
EvalML Documentation

Release 0.17.0

Alteryx, Inc.

Dec 30, 2020

CONTENTS

1	Install	3
2	Start	5
3	Tutorials	9
4	User Guide	37
5	API Reference	97
6	Release Notes	453
	Index	481

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

INSTALL

EvalML is available for Python 3.6+. 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
```

For older Python versions (3.6.0 in particular), you must install `pip >= 18.0` and `setuptools > 40.0.0` in order to 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
```


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
```

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

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

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

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

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.**

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

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](#).

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

Generating pipelines to search over...
*****
* Beginning pipeline search *
*****

Optimizing for F1.
Greater score is better.

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

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

Batch 1: (1/9) Mode Baseline Binary Classification P... Elapsed:00:00
  Starting cross validation
  Finished cross validation - mean F1: 0.000
Batch 1: (2/9) Logistic Regression Classifier w/ Imp... Elapsed:00:00
  Starting cross validation
  Finished cross validation - mean F1: 0.976
Batch 1: (3/9) Random Forest Classifier w/ Imputer      Elapsed:00:02
  Starting cross validation
  Finished cross validation - mean F1: 0.949
Batch 1: (4/9) XGBoost Classifier w/ Imputer            Elapsed:00:03
  Starting cross validation
  Finished cross validation - mean F1: 0.932
Batch 1: (5/9) CatBoost Classifier w/ Imputer           Elapsed:00:04
  Starting cross validation
  Finished cross validation - mean F1: 0.901
Batch 1: (6/9) Elastic Net Classifier w/ Imputer + S... Elapsed:00:05
  Starting cross validation
  Finished cross validation - mean F1: 0.472
High coefficient of variation (cv >= 0.2) within cross validation scores. Elastic Net,
↳Classifier w/ Imputer + Standard Scaler may not perform as estimated on unseen data.
Batch 1: (7/9) Extra Trees Classifier w/ Imputer        Elapsed:00:05
  Starting cross validation
  Finished cross validation - mean F1: 0.933
Batch 1: (8/9) LightGBM Classifier w/ Imputer           Elapsed:00:06
  Starting cross validation
  Finished cross validation - mean F1: 0.943
Batch 1: (9/9) Decision Tree Classifier w/ Imputer      Elapsed:00:07
  Starting cross validation
  Finished cross validation - mean F1: 0.899

Search finished after 00:08
Best pipeline: Logistic Regression Classifier w/ Imputer + Standard Scaler
Best pipeline F1: 0.976081
```

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.

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

	id	pipeline_name	score	\
0	1	Logistic Regression Classifier w/ Imputer + St...	0.976081	
1	2	Random Forest Classifier w/ Imputer	0.948543	
2	7	LightGBM Classifier w/ Imputer	0.942803	
3	6	Extra Trees Classifier w/ Imputer	0.932522	
4	3	XGBoost Classifier w/ Imputer	0.931968	
5	4	CatBoost Classifier w/ Imputer	0.901433	
6	8	Decision Tree Classifier w/ Imputer	0.898505	
7	5	Elastic Net Classifier w/ Imputer + Standard S...	0.471727	
8	0	Mode Baseline Binary Classification Pipeline	0.000000	

	validation_score	percent_better_than_baseline	high_variance_cv	\
0	0.973451	NaN	False	
1	0.944444	NaN	False	
2	0.925926	NaN	False	
3	0.915888	NaN	False	
4	0.927273	NaN	False	
5	0.890909	NaN	False	
6	0.880734	NaN	False	
7	0.556962	NaN	True	
8	0.000000	NaN	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'}}

```

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

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

```

*****
* XGBoost Classifier w/ Imputer *
*****

Problem Type: binary
Model Family: XGBoost

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

```

(continues on next page)

(continued from previous page)

```
2. XGBoost Classifier
    * eta : 0.1
    * max_depth : 6
    * min_child_weight : 1
    * n_estimators : 100

Training
=====
Training for binary problems.
Total training time (including CV): 0.9 seconds

Cross Validation
-----
```

	F1	MCC	Binary Accuracy	Log Loss	Binary # Training	Binary # Validation	AUC	Precision	Balanced Accuracy
0	0.927	0.888	0.132	0.990	0.962	0.			
↪937	0.947	303.000	152.000						
1	0.948	0.917	0.096	0.994	0.932	0.			
↪961	0.961	303.000	152.000						
2	0.920	0.873	0.135	0.989	0.912	0.			
↪938	0.940	304.000	151.000						
mean	0.932	0.892	0.121	0.991	0.936	0.			
↪945	0.949	-	-						
std	0.015	0.022	0.022	0.003	0.025	0.			
↪014	0.010	-	-						
coef of var	0.016	0.025	0.181	0.003	0.027	0.			
↪015	0.011	-	-						

We can also view the pipeline parameters directly:

```
[8]: 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}, 'XGBoostClassifier': {'eta': 0.1, 'max_depth': 6, 'min_child_weight': 1, 'n_estimators': 100}}
```

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

```
[9]: pipeline = automl.best_pipeline
pipeline.fit(X_train, y_train)
pipeline.score(X_holdout, y_holdout, ["f1"])

[9]: OrderedDict([('F1', 0.963855421686747)])
```

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

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

```
[10]:
```

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 is over 4x 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=2500)
```

```

                Number of Features
Boolean                1
Categorical            6
Numeric               5

Number of training examples: 2500
Targets
False      85.92%
True       14.08%
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_state=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.
Lower score is better.
```

(continues on next page)

(continued from previous page)

```

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

```

```

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

```

```

Batch 1: (1/9) Mode Baseline Binary Classification P... Elapsed:00:00
  Starting cross validation
  Finished cross validation - mean Fraud Cost: 0.038
High coefficient of variation (cv >= 0.2) within cross validation scores. Mode_
↪ Baseline Binary Classification Pipeline may not perform as estimated on unseen data.
Batch 1: (2/9) Logistic Regression Classifier w/ Imp... Elapsed:00:00
  Starting cross validation
  Finished cross validation - mean Fraud Cost: 0.018
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.
Batch 1: (3/9) Random Forest Classifier w/ Imputer +... Elapsed:00:03
  Starting cross validation
  Finished cross validation - mean Fraud Cost: 0.002
Batch 1: (4/9) XGBoost Classifier w/ Imputer + One H... Elapsed:00:05
  Starting cross validation
  Finished cross validation - mean Fraud Cost: 0.002
Batch 1: (5/9) CatBoost Classifier w/ Imputer          Elapsed:00:07
  Starting cross validation
  Finished cross validation - mean Fraud Cost: 0.002
Batch 1: (6/9) Elastic Net Classifier w/ Imputer + O... Elapsed:00:08
  Starting cross validation
  Finished cross validation - mean Fraud Cost: 0.002
Batch 1: (7/9) Extra Trees Classifier w/ Imputer + O... Elapsed:00:09
  Starting cross validation
  Finished cross validation - mean Fraud Cost: 0.002
Batch 1: (8/9) LightGBM Classifier w/ Imputer + One ... Elapsed:00:11
  Starting cross validation
  Finished cross validation - mean Fraud Cost: 0.037
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.
Batch 1: (9/9) Decision Tree Classifier w/ Imputer +... Elapsed:00:13
  Starting cross validation
  Finished cross validation - mean Fraud Cost: 0.004
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.

Search finished after 00:14
Best pipeline: Random Forest Classifier w/ Imputer + One Hot Encoder
Best pipeline Fraud Cost: 0.002107

```

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	score	\
0	2	Random Forest Classifier w/ Imputer + One Hot ...	0.002107	
1	3	XGBoost Classifier w/ Imputer + One Hot Encoder	0.002107	
2	4	CatBoost Classifier w/ Imputer	0.002107	
3	5	Elastic Net Classifier w/ Imputer + One Hot En...	0.002107	
4	6	Extra Trees Classifier w/ Imputer + One Hot En...	0.002107	
5	8	Decision Tree Classifier w/ Imputer + One Hot ...	0.004263	
6	1	Logistic Regression Classifier w/ Imputer + On...	0.018184	
7	7	LightGBM Classifier w/ Imputer + One Hot Encoder	0.036774	
8	0	Mode Baseline Binary Classification Pipeline	0.038039	

	validation_score	percent_better_than_baseline	high_variance_cv	\
0	0.002436	94.461059	False	
1	0.002436	94.461059	False	
2	0.002436	94.461059	False	
3	0.002436	94.461059	False	
4	0.002436	94.461059	False	
5	0.006831	88.794105	True	
6	0.002436	52.196758	True	
7	0.037624	3.327100	True	
8	0.037393	0.000000	True	


```

                                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"])

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

Problem Type: binary
Model Family: XGBoost
```

(continues on next page)

(continued from previous page)

```

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 : None
    * handle_unknown : ignore
    * handle_missing : error
3. XGBoost Classifier
    * eta : 0.1
    * max_depth : 6
    * min_child_weight : 1
    * n_estimators : 100

Training
=====
Training for binary problems.
Objective to optimize binary classification pipeline thresholds for: <evalml.
↳objectives.fraud_cost.FraudCost object at 0x7f2bdebad370>
Total training time (including CV): 1.7 seconds

Cross Validation
-----

```

	Fraud Cost	AUC	F1	Precision	# Training	# Validation
0	0.002	0.854	0.247	0.141	1066.000	667.000
1	0.002	0.862	0.247	0.141	1066.000	667.000
2	0.002	0.857	0.247	0.141	1067.000	666.000
mean	0.002	0.858	0.247	0.141	-	-
std	0.000	0.004	0.000	0.000	-	-
coef of var	0.146	0.005	0.001	0.001	-	-

3.1.3 Evaluate on holdout data

Finally, we retrain the best pipeline on all of the training data and evaluate on the holdout data.

```

[9]: best_pipeline.fit(X_train, y_train)

[9]: GeneratedPipeline(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': None, 'handle_unknown': 'ignore', 'handle_missing':
↳'error'}, 'Random Forest Classifier':{'n_estimators': 100, 'max_depth': 6, 'n_jobs':
↳ -1}},))

