components.RandomForestRegressor` method), 360  
`predict()` (`evalml.pipelines.TimeSeriesBinaryClassificationPipeline` method), 192  
`predict_proba()` (`evalml.pipelines.components.StackedEnsembleClassifier` method), 332  
`predict()` (`evalml.pipelines.TimeSeriesClassificationPipeline` method), 187  
`predict_proba()` (`evalml.pipelines.components.StackedEnsembleRegressor` method), 373  
`predict()` (`evalml.pipelines.TimeSeriesMulticlassClassificationPipeline` method), 197  
`predict_proba()` (`evalml.pipelines.components.SVMClassifier` method), 341  
`predict()` (`evalml.pipelines.TimeSeriesRegressionPipeline` method), 202  
`predict_proba()` (`evalml.pipelines.components.SVMRegressor` method), 382  
`predict_proba()` (`evalml.pipelines.BaselineBinaryPipeline` method), 208  
`predict_proba()` (`evalml.pipelines.components.TimeSeriesBaselineEstimator` method), 370  
`predict_proba()` (`evalml.pipelines.BaselineMulticlassPipeline` method), 212  
`predict_proba()` (`evalml.pipelines.components.XGBoostClassifier` method), 325  
`predict_proba()` (`evalml.pipelines.BinaryClassificationPipeline` method), 173  
`predict_proba()` (`evalml.pipelines.components.XGBoostRegressor` method), 363  
`predict_proba()` (`evalml.pipelines.ClassificationPipeline` method), 167  
`predict_proba()` (`evalml.pipelines.ModeBaselineBinaryPipeline` method), 218  
`predict_proba()` (`evalml.pipelines.components.ARIMARegressor` method), 345  
`predict_proba()` (`evalml.pipelines.ModeBaselineMulticlassPipeline` method), 223  
`predict_proba()` (`evalml.pipelines.components.BaselineClassifier` method), 329  
`predict_proba()` (`evalml.pipelines.MulticlassClassificationPipeline` method), 177  
`predict_proba()` (`evalml.pipelines.components.BaselineRegressor` method), 366  
`predict_proba()` (`evalml.pipelines.TimeSeriesBinaryClassificationPipeline` method), 193  
`predict_proba()` (`evalml.pipelines.components.CatBoostClassifier` method), 307  
`predict_proba()` (`evalml.pipelines.TimeSeriesClassificationPipeline` method), 187  
`predict_proba()` (`evalml.pipelines.components.CatBoostRegressor` method), 348  
`predict_proba()` (`evalml.pipelines.TimeSeriesMulticlassClassificationPipeline` method), 197  
`predict_proba()` (`evalml.pipelines.components.DecisionTreeClassifier` method), 335  
`predict_proba()` (`evalml.pipelines.components.ARIMARegressor` attribute), 343  
`predict_proba()` (`evalml.pipelines.components.DecisionTreeRegressor` method), 376  
`predict_proba()` (`evalml.pipelines.components.BaselineClassifier` attribute), 326  
`predict_proba()` (`evalml.pipelines.components.ElasticNetClassifier` method), 310  
`predict_proba()` (`evalml.pipelines.components.BaselineRegressor` attribute), 364  
`predict_proba()` (`evalml.pipelines.components.ElasticNetRegressor` method), 351  
`predict_proba()` (`evalml.pipelines.components.CatBoostClassifier` attribute), 305  
`predict_proba()` (`evalml.pipelines.components.Estimator` method), 247  
`predict_proba()` (`evalml.pipelines.components.CatBoostRegressor` attribute), 346  
`predict_proba()` (`evalml.pipelines.components.ExtraTreesClassifier` method), 313  
`predict_proba()` (`evalml.pipelines.components.DecisionTreeClassifier` attribute), 333  
`predict_proba()` (`evalml.pipelines.components.ExtraTreesRegressor` method), 357  
`predict_proba()` (`evalml.pipelines.components.DecisionTreeRegressor` attribute), 374



`predict Uses y (evalml.pipelines.components.ElasticNetClassifier type (evalml.pipelines.PipelineBase attribute), 308`  
`predict Uses y (evalml.pipelines.components.ElasticNetRegressor type (evalml.pipelines.RegressionPipeline attribute), 349`  
`predict Uses y (evalml.pipelines.components.ExtraTreesClassifier type (evalml.pipelines.TimeSeriesBaselineRegressionPipeline attribute), 311`  
`predict Uses y (evalml.pipelines.components.ExtraTreesRegressor type (evalml.pipelines.TimeSeriesBinaryClassificationPipeline attribute), 355`  