```

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.

```

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

```

```
[10]: OrderedDict([('AUC', 0.8563787375415283),
                  ('Fraud Cost', 0.0013994749372859567)])
```

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.

```
[11]: 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.

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

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

Batch 1: (1/9) Mode Baseline Binary Classification P... Elapsed:00:00
  Starting cross validation
  Finished cross validation - mean AUC: 0.500
Batch 1: (2/9) Logistic Regression Classifier w/ Imp... Elapsed:00:00
  Starting cross validation
  Finished cross validation - mean AUC: 0.802
Batch 1: (3/9) Random Forest Classifier w/ Imputer +... Elapsed:00:01
  Starting cross validation
  Finished cross validation - mean AUC: 0.835
Batch 1: (4/9) XGBoost Classifier w/ Imputer + One H... Elapsed:00:02
  Starting cross validation
  Finished cross validation - mean AUC: 0.856
Batch 1: (5/9) CatBoost Classifier w/ Imputer          Elapsed:00:03
  Starting cross validation
  Finished cross validation - mean AUC: 0.855
Batch 1: (6/9) Elastic Net Classifier w/ Imputer + O... Elapsed:00:04
  Starting cross validation
  Finished cross validation - mean AUC: 0.500
Batch 1: (7/9) Extra Trees Classifier w/ Imputer + O... Elapsed:00:04
  Starting cross validation
  Finished cross validation - mean AUC: 0.825
```

(continues on next page)

(continued from previous page)

```

Batch 1: (8/9) LightGBM Classifier w/ Imputer + One ... Elapsed:00:06
    Starting cross validation
    Finished cross validation - mean AUC: 0.860
Batch 1: (9/9) Decision Tree Classifier w/ Imputer +... Elapsed:00:07
    Starting cross validation
    Finished cross validation - mean AUC: 0.834

Search finished after 00:07
Best pipeline: LightGBM Classifier w/ Imputer + One Hot Encoder
Best pipeline AUC: 0.860129

```

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

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

```

[12]:   id      pipeline_name      score \
0    7  LightGBM Classifier w/ Imputer + One Hot Encoder  0.860129
1    3  XGBoost Classifier w/ Imputer + One Hot Encoder  0.855796
2    4                CatBoost Classifier w/ Imputer  0.854697
3    2  Random Forest Classifier w/ Imputer + One Hot ...  0.835397
4    8  Decision Tree Classifier w/ Imputer + One Hot ...  0.833551
5    6  Extra Trees Classifier w/ Imputer + One Hot En...  0.825415
6    1  Logistic Regression Classifier w/ Imputer + On...  0.802190
7    0      Mode Baseline Binary Classification Pipeline  0.500000
8    5  Elastic Net Classifier w/ Imputer + One Hot En...  0.500000

      validation_score  percent_better_than_baseline  high_variance_cv \
0          0.853960          72.025737          False
1          0.860068          71.159288          False
2          0.841976          70.939409          False
3          0.845048          67.079476          False
4          0.860588          66.710203          False
5          0.825266          65.082986          False
6          0.800546          60.437939          False
7          0.500000           0.000000          False
8          0.500000           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...

```

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

Again, we retrain the best pipeline on all of the training data and evaluate on the holdout data.

```
[14]: best_pipeline_auc.fit(X_train, y_train)
```

```

[14]: GeneratedPipeline(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': None, '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_fraction': 0.9, 'bagging_frequency': 0.9},})

```

(continues on next page)

(continued from previous page)

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

[15]: OrderedDict([('AUC', 0.8441860465116278), ('Fraud Cost', 0.03657633836707538)])

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

[16]: OrderedDict([('AUC', 0.8563787375415283),
                  ('Fraud Cost', 0.0013994749372859567)])
```

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 fraud cost. However, the losses due to fraud are over 3% of the total transaction amount when optimized for AUC and under 1% when optimized for fraud cost. As a result, we lose more than 2% of the total transaction amount by not optimizing for fraud cost specifically.

This happens because optimizing for AUC does not take into account the user-specified `retry_percentage`, `interchange_fee`, `fraud_payout_percentage` values. 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 over 6x 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			
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_state=0)

print(X.types)
```

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

Because the lead scoring labels are binary, we will use `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',
    ↳ objective=lead_scoring_objective,
    ↳ additional_objectives=['auc'],
    ↳ max_batches=1,
    ↳ optimize_thresholds=True)

automl.search()

{'message': 'The following labels fall below 10% of the target: [True]', 'data_check_
    ↳ name': 'ClassImbalanceDataCheck', 'level': 'warning', 'code': 'CLASS_IMBALANCE_
    ↳ BELOW_THRESHOLD', 'details': {'target_values': [True]}}
Generating pipelines to search over...
*****
* Beginning pipeline search *
*****

Optimizing for Lead Scoring.
Greater score is better.

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

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

Batch 1: (1/9) Mode Baseline Binary Classification P... Elapsed:00:00
  Starting cross validation
  Finished cross validation - mean Lead Scoring: 0.000
Batch 1: (2/9) Logistic Regression Classifier w/ Imp... Elapsed:00:01
  Starting cross validation
```

(continues on next page)

(continued from previous page)

```

    Finished cross validation - mean Lead Scoring: 13.784
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.
Batch 1: (3/9) Random Forest Classifier w/ Imputer +... Elapsed:00:04
    Starting cross validation
    Finished cross validation - mean Lead Scoring: 12.062
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.
Batch 1: (4/9) XGBoost Classifier w/ Imputer + One H... Elapsed:00:06
    Starting cross validation
    Finished cross validation - mean Lead Scoring: 13.173
Batch 1: (5/9) CatBoost Classifier w/ Imputer          Elapsed:00:08
    Starting cross validation
    Finished cross validation - mean Lead Scoring: 12.640
High coefficient of variation (cv >= 0.2) within cross validation scores. CatBoost_
↳Classifier w/ Imputer may not perform as estimated on unseen data.
Batch 1: (6/9) Elastic Net Classifier w/ Imputer + O... Elapsed:00:09
    Starting cross validation
    Finished cross validation - mean Lead Scoring: 15.997
Batch 1: (7/9) Extra Trees Classifier w/ Imputer + O... Elapsed:00:11
    Starting cross validation
    Finished cross validation - mean Lead Scoring: 13.028
Batch 1: (8/9) LightGBM Classifier w/ Imputer + One ... Elapsed:00:12
    Starting cross validation
    Finished cross validation - mean Lead Scoring: 14.635
Batch 1: (9/9) Decision Tree Classifier w/ Imputer +... Elapsed:00:14
    Starting cross validation
    Finished cross validation - mean Lead Scoring: 11.531

Search finished after 00:15
Best pipeline: Elastic Net Classifier w/ Imputer + One Hot Encoder + Standard Scaler
Best pipeline Lead Scoring: 15.997325

```

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  score \
0   5  Elastic Net Classifier w/ Imputer + One Hot En... 15.997325
1   7  LightGBM Classifier w/ Imputer + One Hot Encoder 14.635258
2   1  Logistic Regression Classifier w/ Imputer + On... 13.784020
3   3  XGBoost Classifier w/ Imputer + One Hot Encoder 13.172704
4   6  Extra Trees Classifier w/ Imputer + One Hot En... 13.028294
5   4  CatBoost Classifier w/ Imputer                  12.640320
6   2  Random Forest Classifier w/ Imputer + One Hot ... 12.062318
7   8  Decision Tree Classifier w/ Imputer + One Hot ... 11.531380
8   0  Mode Baseline Binary Classification Pipeline    0.000000

 validation_score  percent_better_than_baseline  high_variance_cv \
0          12.374194                        NaN                False

```

(continues on next page)

(continued from previous page)

1	12.419355	NaN	False
2	9.980645	NaN	True
3	11.316129	NaN	False
4	12.496774	NaN	False
5	12.374194	NaN	True
6	11.212903	NaN	True
7	10.161290	NaN	False
8	0.000000	NaN	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`.

```
[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"])

*****
* Elastic Net 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 : None
    * handle_unknown : ignore
    * handle_missing : error
3. Standard Scaler
4. Elastic Net Classifier
    * alpha : 0.5
```

(continues on next page)

(continued from previous page)

```

* l1_ratio : 0.5
* n_jobs : -1
* max_iter : 1000
* penalty : elasticnet
* loss : log

Training
=====
Training for binary problems.
Objective to optimize binary classification pipeline thresholds for: <evalml.
->objectives.lead_scoring.LeadScoring object at 0x7f0c5c2835e0>
Total training time (including CV): 1.7 seconds

Cross Validation
-----

```

	Lead Scoring	AUC	# Training	# Validation
0	12.374	0.500	2479.000	1550.000
1	18.058	0.500	2479.000	1550.000
2	17.560	0.500	2480.000	1549.000
mean	15.997	0.500	-	-
std	3.148	0.000	-	-
coef of var	0.197	0.000	-	-

3.2.4 Evaluate on hold out

Finally, we retrain the best pipeline on all of the training data and evaluate it on the holdout dataset.

```

[10]: best_pipeline.fit(X_train, y_train)

[10]: GeneratedPipeline(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': None, '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'},})

```

Now, we can score the pipeline on the holdout data using both the lead scoring score and the AUC.

```

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

[11]: OrderedDict([('AUC', 0.5), ('Lead Scoring', 15.382631126397248)])

```

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.

```

[12]: automl_auc = evalml.AutoMLSearch(X_train=X_train, y_train=y_train,
                                     problem_type='binary',
                                     objective='auc',
                                     additional_objectives=[],
                                     max_batches=1,
                                     optimize_thresholds=True)

```

(continues on next page)

(continued from previous page)

```

automl_auc.search()

{'message': 'The following labels fall below 10% of the target: [True]', 'data_check_
↳name': 'ClassImbalanceDataCheck', 'level': 'warning', 'code': 'CLASS_IMBALANCE_
↳BELOW_THRESHOLD', 'details': {'target_values': [True]}}
Generating pipelines to search over...
*****
* Beginning pipeline search *
*****

Optimizing for AUC.
Greater score is better.

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

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

Batch 1: (1/9) Mode Baseline Binary Classification P... Elapsed:00:00
  Starting cross validation
  Finished cross validation - mean AUC: 0.500
Batch 1: (2/9) Logistic Regression Classifier w/ Imp... Elapsed:00:00
  Starting cross validation
  Finished cross validation - mean AUC: 0.695
Batch 1: (3/9) Random Forest Classifier w/ Imputer +... Elapsed:00:00
  Starting cross validation
  Finished cross validation - mean AUC: 0.702
Batch 1: (4/9) XGBoost Classifier w/ Imputer + One H... Elapsed:00:01
  Starting cross validation
  Finished cross validation - mean AUC: 0.709
Batch 1: (5/9) CatBoost Classifier w/ Imputer          Elapsed:00:03
  Starting cross validation
  Finished cross validation - mean AUC: 0.557
Batch 1: (6/9) Elastic Net Classifier w/ Imputer + O... Elapsed:00:03
  Starting cross validation
  Finished cross validation - mean AUC: 0.500
Batch 1: (7/9) Extra Trees Classifier w/ Imputer + O... Elapsed:00:04
  Starting cross validation
  Finished cross validation - mean AUC: 0.722
Batch 1: (8/9) LightGBM Classifier w/ Imputer + One ... Elapsed:00:04
  Starting cross validation
  Finished cross validation - mean AUC: 0.678
Batch 1: (9/9) Decision Tree Classifier w/ Imputer +... Elapsed:00:05
  Starting cross validation
  Finished cross validation - mean AUC: 0.587

Search finished after 00:06
Best pipeline: Extra Trees Classifier w/ Imputer + One Hot Encoder
Best pipeline AUC: 0.722008

```

[13]: automl_auc.rankings

```
[13]:
```

	id	pipeline_name	score	\
0	6	Extra Trees Classifier w/ Imputer + One Hot En...	0.722008	
1	3	XGBoost Classifier w/ Imputer + One Hot Encoder	0.708940	
2	2	Random Forest Classifier w/ Imputer + One Hot ...	0.701576	
3	1	Logistic Regression Classifier w/ Imputer + On...	0.694572	
4	7	LightGBM Classifier w/ Imputer + One Hot Encoder	0.678321	
5	8	Decision Tree Classifier w/ Imputer + One Hot ...	0.586996	
6	4	CatBoost Classifier w/ Imputer	0.556956	
7	0	Mode Baseline Binary Classification Pipeline	0.500000	
8	5	Elastic Net Classifier w/ Imputer + One Hot En...	0.500000	

	validation_score	percent_better_than_baseline	high_variance_cv	\
0	0.718159	44.401691	False	
1	0.696173	41.788046	False	
2	0.699430	40.315221	False	
3	0.685761	38.914392	False	
4	0.668776	35.664140	False	
5	0.560719	17.399261	False	
6	0.581441	11.391175	False	
7	0.500000	0.000000	False	
8	0.500000	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.

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

# train on the full training data
best_pipeline_auc.fit(X_train, y_train)

[14]: GeneratedPipeline(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': None, '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},})

[15]: # get the auc and lead scoring score on holdout data
best_pipeline_auc.score(X_holdout, y_holdout, objectives=["auc", lead_scoring_
→objective])

[15]: OrderedDict([('AUC', 0.6800619522514355),
('Lead Scoring', -0.008598452278589854)])
```

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 gained was only \$7 per lead when optimized for AUC and was \$45 when optimized for lead scoring. As a result, we would gain up to 6x 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_state=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.

(continues on next page)

(continued from previous page)

```

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

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

Batch 1: (1/9) Mode Baseline Binary Classification P... Elapsed:00:00
  Starting cross validation
  Finished cross validation - mean Log Loss Binary: 9.164
Batch 1: (2/9) Logistic Regression Classifier w/ Imp... Elapsed:00:00
  Starting cross validation
  Finished cross validation - mean Log Loss Binary: 0.423
Batch 1: (3/9) Random Forest Classifier w/ Imputer +... Elapsed:00:04
  Starting cross validation
  Finished cross validation - mean Log Loss Binary: 0.424
Batch 1: (4/9) XGBoost Classifier w/ Imputer + One H... Elapsed:00:07
  Starting cross validation
  Finished cross validation - mean Log Loss Binary: 0.430
Batch 1: (5/9) CatBoost Classifier w/ Imputer          Elapsed:00:11
  Starting cross validation
  Finished cross validation - mean Log Loss Binary: 0.602
Batch 1: (6/9) Elastic Net Classifier w/ Imputer + O... Elapsed:00:12
  Starting cross validation
  Finished cross validation - mean Log Loss Binary: 0.579
Batch 1: (7/9) Extra Trees Classifier w/ Imputer + O... Elapsed:00:14
  Starting cross validation
  Finished cross validation - mean Log Loss Binary: 0.432
Batch 1: (8/9) LightGBM Classifier w/ Imputer + One ... Elapsed:00:17
  Starting cross validation
    Fold 0: Encountered an error.
    Fold 0: All scores will be replaced with nan.
    Fold 0: Please check /home/docs/checkouts/readthedocs.org/
↳user_builds/feature-labs-inc-evalml/checkouts/v0.17.0/docs/source/demos/evalml_
↳debug.log for the current hyperparameters and stack trace.
    Fold 0: Exception during automl search: Check failed: (best_
↳split_info.right_count) > (0) at /__w/l/s/python-package/compile/src/treearner/
↳serial_tree_learner.cpp, line 661 .

    Finished cross validation - mean Log Loss Binary: 0.436
Batch 1: (9/9) Decision Tree Classifier w/ Imputer +... Elapsed:00:19
  Starting cross validation
  Finished cross validation - mean Log Loss Binary: 0.618

Search finished after 00:21
Best pipeline: Logistic Regression Classifier w/ Imputer + One Hot Encoder + Standard_
↳Scaler
Best pipeline Log Loss Binary: 0.422831

[3]: OrderedDict([('Log Loss Binary', 0.41630102955822595)])

```

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', 25.13014671083768)])
```

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

176991.62328442978
```

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.

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

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

Batch 1: (1/9) Mode Baseline Binary Classification P... Elapsed:00:00
  Starting cross validation
  Finished cross validation - mean Cost Benefit Matrix: -53.063
Batch 1: (2/9) Logistic Regression Classifier w/ Imp... Elapsed:00:00
  Starting cross validation
  Finished cross validation - mean Cost Benefit Matrix: 26.674
Batch 1: (3/9) Random Forest Classifier w/ Imputer +... Elapsed:00:04
  Starting cross validation
  Finished cross validation - mean Cost Benefit Matrix: 16.836
Batch 1: (4/9) XGBoost Classifier w/ Imputer + One H... Elapsed:00:06
  Starting cross validation
  Finished cross validation - mean Cost Benefit Matrix: 25.174
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.
Batch 1: (5/9) CatBoost Classifier w/ Imputer Elapsed:00:10
  Starting cross validation
  Finished cross validation - mean Cost Benefit Matrix: 15.801
```

(continues on next page)

(continued from previous page)

```
High coefficient of variation (cv >= 0.2) within cross validation scores. CatBoost_
↳Classifier w/ Imputer may not perform as estimated on unseen data.
Batch 1: (6/9) Elastic Net Classifier w/ Imputer + O... Elapsed:00:11
    Starting cross validation
    Finished cross validation - mean Cost Benefit Matrix: -53.063
Batch 1: (7/9) Extra Trees Classifier w/ Imputer + O... Elapsed:00:14
    Starting cross validation
    Finished cross validation - mean Cost Benefit Matrix: 14.138
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.
Batch 1: (8/9) LightGBM Classifier w/ Imputer + One ... Elapsed:00:16
    Starting cross validation
        Fold 0: Encountered an error.
        Fold 0: All scores will be replaced with nan.
        Fold 0: Please check /home/docs/checkouts/readthedocs.org/
↳user_builds/feature-labs-inc-evalml/checkouts/v0.17.0/docs/source/demos/evalml_
↳debug.log for the current hyperparameters and stack trace.
        Fold 0: Exception during automl search: Check failed: (best_
↳split_info.right_count) > (0) at /__w/l/s/python-package/compile/src/treelearner/
↳serial_tree_learner.cpp, line 661 .

    Finished cross validation - mean Cost Benefit Matrix: 25.015
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.
Batch 1: (9/9) Decision Tree Classifier w/ Imputer +... Elapsed:00:18
    Starting cross validation
    Finished cross validation - mean Cost Benefit Matrix: 7.484
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.

Search finished after 00:20
Best pipeline: Logistic Regression Classifier w/ Imputer + One Hot Encoder + Standard_
↳Scaler
Best pipeline Cost Benefit Matrix: 26.674162
```

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', 25.13014671083768)])
```

```
[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)

176991.62328442978
```

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



```
0.0
```

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...
```

(continues on next page)

(continued from previous page)

```

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_state=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.