`predict Uses y (evalml.pipelines.components.KNeighborsClassifier type (evalml.pipelines.TimeSeriesClassificationPipeline attribute), 336`  
`predict Uses y (evalml.pipelines.components.LightGBMClassifier type (evalml.pipelines.TimeSeriesMulticlassClassificationPipeline attribute), 317`  
`predict Uses y (evalml.pipelines.components.LightGBMRegressor type (evalml.pipelines.TimeSeriesRegressionPipeline attribute), 377`  
`predict Uses y (evalml.pipelines.components.LinearRegression type (evalml.objectives.AccuracyBinary attribute), 352`  
`predict Uses y (evalml.pipelines.components.LogisticRegressionClassifier type (evalml.objectives.AccuracyMulticlass attribute), 320`  
`predict Uses y (evalml.pipelines.components.RandomForestClassifier type (evalml.objectives.AUC attribute), 314`  
`predict Uses y (evalml.pipelines.components.RandomForestRegressor type (evalml.objectives.AUCMacro attribute), 358`  
`predict Uses y (evalml.pipelines.components.StackedEnsembleClassifier type (evalml.objectives.AUCMicro attribute), 330`  
`predict Uses y (evalml.pipelines.components.StackedEnsembleRegressor type (evalml.objectives.AUCWeighted attribute), 371`  
`predict Uses y (evalml.pipelines.components.SVMClassifier type (evalml.objectives.BalancedAccuracyBinary attribute), 339`  
`predict Uses y (evalml.pipelines.components.SVMRegressor type (evalml.objectives.BalancedAccuracyMulticlass attribute), 380`  
`predict Uses y (evalml.pipelines.components.TimeSeriesBaselineEstimator type (evalml.objectives.BinaryClassificationObjective attribute), 367`  
`predict Uses y (evalml.pipelines.components.XGBoostClassifier type (evalml.objectives.CostBenefitMatrix attribute), 323`  
`predict Uses y (evalml.pipelines.components.XGBoostRegressor type (evalml.objectives.ExpVariance attribute), 361`  
`problem Type (evalml.pipelines.BaselineBinaryPipeline problem_types (evalml.objectives.F1 attribute), 203`  
`problem Type (evalml.pipelines.BaselineMulticlassPipeline problem_types (evalml.objectives.F1Macro attribute), 209`  
`problem Type (evalml.pipelines.BaselineRegressionPipeline problem_types (evalml.objectives.F1Micro attribute), 224`  
`problem Type (evalml.pipelines.BinaryClassificationPipeline problem_types (evalml.objectives.F1Weighted attribute), 169`  
`problem Type (evalml.pipelines.ClassificationPipeline problem_types (evalml.objectives.FraudCost attribute), 164`  
`problem Type (evalml.pipelines.MeanBaselineRegressionPipeline problem_types (evalml.objectives.LeadScoring attribute), 229`  
`problem Type (evalml.pipelines.ModeBaselineBinaryPipeline problem_types (evalml.objectives.LogLossBinary attribute), 214`  
`problem Type (evalml.pipelines.ModeBaselineMulticlassPipeline problem_types (evalml.objectives.LogLossMulticlass attribute), 219`  
`problem Type (evalml.pipelines.MulticlassClassificationPipeline problem_types (evalml.objectives.MAE attribute), 484`  
`problem Type (evalml.pipelines.MulticlassClassificationPipeline problem_types (evalml.objectives.MAPE attribute), 174`

- 486
- `problem_types` (*evalml.objectives.MaxError* attribute), 496
- `problem_types` (*evalml.objectives.MCCBinary* attribute), 454
- `problem_types` (*evalml.objectives.MCCMulticlass* attribute), 457
- `problem_types` (*evalml.objectives.MeanSquaredLogError* attribute), 491
- `problem_types` (*evalml.objectives.MedianAE* attribute), 494
- `problem_types` (*evalml.objectives.MSE* attribute), 489
- `problem_types` (*evalml.objectives.MulticlassClassification* attribute), 401
- `problem_types` (*evalml.objectives.ObjectiveBase* attribute), 396
- `problem_types` (*evalml.objectives.Precision* attribute), 460
- `problem_types` (*evalml.objectives.PrecisionMacro* attribute), 465
- `problem_types` (*evalml.objectives.PrecisionMicro* attribute), 463
- `problem_types` (*evalml.objectives.PrecisionWeighted* attribute), 468
- `problem_types` (*evalml.objectives.R2* attribute), 481
- `problem_types` (*evalml.objectives.Recall* attribute), 470
- `problem_types` (*evalml.objectives.RecallMacro* attribute), 476
- `problem_types` (*evalml.objectives.RecallMicro* attribute), 473
- `problem_types` (*evalml.objectives.RecallWeighted* attribute), 478
- `problem_types` (*evalml.objectives.RegistrationObjective* attribute), 403
- `problem_types` (*evalml.objectives.RootMeanSquaredError* attribute), 501
- `problem_types` (*evalml.objectives.RootMeanSquaredLogError* attribute), 504
- `ProblemTypes` (class in *evalml.problem\_types*), 508
- `propose()` (*evalml.tuners.GridSearchTuner* method), 514
- `propose()` (*evalml.tuners.RandomSearchTuner* method), 516
- `propose()` (*evalml.tuners.SKOptTuner* method), 512
- `propose()` (*evalml.tuners.Tuner* method), 511
- R**
- `R2` (class in *evalml.objectives*), 481
- `raise_error_callback()` (in module *evalml.automl.callbacks*), 159
- `RandomForestClassifier` (class in *evalml.pipelines.components*), 314
- `RandomForestRegressor` (class in *evalml.pipelines.components*), 358
- `RandomSearchTuner` (class in *evalml.tuners*), 515
- `RandomUnderSamplerCVSplit` (class in *evalml.preprocessing*), 136
- `RandomUnderSamplerTVSplit` (class in *evalml.preprocessing*), 137
- `Recall` (class in *evalml.objectives*), 470
- `RecallMacro` (class in *evalml.objectives*), 476
- `RecallMicro` (class in *evalml.objectives*), 473
- `RecallWeighted` (class in *evalml.objectives*), 478
- `RegressionObjective` (class in *evalml.objectives*), 403
- `RegistrationPipeline` (class in *evalml.pipelines*), 179
- `RFClassifierSelectFromModel` (class in *evalml.pipelines.components*), 279
- `RFRegressorSelectFromModel` (class in *evalml.pipelines.components*), 276
- `roc_curve()` (in module *evalml.model\_understanding*), 385
- `RootMeanSquaredError` (class in *evalml.objectives*), 501
- `RootMeanSquaredLogError` (class in *evalml.objectives*), 504
- S**
- `save()` (*evalml.automl.AutoMLSearch* method), 151
- `save()` (*evalml.pipelines.BaselineBinaryPipeline* method), 208
- `save()` (*evalml.pipelines.BaselineMulticlassPipeline* method), 213
- `save()` (*evalml.pipelines.BaselineRegressionPipeline* method), 228
- `save()` (*evalml.pipelines.BinaryClassificationPipeline* method), 173
- `save()` (*evalml.pipelines.ClassificationPipeline* method), 168
- `save()` (*evalml.pipelines.components.ARIMARegressor* method), 345
- `save()` (*evalml.pipelines.components.BaselineClassifier* method), 329
- `save()` (*evalml.pipelines.components.BaselineRegressor* method), 367
- `save()` (*evalml.pipelines.components.CatBoostClassifier* method), 307
- `save()` (*evalml.pipelines.components.CatBoostRegressor* method), 348
- `save()` (*evalml.pipelines.components.ComponentBase* method), 242
- `save()` (*evalml.pipelines.components.DateFeaturizer* method), 288
- `save()` (*evalml.pipelines.components.DecisionTreeClassifier* method), 336

<code>save()</code> ( <code>evalml.pipelines.components.DecisionTreeRegressor</code> method), 377	<code>save()</code> ( <code>evalml.pipelines.components.StandardScaler</code> method), 275
<code>save()</code> ( <code>evalml.pipelines.components.DelayedFeatureTransformer</code> method), 294	<code>save()</code> ( <code>evalml.pipelines.components.SVMClassifier</code> method), 342
<code>save()</code> ( <code>evalml.pipelines.components.DFSTransformer</code> method), 297	<code>save()</code> ( <code>evalml.pipelines.components.SVMRegressor</code> method), 383
<code>save()</code> ( <code>evalml.pipelines.components.DropColumns</code> method), 252	<code>save()</code> ( <code>evalml.pipelines.components.TargetEncoder</code> method), 262
<code>save()</code> ( <code>evalml.pipelines.components.DropNullColumns</code> method), 285	<code>save()</code> ( <code>evalml.pipelines.components.TextFeaturizer</code> method), 291
<code>save()</code> ( <code>evalml.pipelines.components.ElasticNetClassifier</code> method), 310	<code>save()</code> ( <code>evalml.pipelines.components.TimeSeriesBaselineEstimator</code> method), 370