```

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

```

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

```

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

Batch 1: (1/9) Mode Baseline Binary Classification P... Elapsed:00:00
  Starting cross validation
  Finished cross validation - mean Log Loss Binary: 8.638
Batch 1: (2/9) Logistic Regression Classifier w/ Imp... Elapsed:00:00
  Starting cross validation
  Finished cross validation - mean Log Loss Binary: 0.170
High coefficient of variation (cv >= 0.2) within cross validation scores. Logistic_
↪Regression Classifier w/ Imputer + Text Featurization Component + Standard Scaler_
↪may not perform as estimated on unseen data.
Batch 1: (3/9) Random Forest Classifier w/ Imputer +... Elapsed:00:11
  Starting cross validation
  Finished cross validation - mean Log Loss Binary: 0.123
Batch 1: (4/9) XGBoost Classifier w/ Imputer + Text ... Elapsed:00:21
  Starting cross validation
  Finished cross validation - mean Log Loss Binary: 0.129
Batch 1: (5/9) CatBoost Classifier w/ Imputer + Text... Elapsed:00:30
  Starting cross validation
  Finished cross validation - mean Log Loss Binary: 0.519
Batch 1: (6/9) Elastic Net Classifier w/ Imputer + T... Elapsed:00:39
  Starting cross validation
  Finished cross validation - mean Log Loss Binary: 0.531
Batch 1: (7/9) Extra Trees Classifier w/ Imputer + T... Elapsed:00:48
  Starting cross validation
  Finished cross validation - mean Log Loss Binary: 0.220
Batch 1: (8/9) LightGBM Classifier w/ Imputer + Text... Elapsed:00:57
  Starting cross validation
  Finished cross validation - mean Log Loss Binary: 0.171
Batch 1: (9/9) Decision Tree Classifier w/ Imputer +... Elapsed:01:06
  Starting cross validation
  Finished cross validation - mean Log Loss Binary: 0.726

Search finished after 01:15
Best pipeline: Random Forest Classifier w/ Imputer + Text Featurization Component
Best pipeline Log Loss Binary: 0.123494
```

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	score \
0	2	Random Forest Classifier w/ Imputer + Text Fea...	0.123494
1	3	XGBoost Classifier w/ Imputer + Text Featuriza...	0.128840
2	1	Logistic Regression Classifier w/ Imputer + Te...	0.170443
3	7	LightGBM Classifier w/ Imputer + Text Featuriz...	0.171467
4	6	Extra Trees Classifier w/ Imputer + Text Featu...	0.219864
5	4	CatBoost Classifier w/ Imputer + Text Featuriz...	0.518860
6	5	Elastic Net Classifier w/ Imputer + Text Featu...	0.530537
7	8	Decision Tree Classifier w/ Imputer + Text Fea...	0.726110
8	0	Mode Baseline Binary Classification Pipeline	8.638305

(continues on next page)

(continued from previous page)

	validation_score	percent_better_than_baseline	high_variance_cv	\
0	0.129007	98.570392	False	
1	0.134231	98.508509	False	
2	0.148005	98.026895	True	
3	0.183049	98.015036	False	
4	0.250778	97.454783	False	
5	0.522827	93.993496	False	
6	0.561991	93.858323	False	
7	0.595490	91.594296	False	
8	8.623860	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`.

```
[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/ Imputer + Text Featurization Component *
*****

Problem Type: binary
Model Family: Random Forest

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

Training
=====
Training for binary problems.
```

(continues on next page)

(continued from previous page)

Total training time (including CV): 9.5 seconds

Cross Validation

	Log Loss	Binary Accuracy	Binary MCC	Binary AUC	Precision	F1	Balanced Accuracy
↪ Binary	Accuracy	Binary # Training	# Validation				
0		0.129	0.885	0.979	0.937	0.912	0.
↪ 935	0.957	1594.000	797.000				
1		0.127	0.892	0.987	0.937	0.918	0.
↪ 940	0.960	1594.000	797.000				
2		0.114	0.895	0.984	0.952	0.920	0.
↪ 937	0.961	1594.000	797.000				
mean		0.123	0.891	0.983	0.942	0.917	0.
↪ 937	0.959	-	-	-	-	-	-
std		0.008	0.005	0.004	0.009	0.004	0.
↪ 003	0.002	-	-	-	-	-	-
coef of var		0.065	0.006	0.004	0.009	0.004	0.
↪ 003	0.002	-	-	-	-	-	-

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

```
[10]:
```

Notice above that there is a Text Featurization Component as the second 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

Finally, we retrain the best pipeline on all of the training data and evaluate on the holdout data.

```
[11]: best_pipeline.fit(X_train, y_train)
```

```
[11]: GeneratedPipeline(parameters={'Imputer':{'categorical_impute_strategy': 'most_frequent',
↪ 'numeric_impute_strategy': 'mean', 'categorical_fill_value': None, 'numeric_fill_
↪ value': None}, 'Text Featurization Component':{'text_columns': ['Message']},
↪ 'Random Forest Classifier':{'n_estimators': 100, 'max_depth': 6, 'n_jobs': -1},})
```

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

```
[12]: scores = best_pipeline.score(X_holdout, y_holdout, objectives=evalml.objectives.get_
↪ core_objectives('binary'))
print(f'Accuracy Binary: {scores["Accuracy Binary"]}')
Accuracy Binary: 0.9732441471571907
```

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.

```
[13]: import woodwork as ww
X_train_categorical = X_train.set_types(logical_types={'Message': 'Categorical'})

[14]: 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.

Searching up to 1 batches for a total of 9 pipelines.
Allowed model families: extra_trees, linear_model, random_forest, decision_tree,
↳catboost, lightgbm, xgboost
```

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

```
Batch 1: (1/9) Mode Baseline Binary Classification P... Elapsed:00:00
  Starting cross validation
  Finished cross validation - mean Log Loss Binary: 8.638
Batch 1: (2/9) Logistic Regression Classifier w/ Imp... Elapsed:00:00
  Starting cross validation
  Finished cross validation - mean Log Loss Binary: 0.170
High coefficient of variation (cv >= 0.2) within cross validation scores. Logistic_
↳Regression Classifier w/ Imputer + Text Featurization Component + Standard Scaler_
↳may not perform as estimated on unseen data.
Batch 1: (3/9) Random Forest Classifier w/ Imputer +... Elapsed:00:08
  Starting cross validation
  Finished cross validation - mean Log Loss Binary: 0.123
Batch 1: (4/9) XGBoost Classifier w/ Imputer + Text ... Elapsed:00:18
  Starting cross validation
  Finished cross validation - mean Log Loss Binary: 0.129
Batch 1: (5/9) CatBoost Classifier w/ Imputer + Text... Elapsed:00:27
  Starting cross validation
  Finished cross validation - mean Log Loss Binary: 0.519
Batch 1: (6/9) Elastic Net Classifier w/ Imputer + T... Elapsed:00:36
  Starting cross validation
  Finished cross validation - mean Log Loss Binary: 0.531
Batch 1: (7/9) Extra Trees Classifier w/ Imputer + T... Elapsed:00:45
  Starting cross validation
  Finished cross validation - mean Log Loss Binary: 0.220
Batch 1: (8/9) LightGBM Classifier w/ Imputer + Text... Elapsed:00:54
```

(continues on next page)

(continued from previous page)

```

    Starting cross validation
    Finished cross validation - mean Log Loss Binary: 0.171
Batch 1: (9/9) Decision Tree Classifier w/ Imputer +... Elapsed:01:03
    Starting cross validation
    Finished cross validation - mean Log Loss Binary: 0.726

Search finished after 01:12
Best pipeline: Random Forest Classifier w/ Imputer + Text Featurization Component
Best pipeline Log Loss Binary: 0.123494

```

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

```
[15]: automl_no_text.rankings
```

```

[15]:   id      pipeline_name      score \
0    2  Random Forest Classifier w/ Imputer + Text Fea...  0.123494
1    3  XGBoost Classifier w/ Imputer + Text Featuriza...  0.128840
2    1  Logistic Regression Classifier w/ Imputer + Te...  0.170443
3    7  LightGBM Classifier w/ Imputer + Text Featuriz...  0.171467
4    6  Extra Trees Classifier w/ Imputer + Text Featu...  0.219864
5    4  CatBoost Classifier w/ Imputer + Text Featuriz...  0.518860
6    5  Elastic Net Classifier w/ Imputer + Text Featu...  0.530537
7    8  Decision Tree Classifier w/ Imputer + Text Fea...  0.726110
8    0      Mode Baseline Binary Classification Pipeline  8.638305

      validation_score  percent_better_than_baseline  high_variance_cv \
0          0.129007          98.570392          False
1          0.134231          98.508509          False
2          0.148005          98.026895           True
3          0.183049          98.015036          False
4          0.250778          97.454783          False
5          0.522827          93.993496          False
6          0.561991          93.858323          False
7          0.595490          91.594296          False
8          8.623860           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'}}

```

```
[16]: 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.

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

```
[17]:
```

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

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

Problem Type: binary
Model Family: Random Forest

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

Training
=====
Training for binary problems.
Total training time (including CV): 9.5 seconds

Cross Validation
-----
```

	Log Loss Binary	MCC Binary	AUC	Precision	F1	Balanced Accuracy
Binary Accuracy	0.129	0.885	0.979	0.937	0.912	0.
935	0.957	1594.000	797.000			
1	0.127	0.892	0.987	0.937	0.918	0.
940	0.960	1594.000	797.000			
2	0.114	0.895	0.984	0.952	0.920	0.
937	0.961	1594.000	797.000			
mean	0.123	0.891	0.983	0.942	0.917	0.
937	0.959	-	-			
std	0.008	0.005	0.004	0.009	0.004	0.
003	0.002	-	-			
coef of var	0.065	0.006	0.004	0.009	0.004	0.
003	0.002	-	-			

```
[19]: # train on the full training data
best_pipeline_no_text.fit(X_train, y_train)

# 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: 0.9732441471571907
```

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` did not beat the random guess of predicting “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.

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(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: 99992
Targets
False      84.82%
True       15.18%
Name: fraud, dtype: object
```

```
[2]: 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.
```

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

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

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

```

Batch 1: (1/9) Mode Baseline Binary Classification P... Elapsed:00:00
    Starting cross validation
    Finished cross validation - mean Log Loss Binary: 5.243
Batch 1: (2/9) Logistic Regression Classifier w/ Imp... Elapsed:00:01
    Starting cross validation
    Finished cross validation - mean Log Loss Binary: 0.326
Batch 1: (3/9) Random Forest Classifier w/ Imputer +... Elapsed:00:29
    Starting cross validation
    Finished cross validation - mean Log Loss Binary: 0.252
Batch 1: (4/9) XGBoost Classifier w/ Imputer + DateT... Elapsed:01:00
    Starting cross validation
    Finished cross validation - mean Log Loss Binary: 0.189
Batch 1: (5/9) CatBoost Classifier w/ Imputer + Date... Elapsed:02:00
    Starting cross validation
    Finished cross validation - mean Log Loss Binary: 0.507
Batch 1: (6/9) Elastic Net Classifier w/ Imputer + D... Elapsed:02:19
    Starting cross validation
    Finished cross validation - mean Log Loss Binary: 0.426
Batch 1: (7/9) Extra Trees Classifier w/ Imputer + D... Elapsed:02:43
    Starting cross validation
    Finished cross validation - mean Log Loss Binary: 0.355
Batch 1: (8/9) LightGBM Classifier w/ Imputer + Date... Elapsed:03:09
    Starting cross validation
    Finished cross validation - mean Log Loss Binary: 0.190
Batch 1: (9/9) Decision Tree Classifier w/ Imputer +... Elapsed:03:33
    Starting cross validation
    Finished cross validation - mean Log Loss Binary: 0.265
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.

Search finished after 03:55
Best pipeline: XGBoost Classifier w/ Imputer + DateTime Featurization Component + One_
↪Hot Encoder
Best pipeline Log Loss Binary: 0.188964

```

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.

```
[3]: 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)

[3]: <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.