<code>save()</code> ( <code>evalml.pipelines.components.ElasticNetRegressors</code> method), 351	<code>save()</code> ( <code>evalml.pipelines.components.Transformer</code> method), 244
<code>save()</code> ( <code>evalml.pipelines.components.Estimator</code> method), 247	<code>save()</code> ( <code>evalml.pipelines.components.Undersampler</code> method), 304
<code>save()</code> ( <code>evalml.pipelines.components.ExtraTreesClassifiers</code> method), 313	<code>save()</code> ( <code>evalml.pipelines.components.XGBoostClassifier</code> method), 326
<code>save()</code> ( <code>evalml.pipelines.components.ExtraTreesRegressors</code> method), 357	<code>save()</code> ( <code>evalml.pipelines.components.XGBoostRegressor</code> method), 363
<code>save()</code> ( <code>evalml.pipelines.components.Imputer</code> method), 269	<code>save()</code> ( <code>evalml.pipelines.MeanBaselineRegressionPipeline</code> method), 233
<code>save()</code> ( <code>evalml.pipelines.components.KNeighborsClassifier</code> method), 339	<code>save()</code> ( <code>evalml.pipelines.ModeBaselineBinaryPipeline</code> method), 218
<code>save()</code> ( <code>evalml.pipelines.components.LightGBMClassifiers</code> method), 320	<code>save()</code> ( <code>evalml.pipelines.ModeBaselineMulticlassPipeline</code> method), 223
<code>save()</code> ( <code>evalml.pipelines.components.LightGBMRegressors</code> method), 380	<code>save()</code> ( <code>evalml.pipelines.MulticlassClassificationPipeline</code> method), 178
<code>save()</code> ( <code>evalml.pipelines.components.LinearRegressor</code> method), 354	<code>save()</code> ( <code>evalml.pipelines.PipelineBase</code> method), 163
<code>save()</code> ( <code>evalml.pipelines.components.LogisticRegressionClassifier</code> method), 323	<code>save()</code> ( <code>evalml.pipelines.RegistrationPipeline</code> method), 182
<code>save()</code> ( <code>evalml.pipelines.components.OneHotEncoder</code> method), 259	<code>save()</code> ( <code>evalml.pipelines.TimeSeriesBaselineRegressionPipeline</code> method), 238
<code>save()</code> ( <code>evalml.pipelines.components.PerColumnImputer</code> method), 265	<code>save()</code> ( <code>evalml.pipelines.TimeSeriesBinaryClassificationPipeline</code> method), 193
<code>save()</code> ( <code>evalml.pipelines.components.PolynomialDetrender</code> method), 301	<code>save()</code> ( <code>evalml.pipelines.TimeSeriesClassificationPipeline</code> method), 187
<code>save()</code> ( <code>evalml.pipelines.components.RandomForestClassifier</code> method), 316	<code>save()</code> ( <code>evalml.pipelines.TimeSeriesMulticlassClassificationPipeline</code> method), 198
<code>save()</code> ( <code>evalml.pipelines.components.RandomForestRegressor</code> method), 360	<code>save()</code> ( <code>evalml.pipelines.TimeSeriesRegressionPipeline</code> method), 202
<code>save()</code> ( <code>evalml.pipelines.components.RFClassifierSelectFromModel</code> method), 282	<code>save_plot()</code> (in module <code>evalml.utils</code> ), 545
<code>save()</code> ( <code>evalml.pipelines.components.RFRegressorSelectFromModel</code> method), 278	<code>score()</code> ( <code>evalml.objectives.AccuracyBinary</code> method), 419
<code>save()</code> ( <code>evalml.pipelines.components.SelectColumns</code> method), 255	<code>score()</code> ( <code>evalml.objectives.AccuracyMulticlass</code> method), 421
<code>save()</code> ( <code>evalml.pipelines.components.SimpleImputer</code> method), 272	<code>score()</code> ( <code>evalml.objectives.AUC</code> method), 424
<code>save()</code> ( <code>evalml.pipelines.components.StackedEnsembleClassifier</code> method), 333	<code>score()</code> ( <code>evalml.objectives.AUCMacro</code> method), 427
<code>save()</code> ( <code>evalml.pipelines.components.StackedEnsembleRegressor</code> method), 374	<code>score()</code> ( <code>evalml.objectives.AUCMicro</code> method), 429
	<code>score()</code> ( <code>evalml.objectives.AUCWeighted</code> method), 432
	<code>score()</code> ( <code>evalml.objectives.BalancedAccuracyBinary</code> method), 435
	<code>score()</code> ( <code>evalml.objectives.BalancedAccuracyMulticlass</code> method), 437



`score()` (*evalml.objectives.BinaryClassificationObjective* method), 400  
`score()` (*evalml.objectives.CostBenefitMatrix* method), 415  
`score()` (*evalml.objectives.ExpVariance* method), 500  
`score()` (*evalml.objectives.F1* method), 440  
`score()` (*evalml.objectives.F1Macro* method), 445  
`score()` (*evalml.objectives.F1Micro* method), 443  
`score()` (*evalml.objectives.F1Weighted* method), 448  
`score()` (*evalml.objectives.FraudCost* method), 408  
`score()` (*evalml.objectives.LeadScoring* method), 412  
`score()` (*evalml.objectives.LogLossBinary* method), 451  
`score()` (*evalml.objectives.LogLossMulticlass* method), 453  
`score()` (*evalml.objectives.MAE* method), 485  
`score()` (*evalml.objectives.MAPE* method), 488  
`score()` (*evalml.objectives.MaxError* method), 498  
`score()` (*evalml.objectives.MCCBinary* method), 456  
`score()` (*evalml.objectives.MCCMulticlass* method), 459  
`score()` (*evalml.objectives.MeanSquaredLogError* method), 493  
`score()` (*evalml.objectives.MedianAE* method), 495  
`score()` (*evalml.objectives.MSE* method), 490  
`score()` (*evalml.objectives.MulticlassClassificationObjective* method), 402  
`score()` (*evalml.objectives.ObjectiveBase* method), 397  
`score()` (*evalml.objectives.Precision* method), 462  
`score()` (*evalml.objectives.PrecisionMacro* method), 467  
`score()` (*evalml.objectives.PrecisionMicro* method), 464  
`score()` (*evalml.objectives.PrecisionWeighted* method), 469  
`score()` (*evalml.objectives.R2* method), 483  
`score()` (*evalml.objectives.Recall* method), 472  
`score()` (*evalml.objectives.RecallMacro* method), 477  
`score()` (*evalml.objectives.RecallMicro* method), 475  
`score()` (*evalml.objectives.RecallWeighted* method), 480  
`score()` (*evalml.objectives.RegressionObjective* method), 405  
`score()` (*evalml.objectives.RootMeanSquaredError* method), 503  
`score()` (*evalml.objectives.RootMeanSquaredLogError* method), 505  
`score()` (*evalml.pipelines.BaselineBinaryPipeline* method), 208  
`score()` (*evalml.pipelines.BaselineMulticlassPipeline* method), 213  
`score()` (*evalml.pipelines.BaselineRegressionPipeline* method), 228  
`score()` (*evalml.pipelines.BinaryClassificationPipeline* method), 173  
`score()` (*evalml.pipelines.ClassificationPipeline* method), 168  
`score()` (*evalml.pipelines.MeanBaselineRegressionPipeline* method), 233  
`score()` (*evalml.pipelines.ModeBaselineBinaryPipeline* method), 218  
`score()` (*evalml.pipelines.ModeBaselineMulticlassPipeline* method), 223  
`score()` (*evalml.pipelines.MulticlassClassificationPipeline* method), 178  
`score()` (*evalml.pipelines.PipelineBase* method), 163  
`score()` (*evalml.pipelines.RegressionPipeline* method), 183  
`score()` (*evalml.pipelines.TimeSeriesBaselineRegressionPipeline* method), 238  
`score()` (*evalml.pipelines.TimeSeriesBinaryClassificationPipeline* method), 193  
`score()` (*evalml.pipelines.TimeSeriesClassificationPipeline* method), 188  
`score()` (*evalml.pipelines.TimeSeriesMulticlassClassificationPipeline* method), 198  
`score()` (*evalml.pipelines.TimeSeriesRegressionPipeline* method), 203  
`score_needs_proba` (*evalml.objectives.AccuracyBinary* attribute), 417  
`score_needs_proba` (*evalml.objectives.AccuracyMulticlass* attribute), 420  
`score_needs_proba` (*evalml.objectives.AUC* attribute), 422  
`score_needs_proba` (*evalml.objectives.AUCMacro* attribute), 425  
`score_needs_proba` (*evalml.objectives.AUCMicro* attribute), 428  
`score_needs_proba` (*evalml.objectives.AUCWeighted* attribute), 430  
`score_needs_proba` (*evalml.objectives.BalancedAccuracyBinary* attribute), 433  