```
[4]: 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
```

Data Checks

`AutoMLSearch.search` runs a set of data checks before beginning the search process to ensure that the input data being passed will not run into some common issues before running a potentially time-consuming search. If the data checks find any potential errors, an exception will be thrown before the search begins, allowing users to inspect their data to avoid confusing errors that may arise later during the search process.

This behavior is controlled by the `data_checks` parameter which can take in either a `DataChecks` object, a list of `DataCheck` objects, `None`, or valid string inputs (`"disabled"`, `"auto"`). By default, this parameter is set to `auto`, which runs the default collection of data sets defined in the `DefaultDataChecks` class. If set to `"disabled"` or `None`, no data checks will run.

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.

```
[5]: 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.

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.

```
[6]: automl.rankings

[6]:   id  pipeline_name  score \
0    3  XGBoost Classifier w/ Imputer + DateTime Featu...  0.188964
1    7  LightGBM Classifier w/ Imputer + DateTime Feat...  0.190269
2    2  Random Forest Classifier w/ Imputer + DateTime...  0.252276
3    8  Decision Tree Classifier w/ Imputer + DateTime...  0.264999
4    1  Logistic Regression Classifier w/ Imputer + Da...  0.326190
5    6  Extra Trees Classifier w/ Imputer + DateTime F...  0.354583
6    5  Elastic Net Classifier w/ Imputer + DateTime F...  0.426029
7    4  CatBoost Classifier w/ Imputer + DateTime Feat...  0.507124
8    0  Mode Baseline Binary Classification Pipeline  5.243060

   validation_score  percent_better_than_baseline  high_variance_cv \
0          0.191819             96.395914             False
1          0.193413             96.371025             False
2          0.239547             95.188384             False
3          0.361423             94.945727              True
4          0.328929             93.778639             False
5          0.349722             93.237090             False
6          0.426286             91.874414             False
7          0.507580             90.327707             False
8          5.243353              0.000000             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'}}

```

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`.

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

```

*****
* Logistic Regression Classifier w/ Imputer + DateTime Featurization Component + 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. 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 : None
    * handle_unknown : ignore
    * handle_missing : error
4. Standard Scaler
5. Logistic Regression Classifier
    * penalty : l2
    * C : 1.0
    * n_jobs : -1
    * multi_class : auto
    * solver : lbfgs

Training
=====
Training for binary problems.
Total training time (including CV): 28.5 seconds

Cross Validation

```

(continues on next page)

(continued from previous page)

	Log Loss Binary	MCC Binary	AUC	Precision	F1	Balanced Accuracy
Binary Accuracy	Binary # Training	Binary # Validation				
0	0.329	0.552	0.831	0.997	0.509	0.
→671	0.900	66661.000	33331.000			
1	0.324	0.533	0.836	0.990	0.486	0.
→661	0.897	66661.000	33331.000			
2	0.325	0.533	0.835	0.993	0.486	0.
→661	0.897	66662.000	33330.000			
mean	0.326	0.539	0.834	0.994	0.494	0.
→664	0.898	-	-			
std	0.002	0.011	0.003	0.003	0.013	0.
→006	0.002	-	-			
coef of var	0.007	0.020	0.003	0.003	0.027	0.
→009	0.002	-	-			

4.1.5 Get Pipeline

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

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

```
Logistic Regression Classifier w/ Imputer + DateTime Featurization Component + One_
→Hot Encoder + Standard Scaler
{'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': None, 'handle_unknown': 'ignore',
→'handle_missing': 'error'}, 'Logistic Regression Classifier': {'penalty': 'l2', 'C':
→1.0, 'n_jobs': -1, 'multi_class': 'auto', 'solver': 'lbfgs'}}
```

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.

```
[9]: 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': None, 'handle_unknown': 'ignore',
→'handle_missing': 'error'}, 'XGBoost Classifier': {'eta': 0.1, 'max_depth': 6, 'min_
→child_weight': 1, 'n_estimators': 100}}
```

```
[9]: 0      False
      1      False
      2      False
      3      False
      4      False
      ...
      99987   False
      99988   False
      99989   False
      99990   False
      99991    True
      Length: 99992, dtype: bool
```

4.1.6 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, use `make_pipeline` to define a pipeline with a custom hyperparameter range. Hyperparameter ranges can be found through the [API reference](#) for each component. Note: The default value of the component must be included in any specified hyperparameter range for AutoMLSearch to succeed. Additionally, the parameter value must be specified as a list, even for just one value.

```
[10]: from evalml import AutoMLSearch
      from evalml.demos import load_fraud
      from evalml.pipelines.components.utils import get_estimators
      from evalml.model_family import ModelFamily
      from evalml.pipelines.utils import make_pipeline
      import woodwork as ww

      X, y = load_fraud()

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

      # limit the numeric impute strategy to include only `mean` and `median`
      # `mean` is the default value for this argument, and it needs to be included in the
      # specified hyperparameter range
      custom_hyperparameters = {'Imputer': {
          'numeric_impute_strategy': ['mean', 'median']
      }}

      estimators = get_estimators('binary', [ModelFamily.EXTRA_TREES])
      pipelines_with_custom_hyperparameters = [make_pipeline(X, y, estimator, 'binary',
          custom_hyperparameters) for estimator in estimators]
      automl = AutoMLSearch(X_train=X, y_train=y, problem_type='binary', allowed_
          pipelines=pipelines_with_custom_hyperparameters)
      automl.search()
      automl.best_pipeline.hyperparameters
```

	Number of Features
Boolean	1
Categorical	6
Numeric	5

(continues on next page)

(continued from previous page)

```

Number of training examples: 99992
Targets
False      84.82%
True       15.18%
Name: fraud, dtype: object
Using default limit of max_batches=1.

```

```

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

```

```

Optimizing for Log Loss Binary.
Lower score is better.

```

```

Searching up to 1 batches for a total of 2 pipelines.
Allowed model families: extra_trees

```

```

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

```

```

Batch 1: (1/2) Mode Baseline Binary Classification P... Elapsed:00:00
  Starting cross validation
  Finished cross validation - mean Log Loss Binary: 5.243
Batch 1: (2/2) Extra Trees Classifier w/ Imputer + D... Elapsed:00:01
  Starting cross validation
  Finished cross validation - mean Log Loss Binary: 0.360

```

```

Search finished after 00:24
Best pipeline: Extra Trees Classifier w/ Imputer + DateTime Featurization Component +
↳ One Hot Encoder
Best pipeline Log Loss Binary: 0.360414

```

```

[10]: {'Imputer': {'categorical_impute_strategy': ['most_frequent'],
                  'numeric_impute_strategy': ['mean', 'median']},
       'DateTime Featurization Component': {},
       'One Hot Encoder': {},
       'Extra Trees Classifier': {'n_estimators': Integer(low=10, high=1000, prior='uniform',
↳ transform='identity'),
                                  'max_features': ['auto', 'sqrt', 'log2'],
                                  'max_depth': Integer(low=4, high=10, prior='uniform', transform='identity')}}

```

4.1.7 Access raw results

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

```

[11]: automl.results
[11]: {'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'}}},

```

(continues on next page)

(continued from previous page)

```

'score': 5.243060307948458,
'high_variance_cv': False,
'training_time': 1.3644850254058838,
'cv_data': [{'all_objective_scores': OrderedDict([('Log Loss Binary',
5.243353291477845),
('MCC Binary', 0.0),
('AUC', 0.5),
('Precision', 0.0),
('F1', 0.0),
('Balanced Accuracy Binary', 0.5),
('Accuracy Binary', 0.848189373256128),
('# Training', 66661),
('# Validation', 33331))]),
'score': 5.243353291477845,
'binary_classification_threshold': 0.5},
{'all_objective_scores': OrderedDict([('Log Loss Binary',
5.243353291477846),
('MCC Binary', 0.0),
('AUC', 0.5),
('Precision', 0.0),
('F1', 0.0),
('Balanced Accuracy Binary', 0.5),
('Accuracy Binary', 0.848189373256128),
('# Training', 66661),
('# Validation', 33331))]),
'score': 5.243353291477846,
'binary_classification_threshold': 0.5},
{'all_objective_scores': OrderedDict([('Log Loss Binary',
5.242474340889683),
('MCC Binary', 0.0),
('AUC', 0.5),
('Precision', 0.0),
('F1', 0.0),
('Balanced Accuracy Binary', 0.5),
('Accuracy Binary', 0.8482148214821482),
('# Training', 66662),
('# Validation', 33330))]),
'score': 5.242474340889683,
'binary_classification_threshold': 0.5}},
'percent_better_than_baseline_all_objectives': {'Log Loss Binary': 0,
'MCC Binary': nan,
'AUC': 0,
'Precision': nan,
'F1': nan,
'Balanced Accuracy Binary': 0,
'Accuracy Binary': 0},
'percent_better_than_baseline': 0,
'validation_score': 5.243353291477845},
1: {'id': 1,
'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': 'mean',
'categorical_fill_value': None,