`score_needs_proba` (*evalml.objectives.BalancedAccuracyMulticlass* attribute), 436  
`score_needs_proba` (*evalml.objectives.CostBenefitMatrix* attribute), 413  
`score_needs_proba` (*evalml.objectives.ExpVariance* attribute), 499  
`score_needs_proba` (*evalml.objectives.F1* attribute), 438  
`score_needs_proba` (*evalml.objectives.F1Macro* at-

[tribute](#)), 444  
[score\\_needs\\_proba \(evalml.objectives.F1Micro attribute\)](#), 441  
[score\\_needs\\_proba \(evalml.objectives.F1Weighted attribute\)](#), 446  
[score\\_needs\\_proba \(evalml.objectives.FraudCost attribute\)](#), 406  
[score\\_needs\\_proba \(evalml.objectives.LeadScoring attribute\)](#), 409  
[score\\_needs\\_proba \(evalml.objectives.LogLossBinary attribute\)](#), 449  
[score\\_needs\\_proba \(evalml.objectives.LogLossMulticlass attribute\)](#), 452  
[score\\_needs\\_proba \(evalml.objectives.MAE attribute\)](#), 484  
[score\\_needs\\_proba \(evalml.objectives.MAPE attribute\)](#), 486  
[score\\_needs\\_proba \(evalml.objectives.MaxError attribute\)](#), 496  
[score\\_needs\\_proba \(evalml.objectives.MCCBinary attribute\)](#), 454  
[score\\_needs\\_proba \(evalml.objectives.MCCMulticlass attribute\)](#), 457  
[score\\_needs\\_proba \(evalml.objectives.MeanSquaredLogError attribute\)](#), 491  
[score\\_needs\\_proba \(evalml.objectives.MedianAE attribute\)](#), 494  
[score\\_needs\\_proba \(evalml.objectives.MSE attribute\)](#), 489  
[score\\_needs\\_proba \(evalml.objectives.Precision attribute\)](#), 460  
[score\\_needs\\_proba \(evalml.objectives.PrecisionMacro attribute\)](#), 465  
[score\\_needs\\_proba \(evalml.objectives.PrecisionMicro attribute\)](#), 463  
[score\\_needs\\_proba \(evalml.objectives.PrecisionWeighted attribute\)](#), 468  
[score\\_needs\\_proba \(evalml.objectives.R2 attribute\)](#), 481  
[score\\_needs\\_proba \(evalml.objectives.Recall attribute\)](#), 470  
[score\\_needs\\_proba \(evalml.objectives.RecallMacro attribute\)](#), 476  
[score\\_needs\\_proba \(evalml.objectives.RecallMicro attribute\)](#), 473  
[score\\_needs\\_proba \(evalml.objectives.RecallWeighted attribute\)](#), 478  
[score\\_needs\\_proba \(evalml.objectives.RootMeanSquaredError attribute\)](#), 501  
[score\\_needs\\_proba \(evalml.objectives.RootMeanSquaredLogError attribute\)](#), 504  
[score\\_pipelines\(\) \(evalml.automl.AutoMLSearch method\)](#), 152  
[search\(\) \(evalml.automl.AutoMLSearch method\)](#), 152  
[SelectColumns \(class in evalml.pipelines.components\)](#), 253  
[silent\\_error\\_callback\(\) \(in module evalml.automl.callbacks\)](#), 159  
[SimpleImputer \(class in evalml.pipelines.components\)](#), 270  
[SKOptTuner \(class in evalml.tuners\)](#), 511  
[SMOTENCCVSplit \(class in evalml.preprocessing\)](#), 139  
[SMOTENCTVSplit \(class in evalml.preprocessing\)](#), 141  
[SMOTETomekCVSplit \(class in evalml.preprocessing\)](#), 132  
[SMOTETomekTVSplit \(class in evalml.preprocessing\)](#), 134  
[split\(\) \(evalml.preprocessing.BalancedClassificationDataCVSplit method\)](#), 128  
[split\(\) \(evalml.preprocessing.BalancedClassificationDataTVSplit method\)](#), 126  
[split\(\) \(evalml.preprocessing.KMeansSMOTECVSplit method\)](#), 129  
[split\(\) \(evalml.preprocessing.KMeansSMOTETVSplit method\)](#), 131  
[split\(\) \(evalml.preprocessing.RandomUnderSamplerCVSplit method\)](#), 137  
[split\(\) \(evalml.preprocessing.RandomUnderSamplerTVSplit method\)](#), 138  
[split\(\) \(evalml.preprocessing.SMOTENCCVSplit method\)](#), 140  
[split\(\) \(evalml.preprocessing.SMOTENCTVSplit method\)](#), 142  
[split\(\) \(evalml.preprocessing.SMOTETomekCVSplit method\)](#), 133  
[split\(\) \(evalml.preprocessing.SMOTETomekTVSplit method\)](#), 135  
[split\\_data\(\) \(in module evalml.preprocessing\)](#), 124  
[StackedEnsembleClassifier \(class in evalml.pipelines.components\)](#), 330  
[StackedEnsembleRegressor \(class in evalml.pipelines.components\)](#), 371  
[StandardScaler \(class in evalml.pipelines.components\)](#), 273  
[summary \(evalml.pipelines.BaselineBinaryPipeline attribute\)](#), 203  
[summary \(evalml.pipelines.BaselineMulticlassPipeline attribute\)](#), 203



[TimeSeriesClassificationPipeline](#) (class in [evalml.pipelines](#)), 183  
[TimeSeriesMulticlassClassificationPipeline](#) (class in [evalml.pipelines](#)), 194  
[TimeSeriesRegressionPipeline](#) (class in [evalml.pipelines](#)), 199  
[to\\_dict\(\)](#) ([evalml.data\\_checks.DataCheckError](#) method), 539  
[to\\_dict\(\)](#) ([evalml.data\\_checks.DataCheckMessage](#) method), 537  
[to\\_dict\(\)](#) ([evalml.data\\_checks.DataCheckWarning](#) method), 540  
[train\\_pipelines\(\)](#) ([evalml.automl.AutoMLSearch](#) method), 152  
[transform\(\)](#) ([evalml.pipelines.components.DateTimeFeaturizer](#) method), 288  
[transform\(\)](#) ([evalml.pipelines.components.DelayedFeatureTransformer](#) method), 295  
[transform\(\)](#) ([evalml.pipelines.components.DFSTransformer](#) method), 298  
[transform\(\)](#) ([evalml.pipelines.components.DropColumns](#) method), 252  
[transform\(\)](#) ([evalml.pipelines.components.DropNullColumns](#) method), 285  
[transform\(\)](#) ([evalml.pipelines.components.Imputer](#) method), 269  
[transform\(\)](#) ([evalml.pipelines.components.OneHotEncoder](#) method), 259  
[transform\(\)](#) ([evalml.pipelines.components.PerColumnImputer](#) method), 266  
[transform\(\)](#) ([evalml.pipelines.components.PolynomialDegreeReducer](#) method), 301  
[transform\(\)](#) ([evalml.pipelines.components.RFClassifierSelectFromModel](#) method), 282  
[transform\(\)](#) ([evalml.pipelines.components.RFRegressorSelectFromModel](#) method), 278  
[transform\(\)](#) ([evalml.pipelines.components.SelectColumns](#) method), 255  
[transform\(\)](#) ([evalml.pipelines.components.SimpleImputer](#) method), 272  
[transform\(\)](#) ([evalml.pipelines.components.StandardScaler](#) method), 275  
[transform\(\)](#) ([evalml.pipelines.components.TargetEncoder](#) method), 262  
[transform\(\)](#) ([evalml.pipelines.components.TextFeaturizer](#) method), 291  
[transform\(\)](#) ([evalml.pipelines.components.Transformer](#) method), 245  
[transform\(\)](#) ([evalml.pipelines.components.Undersampler](#) method), 304  
[transform\\_sample\(\)](#) ([evalml.preprocessing.BalancedClassificationDataCVSplit](#) method), 128  
[transform\\_sample\(\)](#) ([evalml.preprocessing.BalancedClassificationDataTVSplit](#) method), 126  
[transform\\_sample\(\)](#) ([evalml.preprocessing.KMeansSMOTECVSplit](#) method), 130  
[transform\\_sample\(\)](#) ([evalml.preprocessing.KMeansSMOTETVSplit](#) method), 131  
[transform\\_sample\(\)](#) ([evalml.preprocessing.RandomUnderSamplerCVSplit](#) method), 137  
[transform\\_sample\(\)](#) ([evalml.preprocessing.RandomUnderSamplerTVSplit](#) method), 139  
[transform\\_sample\(\)](#) ([evalml.preprocessing.SMOTENCCVSplit](#) method), 140  
[transform\\_sample\(\)](#) ([evalml.preprocessing.SMOTENCTVSplit](#) method), 142  
[transform\\_sample\(\)](#) ([evalml.preprocessing.SMOTETomekCVSplit](#) method), 133  
[transform\\_sample\(\)](#) ([evalml.preprocessing.SMOTETomekTVSplit](#) method), 135  
[transformer](#) (class in [evalml.pipelines.components](#)), 243  
[Tuner](#) (class in [evalml.tuners](#)), 510  
[Undersampler](#) (class in [evalml.pipelines.components](#)), 302  
[validate\(\)](#) ([evalml.data\\_checks.ClassImbalanceDataCheck](#) method), 529  
[validate\(\)](#) ([evalml.data\\_checks.DataCheck](#) method), 518  
[validate\(\)](#) ([evalml.data\\_checks.DataChecks](#) method), 534  
[validate\(\)](#) ([evalml.data\\_checks.DateTimeNaNDataCheck](#) method), 531  
[validate\(\)](#) ([evalml.data\\_checks.DefaultDataChecks](#) method), 536  
[validate\(\)](#) ([evalml.data\\_checks.HighlyNullDataCheck](#) method), 521  