```

(continues on next page)

(continued from previous page)

```

    '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': None,
    '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}},
    'score': 0.3604142375161434,
    'high_variance_cv': False,
    'training_time': 23.1845965385437,
    'cv_data': [{'all_objective_scores': OrderedDict([('Log Loss Binary',
    0.35722523278292945),
    ('MCC Binary', 0.0),
    ('AUC', 0.8327890925252948),
    ('Precision', 0.0),
    ('F1', 0.0),
    ('Balanced Accuracy Binary', 0.5),
    ('Accuracy Binary', 0.848189373256128),
    ('# Training', 66661),
    ('# Validation', 33331)]),
    'score': 0.35722523278292945,
    'binary_classification_threshold': 0.5},
    {'all_objective_scores': OrderedDict([('Log Loss Binary',
    0.35710265483449705),
    ('MCC Binary', 0.0),
    ('AUC', 0.8375894242385562),
    ('Precision', 0.0),
    ('F1', 0.0),
    ('Balanced Accuracy Binary', 0.5),
    ('Accuracy Binary', 0.848189373256128),
    ('# Training', 66661),
    ('# Validation', 33331)]),
    'score': 0.35710265483449705,
    'binary_classification_threshold': 0.5},
    {'all_objective_scores': OrderedDict([('Log Loss Binary',
    0.36691482493100364),
    ('MCC Binary', 0.0),
    ('AUC', 0.8349621647188481),
    ('Precision', 0.0),
    ('F1', 0.0),
    ('Balanced Accuracy Binary', 0.5),
    ('Accuracy Binary', 0.8482148214821482),
    ('# Training', 66662),
    ('# Validation', 33330)]),
    'score': 0.36691482493100364,
    'binary_classification_threshold': 0.5}],

```

(continues on next page)

(continued from previous page)

```

    'percent_better_than_baseline_all_objectives': {'Log Loss Binary': 93.
↪12588037620402,
    'MCC Binary': nan,
    'AUC': 67.0227120988466,
    'Precision': nan,
    'F1': nan,
    'Balanced Accuracy Binary': 0,
    'Accuracy Binary': 0},
    'percent_better_than_baseline': 93.12588037620402,
    'validation_score': 0.35722523278292945}},
    'search_order': [0, 1],
    'errors': []}

```

4.1.8 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.

```

[12]: 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.

Searching up to 5 batches for a total of 20 pipelines.
Allowed model families: linear_model, random_forest

```

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

```

```

Batch 1: (1/20) Mode Baseline Binary Classification P... Elapsed:00:00
    Starting cross validation
    Finished cross validation - mean Log Loss Binary: 12.868
Batch 1: (2/20) Logistic Regression Classifier w/ Imp... Elapsed:00:00
    Starting cross validation
    Finished cross validation - mean Log Loss Binary: 0.076
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.
Batch 1: (3/20) Random Forest Classifier w/ Imputer      Elapsed:00:00
    Starting cross validation
    Finished cross validation - mean Log Loss Binary: 0.174
High coefficient of variation (cv >= 0.2) within cross validation scores. Random_
↳Forest Classifier w/ Imputer may not perform as estimated on unseen data.
Batch 1: (4/20) Elastic Net Classifier w/ Imputer + S... Elapsed:00:02
    Starting cross validation
    Finished cross validation - mean Log Loss Binary: 0.503
Batch 2: (5/20) Logistic Regression Classifier w/ Imp... Elapsed:00:02
    Starting cross validation
    Finished cross validation - mean Log Loss Binary: 0.118
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.
Batch 2: (6/20) Logistic Regression Classifier w/ Imp... Elapsed:00:03
    Starting cross validation
    Finished cross validation - mean Log Loss Binary: 0.104
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.
Batch 2: (7/20) Logistic Regression Classifier w/ Imp... Elapsed:00:03
    Starting cross validation
    Finished cross validation - mean Log Loss Binary: 0.092
Batch 2: (8/20) Logistic Regression Classifier w/ Imp... Elapsed:00:04
    Starting cross validation
    Finished cross validation - mean Log Loss Binary: 0.115
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.
Batch 2: (9/20) Logistic Regression Classifier w/ Imp... Elapsed:00:05
    Starting cross validation
    Finished cross validation - mean Log Loss Binary: 0.097
High coefficient of variation (cv >= 0.2) within cross validation scores. Logistic_
↳Regression Classifier w/ Imputer + Standard Scaler may not perform as estimated on_
↳unseen data.
Batch 3: (10/20) Random Forest Classifier w/ Imputer      Elapsed:00:05
    Starting cross validation
    Finished cross validation - mean Log Loss Binary: 0.137
Batch 3: (11/20) Random Forest Classifier w/ Imputer      Elapsed:00:08
    Starting cross validation
    Finished cross validation - mean Log Loss Binary: 0.158
Batch 3: (12/20) Random Forest Classifier w/ Imputer      Elapsed:00:09
    Starting cross validation
    Finished cross validation - mean Log Loss Binary: 0.119
Batch 3: (13/20) Random Forest Classifier w/ Imputer      Elapsed:00:11
    Starting cross validation
    Finished cross validation - mean Log Loss Binary: 0.117
Batch 3: (14/20) Random Forest Classifier w/ Imputer      Elapsed:00:14

```

(continues on next page)

(continued from previous page)

```

    Starting cross validation
    Finished cross validation - mean Log Loss Binary: 0.120
Batch 4: (15/20) Elastic Net Classifier w/ Imputer + S... Elapsed:00:15
    Starting cross validation
    Finished cross validation - mean Log Loss Binary: 0.660
Batch 4: (16/20) Elastic Net Classifier w/ Imputer + S... Elapsed:00:16
    Starting cross validation
    Finished cross validation - mean Log Loss Binary: 0.627
Batch 4: (17/20) Elastic Net Classifier w/ Imputer + S... Elapsed:00:16
    Starting cross validation
    Finished cross validation - mean Log Loss Binary: 0.588
Batch 4: (18/20) Elastic Net Classifier w/ Imputer + S... Elapsed:00:17
    Starting cross validation
    Finished cross validation - mean Log Loss Binary: 0.341
Batch 4: (19/20) Elastic Net Classifier w/ Imputer + S... Elapsed:00:17
    Starting cross validation
    Finished cross validation - mean Log Loss Binary: 0.534
Batch 5: (20/20) Stacked Ensemble Classification Pipeline Elapsed:00:18
    Starting cross validation
    Finished cross validation - mean Log Loss Binary: 0.073

Search finished after 00:21
Best pipeline: Stacked Ensemble Classification Pipeline
Best pipeline Log Loss Binary: 0.073164

```

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

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

```

*****
* Stacked Ensemble Classification Pipeline *
*****

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

Pipeline Steps
=====
1. Stacked Ensemble Classifier
    * input_pipelines : [GeneratedPipeline(parameters={'Imputer':{'categorical_
    ↪impute_strategy': 'most_frequent', 'numeric_impute_strategy': 'mean', 'categorical_
    ↪fill_value': None, 'numeric_fill_value': None}, 'Logistic Regression Classifier':{'
    ↪penalty': 'l2', 'C': 1.0, 'n_jobs': -1, 'multi_class': 'auto', 'solver': 'lbfgs'}},
    ↪), GeneratedPipeline(parameters={'Imputer':{'categorical_impute_strategy': 'most_
    ↪frequent', 'numeric_impute_strategy': 'most_frequent', 'categorical_fill_value': _
    ↪None, 'numeric_fill_value': None}, 'Random Forest Classifier':{'n_estimators': 301,
    ↪'max_depth': 9, 'n_jobs': -1}},)]
    * final_estimator : None
    * cv : None
    * n_jobs : 1

```

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
```

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
```

(continues on next page)

(continued from previous page)

```

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', 0.7142857142857143)])
0.38695754323100356
OrderedDict([('F1', 1.0)])

```

4.2.3 Custom Objectives

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

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

Defining a Custom Objective Function

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

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

- `perfect_score`: The score achieved by a perfect model on this objective.

Example: Fraud Detection

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

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

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

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

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

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

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

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

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

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

        Returns:
            pd.Series: Series of predicted fraud labels using X and threshold
        """
```

(continues on next page)

(continued from previous page)

```

    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)

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

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

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

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

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

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

    loss_per_total_processed = loss / transaction_amount.sum()

    return loss_per_total_processed

```

4.3 Components

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

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

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

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

4.3.1 Transformers

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

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

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

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

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

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

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

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

```
   0    1    2
0  1.0  2.0  3.0
1  1.0  2.0  3.0
```

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: DFS Transformer
Transformer: Delayed Feature Transformer
Transformer: Text Featurization Component
```

(continues on next page)

(continued from previous page)

```

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: Standard Scaler
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]: 0      0
     1      0
dtype: int64

```

Below is a list of all estimators included with EvalML:

```

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

Estimator: Stacked Ensemble Regressor
Estimator: Stacked Ensemble Classifier
Estimator: Time Series Baseline Regressor
Estimator: Decision Tree Regressor
Estimator: Baseline Regressor
Estimator: Extra Trees Regressor
Estimator: XGBoost Regressor
Estimator: CatBoost Regressor
Estimator: Random Forest Regressor

```

(continues on next page)

(continued from previous page)

```

Estimator: LightGBM Regressor
Estimator: Linear Regressor
Estimator: Elastic Net Regressor
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]: import pandas as pd
from evalml.pipelines.components import Transformer

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_state=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_state=random_state)

```

(continues on next page)

(continued from previous page)