[validate\(\)](#) ([evalml.data\\_checks.IDColumnsDataCheck](#) method), 522  
[validate\(\)](#) ([evalml.data\\_checks.InvalidTargetDataCheck](#) method), 519  
[validate\(\)](#) ([evalml.data\\_checks.MulticollinearityDataCheck](#) method), 530

`validate()` (*evalml.data\_checks.NaturalLanguageNaNDataCheck* [method](#)), [533](#)  
`validate()` (*evalml.data\_checks.NoVarianceDataCheck* [method](#)), [527](#)  
`validate()` (*evalml.data\_checks.OutliersDataCheck* [method](#)), [526](#)  
`validate()` (*evalml.data\_checks.TargetLeakageDataCheck* [method](#)), [524](#)  
`validate_inputs()` (*evalml.objectives.AccuracyBinary* [method](#)), [419](#)  
`validate_inputs()` (*evalml.objectives.AccuracyMulticlass* [method](#)), [422](#)  
`validate_inputs()` (*evalml.objectives.AUC* [method](#)), [425](#)  
`validate_inputs()` (*evalml.objectives.AUCMacro* [method](#)), [427](#)  
`validate_inputs()` (*evalml.objectives.AUCMicro* [method](#)), [430](#)  
`validate_inputs()` (*evalml.objectives.AUCWeighted* [method](#)), [432](#)  
`validate_inputs()` (*evalml.objectives.BalancedAccuracyBinary* [method](#)), [435](#)  
`validate_inputs()` (*evalml.objectives.BalancedAccuracyMulticlass* [method](#)), [438](#)  
`validate_inputs()` (*evalml.objectives.BinaryClassificationObjective* [method](#)), [400](#)  
`validate_inputs()` (*evalml.objectives.CostBenefitMatrix* [method](#)), [415](#)  
`validate_inputs()` (*evalml.objectives.ExpVariance* [method](#)), [501](#)  
`validate_inputs()` (*evalml.objectives.F1* [method](#)), [441](#)  
`validate_inputs()` (*evalml.objectives.F1Macro* [method](#)), [446](#)  
`validate_inputs()` (*evalml.objectives.F1Micro* [method](#)), [443](#)  
`validate_inputs()` (*evalml.objectives.F1Weighted* [method](#)), [448](#)  
`validate_inputs()` (*evalml.objectives.FraudCost* [method](#)), [409](#)  
`validate_inputs()` (*evalml.objectives.LeadScoring* [method](#)), [412](#)  
`validate_inputs()` (*evalml.objectives.LogLossBinary* [method](#)), [451](#)  
`validate_inputs()` (*evalml.objectives.LogLossMulticlass* [method](#)),  
`validate_inputs()` (*evalml.objectives.MAE* [method](#)), [486](#)  
`validate_inputs()` (*evalml.objectives.MAPE* [method](#)), [488](#)  
`validate_inputs()` (*evalml.objectives.MaxError* [method](#)), [498](#)  
`validate_inputs()` (*evalml.objectives.MCCBinary* [method](#)), [457](#)  
`validate_inputs()` (*evalml.objectives.MCCMulticlass* [method](#)), [459](#)  
`validate_inputs()` (*evalml.objectives.MeanSquaredLogError* [method](#)), [493](#)  
`validate_inputs()` (*evalml.objectives.MedianAE* [method](#)), [496](#)  
`validate_inputs()` (*evalml.objectives.MSE* [method](#)), [491](#)  
`validate_inputs()` (*evalml.objectives.MulticlassClassificationObjective* [method](#)), [403](#)  
`validate_inputs()` (*evalml.objectives.ObjectiveBase* [method](#)), [397](#)  
`validate_inputs()` (*evalml.objectives.Precision* [method](#)), [462](#)  
`validate_inputs()` (*evalml.objectives.PrecisionMacro* [method](#)), [467](#)  
`validate_inputs()` (*evalml.objectives.PrecisionMicro* [method](#)), [465](#)  
`validate_inputs()` (*evalml.objectives.PrecisionWeighted* [method](#)), [470](#)  
`validate_inputs()` (*evalml.objectives.R2* [method](#)), [483](#)  
`validate_inputs()` (*evalml.objectives.Recall* [method](#)), [473](#)  
`validate_inputs()` (*evalml.objectives.RecallMacro* [method](#)), [478](#)  
`validate_inputs()` (*evalml.objectives.RecallMicro* [method](#)), [475](#)  
`validate_inputs()` (*evalml.objectives.RecallWeighted* [method](#)), [480](#)  
`validate_inputs()` (*evalml.objectives.RegressionObjective* [method](#)), [405](#)  
`validate_inputs()` (*evalml.objectives.RootMeanSquaredError* [method](#)), [503](#)



`validate_inputs()`  
(*evalml.objectives.RootMeanSquaredLogError*  
*method*), 506

## X

XGBoostClassifier (class in  
*evalml.pipelines.components*), 323  
XGBoostRegressor (class in  
*evalml.pipelines.components*), 361