```

def fit(self, X, y=None):
    pct_null_threshold = self.parameters["pct_null_threshold"]
    if not isinstance(X, pd.DataFrame):
        X = pd.DataFrame(X)
    percent_null = X.isnull().mean()
    if pct_null_threshold == 0.0:
        null_cols = percent_null[percent_null > 0]
    else:
        null_cols = percent_null[percent_null >= pct_null_threshold]
    self._cols_to_drop = list(null_cols.index)
    return self

def transform(self, X, y=None):
    """Transforms data X by dropping columns that exceed the threshold of null_
    ↪ values.
    Arguments:
        X (pd.DataFrame): Data to transform
        y (pd.Series, optional): Targets
    Returns:
        pd.DataFrame: Transformed X
    """
    if not isinstance(X, pd.DataFrame):
        X = pd.DataFrame(X)
    return X.drop(columns=self._cols_to_drop, axis=1)

```

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_state` 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.
- `transform()` - after fitting a component, the `transform()` method will take in new data and transform accordingly. 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]

    def __init__(self, strategy="mean", random_state=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_state (int, np.random.RandomState): seed for the random number_
            ↪ generator

        """
        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_state=random_state)

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

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

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

(continues on next page)

(continued from previous page)

```

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

@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:
['random_forest', 'xgboost', 'lightgbm', 'linear_model', 'catboost', 'extra_trees',
↪'ensemble', 'decision_tree', '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_state` 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_state=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_state=random_state)

    @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.

```
[10]: 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."""
```

(continues on next page)

(continued from previous page)

```

name = "Linear Regressor"
hyperparameter_ranges = {
    'fit_intercept': [True, False],
    'normalize': [True, False]
}
model_family = ModelFamily.LINEAR_MODEL
supported_problem_types = [ProblemTypes.REGRESSION]

def __init__(self, fit_intercept=True, normalize=False, n_jobs=-1, random_state=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_state=random_state)

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

```

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.

```

[11]: 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'})

[12]: # this string can then be copy and pasted into a separate window and executed as
      → python code
      exec(code)
      logisticRegressionClassifier

[12]: LogisticRegressionClassifier(penalty='l2', C=5, n_jobs=-1, multi_class='auto', solver=
      → 'lbfgs')

[13]: # custom component
      from evalml.pipelines.components import Transformer

```

(continues on next page)

(continued from previous page)

```

import pandas as pd
from evalml.pipelines.components.utils import generate_component_code

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_state=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_state=random_state)

    def fit(self, X, y=None):
        pct_null_threshold = self.parameters["pct_null_threshold"]
        if not isinstance(X, pd.DataFrame):
            X = pd.DataFrame(X)
        percent_null = X.isnull().mean()
        if pct_null_threshold == 0.0:
            null_cols = percent_null[percent_null > 0]
        else:
            null_cols = percent_null[percent_null >= pct_null_threshold]
        self._cols_to_drop = list(null_cols.index)
        return self

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

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

        Returns:
            pd.DataFrame: Transformed X
        """
        if not isinstance(X, pd.DataFrame):
            X = pd.DataFrame(X)
        return X.drop(columns=self._cols_to_drop, axis=1)

```

(continues on next page)

(continued from previous page)

```
myDropNull = MyDropNullColumns()
print(generate_component_code(myDropNull))

myDropNullColumnsTransformer = MyDropNullColumns(**{'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'],
```

(continues on next page)

(continued from previous page)

```

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

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']))

0      class_0
1      class_0
2      class_0
3      class_0
4      class_0
...
173     class_2
174     class_2
175     class_2
176     class_2
177     class_2
Length: 178, dtype: object
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 lists or 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)

class CustomPipeline(MulticlassClassificationPipeline):
    component_graph = ['Imputer', 'One Hot Encoder', 'Standard Scaler', 'Logistic_
↳Regression Classifier']
    custom_hyperparameters = {
        'Simple Imputer' : {
            'impute_strategy': Categorical(['most_frequent']),
            # Can also pass in a list, like below
            'another_hyperparameter': ['value']
        }
    }

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.
↳01, high=10, prior='uniform', transform='identity')}}
```

(continues on next page)

(continued from previous page)

```
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.
↪01, 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': None, '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,
    }
}

cp = CustomPipeline(parameters=parameters, random_state=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 : None
    * 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 *
*****

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 : None
    * handle_unknown : ignore
    * handle_missing : error
3. Elastic Net Classifier
    * alpha : 0.5
    * l1_ratio : 0.5
    * n_jobs : -1
```

(continues on next page)

(continued from previous page)

```

        * 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')

```

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`.

```

[15]: cp.estimator
[15]: 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.

```

[16]: pipeline.input_feature_names
[16]: {'Imputer': ['alcohol',
    'malic_acid',
    'ash',
    'alcalinity_of_ash',
    'magnesium',

```

(continues on next page)

(continued from previous page)

```
'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

```
[17]: from evalml.pipelines.utils import generate_pipeline_code
from evalml.pipelines import MulticlassClassificationPipeline
import pandas as pd

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_state=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.")
```

(continues on next page)

(continued from previous page)

```

parameters = {"pct_null_threshold": pct_null_threshold}
parameters.update(kwargs)

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

def fit(self, X, y=None):
    pct_null_threshold = self.parameters["pct_null_threshold"]
    if not isinstance(X, pd.DataFrame):
        X = pd.DataFrame(X)
    percent_null = X.isnull().mean()
    if pct_null_threshold == 0.0:
        null_cols = percent_null[percent_null > 0]
    else:
        null_cols = percent_null[percent_null >= pct_null_threshold]
    self._cols_to_drop = list(null_cols.index)
    return self

def transform(self, X, y=None):
    """Transforms data X by dropping columns that exceed the threshold of null_
    ↪ values.
    Arguments:
        X (pd.DataFrame): Data to transform
        y (pd.Series, optional): Targets
    Returns:
        pd.DataFrame: Transformed X
    """

    if not isinstance(X, pd.DataFrame):
        X = pd.DataFrame(X)
    return X.drop(columns=self._cols_to_drop, axis=1)

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
    },
    "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": null,
        "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

```

(continues on next page)

(continued from previous page)

```
False      84.82%
True       15.18%
Name: fraud, dtype: object
```

```
[17]: 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': None, '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 RFBinaryClassificationPipeline(evalml.pipelines.BinaryClassificationPipeline):
    component_graph = ['Simple Imputer', 'Random Forest Classifier']

X, y = evalml.demos.load_breast_cancer()

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

```
[2]: pipeline.feature_importance
```

```
[2]:
```

	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

```
[3]: 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.

```
[4]: from evalml.model_understanding.graphs import calculate_permutation_importance
```

```
calculate_permutation_importance(pipeline, X, y, 'log loss binary')
```

```
[4]:
```

	feature	importance
0	worst perimeter	0.078033
1	worst radius	0.074341
2	worst concave points	0.068313

(continues on next page)

(continued from previous page)

3	worst area	0.067733
4	mean concave points	0.041261
5	worst concavity	0.037533
6	mean concavity	0.036664
7	area error	0.035838
8	mean perimeter	0.025783
9	mean area	0.025203
10	worst texture	0.016211
11	perimeter error	0.011738
12	mean texture	0.011716
13	radius error	0.010910
14	mean radius	0.010775
15	worst compactness	0.008322
16	worst smoothness	0.008281
17	mean smoothness	0.005707
18	worst symmetry	0.004454
19	worst fractal dimension	0.003889
20	concavity error	0.003858
21	compactness error	0.003572
22	concave points error	0.003449
23	mean compactness	0.003173
24	smoothness error	0.003172
25	fractal dimension error	0.002618
26	texture error	0.002533
27	mean fractal dimension	0.002228
28	symmetry error	0.002126
29	mean symmetry	0.001786

```
[5]: 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 [partial dependence plots](#) for a feature.

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

```
[6]: 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]

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

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.

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

```
[8]:
```

	benign	malignant
benign	1.000000	0.000000
malignant	0.009434	0.990566

```
[9]: 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.

```
[10]: 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.

```
[11]: 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.

```
[12]: 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`.

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

```
[13]:
```

	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]

```
[14]: 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.

```
[15]: 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

Explaining Individual Predictions

We can explain why the model made an individual prediction with the `explain_prediction` function. This will use the [Shapley Additive Explanations \(SHAP\)](#) algorithms 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 (must correspond to one row of the input data) and the training data. 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%.

```
[16]: from evalml.model_understanding.prediction_explanations import explain_prediction

table = explain_prediction(pipeline=pipeline, input_features=X.iloc[3:4],
                           training_data=X, top_k=6, include_shap_values=True)
print(table)
```

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

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.

Explaining Multiple 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_prediction` 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.

```
[17]: 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	+

(continues on next page)

(continued from previous page)

↪ 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				
Target Value: malignant				
Cross Entropy: 0.0				
Index ID: 564				
↪SHAP Value	Feature Name	Feature Value	Contribution to Prediction	↪
↪=====				
↪ 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				
↪SHAP Value	Feature Name	Feature Value	Contribution to Prediction	↪
↪=====				
↪0.04	smoothness error	0.00	+	↪
↪0.03	mean texture	21.58	+	↪
↪0.02	worst texture	30.25	+	↪
↪0.02	worst area	787.90	+	↪

(continues on next page)

(continued from previous page)

	worst radius	15.93	-	-
↪0.03				
	mean concave points	0.02	-	-
↪0.03				
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				
↪SHAP Value	Feature Name	Feature Value	Contribution to Prediction	
↪=====				
	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)
```

We can also manually explain predictions on any subset of the training data with the [explain_predictions](#) function. Below, we explain the predictions on the first, fifth, and tenth row of the data.

```
[18]: from evalml.model_understanding.prediction_explanations import explain_predictions
```

(continues on next page)

(continued from previous page)

```
report = explain_predictions(pipeline=pipeline, input_features=X.iloc[[0, 4, 9]],
    include_shap_values=True)
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	+

Changing Output Formats

Instead of getting the prediction explanations as text, you can get the report as a python dictionary. All you have to do is pass `output_format="dict"` to either `explain_prediction`, `explain_predictions`, or `explain_predictions_best_worst`.

```
[19]: import json
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,
            122.3
          ],
          "qualitative_explanation": [
            "+",
            "+",
            "+",
            "+",
            "+",
            "+"
          ],
          "quantitative_explanation": [
```

(continues on next page)

(continued from previous page)

```

        0.09988982304983156,
        0.08240174808629956,
        0.07868368954615064,
        0.06242860386204596,
        0.051970789425386396,
        0.04459155806887927
    ],
    "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
            ],
            "qualitative_explanation": [
                "+",
                "+",
                "-",
                "-",
                "-",
                "-"
            ],
            "quantitative_explanation": [
                0.05245422607466413,
                0.03035933540832274,
                -0.03461744299818247,

```

(continues on next page)

(continued from previous page)

```

        -0.04174884967530769,
        -0.0491285663898271,
        -0.05666940833106337
    ],
    "class_name": "malignant"
}
]
}
]
}

```

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` used in `AutoMLSearch.search`.

4.6.1 Missing Data

Missing data or rows with NaN values provide many challenges for machine learning pipelines. In the worst case, many algorithms simply will not run with missing data! EvalML pipelines contain imputation *components* to ensure that doesn't happen. Imputation works by approximating missing values with existing values. However, if a column contains a high number of missing values, a large percentage of the column would be approximated by a small percentage. This could potentially create a column without useful information for machine learning pipelines. By using `HighlyNullDataCheck`, EvalML will alert you to this potential problem by returning the columns that pass the missing values threshold.

```

[1]: import numpy as np
import pandas as pd

from evalml.data_checks import HighlyNullDataCheck

X = pd.DataFrame([[1, 2, 3],
                  [0, 4, np.nan],
                  [1, 4, np.nan],
                  [9, 4, np.nan],
                  [8, 6, np.nan]])

null_check = HighlyNullDataCheck(pct_null_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:

- OutliersDataCheck
- ClassImbalanceDataCheck
- IDColumnsDataCheck
- NoVarianceDataCheck
- HighVarianceCVDataCheck
- InvalidTargetDataCheck

Zero Variance

Data with zero variance indicates that all values are identical. If a feature has zero variance, it is not likely to be a useful feature. Similarly, if the target has zero variance, there is likely something wrong. `NoVarianceDataCheck` checks if the target or any feature has only one unique value and alerts you to any such columns.

```
[2]: from evalml.data_checks import NoVarianceDataCheck
X = pd.DataFrame({"no var col": [0, 0, 0],
                  "good col": [0, 4, 1]})
y = pd.Series([1, 0, 1])
no_variance_data_check = NoVarianceDataCheck()
results = no_variance_data_check.validate(X, y)

for message in results['warnings']:
    print("Warning:", message['message'])

for message in results['errors']:
    print("Error:", message['message'])
```

Error: no var col has 1 unique value.

Note that you can set `NaN` to count as an unique value, but `NoVarianceDataCheck` will still return a warning if there is only one unique non-`NaN` value in a given column.

```
[3]: from evalml.data_checks import NoVarianceDataCheck

X = pd.DataFrame({"no var col": [0, 0, 0],
                  "no var col with nan": [1, np.nan, 1],
                  "good col": [0, 4, 1]})
y = pd.Series([1, 0, 1])

no_variance_data_check = NoVarianceDataCheck(count_nan_as_value=True)
results = no_variance_data_check.validate(X, y)

for message in results['warnings']:
    print("Warning:", message['message'])

for message in results['errors']:
    print("Error:", message['message'])
```

Warning: no var col with nan has two unique values including nulls. Consider encoding the nulls for this column to be useful for machine learning.
 Error: no var col has 1 unique value.

Class Imbalance

For classification problems, the distribution of examples across each class can vary. For small variations, this is normal and expected. However, when the number of examples for each class label is disproportionately biased or skewed towards a particular class (or classes), it can be difficult for machine learning models to predict well. In addition, having a low number of examples for a given class could mean that one or more of the CV folds generated for the training data could only have few or no examples from that class. This may cause the model to only predict the majority class and ultimately resulting in a poor-performant model.

`ClassImbalanceDataCheck` checks if the target labels are imbalanced beyond a specified threshold for a certain number of CV folds. It returns `DataCheckError` messages for any classes that have less samples than double the number of CV folds specified (since that indicates the likelihood of having at little to no samples of that class in a given fold), and `DataCheckWarning` messages for any classes that fall below the set threshold percentage.

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

X = pd.DataFrame([[1, 2, 0, 1],
                  [4, 1, 9, 0],
                  [4, 4, 8, 3],
                  [9, 2, 7, 1]])
y = pd.Series([0, 1, 1, 1, 1])

class_imbalance_check = ClassImbalanceDataCheck(threshold=0.25, num_cv_folds=4)
results = class_imbalance_check.validate(X, y)

for message in results['warnings']:
    print("Warning:", message['message'])

for message in results['errors']:
    print("Error:", message['message'])

Warning: The following labels fall below 25% of the target: [0]
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'])
```

(continues on next page)

(continued from previous page)

```
for message in results['errors']:
    print("Error:", message['message'])
```

```
Warning: Column 'leak' is 80.0% or more correlated with the target
```

Invalid Target Data

The `InvalidTargetDataCheck` checks if the target data contains any missing or invalid values. Specifically:

- if any of the target values are missing, a `DataCheckError` message is returned
- if the specified problem type is a binary classification problem but there is more or less than two unique values in the target, a `DataCheckError` message is returned
- if binary classification target classes are numeric values not equal to `{0, 1}`, a `DataCheckError` message is returned because it can cause unpredictable behavior when passed to pipelines

```
[6]: from evalml.data_checks import InvalidTargetDataCheck
```

```
X = pd.DataFrame({})
y = pd.Series([0, 1, None, None])

invalid_target_check = InvalidTargetDataCheck('binary')
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'])
```

```
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 exist. 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
```

High Variance Cross-Validation Scores

The `HighVarianceCVDDataCheck` data check is used in `AutoMLSearch` to detect if the variance between folds in cross-validation is higher than a specified threshold. High variance across cross-validation folds indicates that the underlying model may be overfitting to the fold data; this is unfavorable and can create an underperforming model in production.

The `HighVarianceCVDDataCheck` is unique because it is the only data check that is not run before the search in `AutoMLSearch.search()` begins, but rather, during each CV fold.

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.

```
[8]: 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.

```
[9]: 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.

```
[10]: 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]})
```

(continues on next page)

(continued from previous page)

```

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.

```

[11]: 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 and used in `AutoMLSearch`.

Default Data Checks

By default, `AutoMLSearch.search` runs a collection of data checks before it searches and iterates over pipelines. This collection of data checks is stored in the `DefaultDataChecks` class. It consists of a few data checks that are generally helpful for any machine learning problem. They are:

- `HighlyNullDataCheck`
- `IDColumnsDataCheck`
- `TargetLeakageDataCheck`
- `InvalidTargetDataCheck`
- `ClassImbalanceDataCheck` (for classification problem types)

- NoVarianceDataCheck

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.

```
[12]: # 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):
        """
        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}})
        else:
            super().__init__(self.data_checks + [ClassImbalanceDataCheck],
                             data_check_params={"InvalidTargetDataCheck": {"problem_
↳type": problem_type}})

custom_data_checks = MyCustomDataChecks(problem_type=ProblemTypes.REGRESSION)
for data_check in custom_data_checks.data_checks:
    print(data_check.name)
```

```
HighlyNullDataCheck
InvalidTargetDataCheck
NoVarianceDataCheck
TargetLeakageDataCheck
```

```
[13]: # 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}})
for data_check in same_custom_data_checks.data_checks:
    print(data_check.name)
```

```
HighlyNullDataCheck
InvalidTargetDataCheck
```

(continues on next page)

(continued from previous page)

```
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: 0.17.0
EvalML installation directory: /home/docs/checkouts/readthedocs.org/user_builds/
➔feature-labs-inc-evalml/envs/v0.17.0/lib/python3.8/site-packages/evalml

SYSTEM INFO
-----
python: 3.8.6.final.0
python-bits: 64
OS: Linux
OS-release: 4.15.0-1051-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: 6.1G

INSTALLED VERSIONS
-----
zict: 2.0.0
xgboost: 1.2.1
woodwork: 0.0.7
widgetsnbextension: 3.5.1
wheel: 0.36.2
webencodings: 0.5.1
wcwidth: 0.2.5
urllib3: 1.26.2
```

(continues on next page)

(continued from previous page)

```
traitlets: 5.0.5
tqdm: 4.55.0
tornado: 6.1
toolz: 0.11.1
threadpoolctl: 2.1.0
texttable: 1.6.3
testpath: 0.4.4
terminado: 0.9.1
tblib: 1.7.0
statsmodels: 0.12.1
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: 3.4.1
sphinx-rtd-theme: 0.4.3
sortedcontainers: 2.3.0
snowballstemmer: 2.0.0
slicer: 0.0.3
six: 1.15.0
shap: 0.37.0
setuptools: 51.1.1
send2trash: 1.5.0
seaborn: 0.11.1
scipy: 1.5.4
scikit-optimize: 0.8.1
scikit-learn: 0.23.2
retrying: 1.3.3
requirements-parser: 0.2.0
requests: 2.25.1
regex: 2020.11.13
recommonmark: 0.5.0
readthedocs-sphinx-ext: 2.1.1
pyzmq: 20.0.0
pyyaml: 5.3.1
pytz: 2020.5
python-dateutil: 2.8.1
pysistent: 0.17.3
pyparsing: 2.4.7
pygments: 2.7.3
pydata-sphinx-theme: 0.4.1
pycuparser: 2.20
pyarrow: 2.0.0
pyaml: 20.4.0
ptyprocess: 0.7.0
psutil: 5.8.0
prompt-toolkit: 3.0.8
prometheus-client: 0.9.0
plotly: 4.14.1
pip: 20.3.3
pillow: 8.0.1
pickleshare: 0.7.5
pexpect: 4.8.0
patsy: 0.5.1
```

(continues on next page)

(continued from previous page)

```
partd: 1.1.0
parso: 0.8.1
pandocfilters: 1.4.3
pandas: 1.1.5
packaging: 20.8
numpy: 1.19.4
numba: 0.52.0
notebook: 6.1.6
nltk: 3.5
nlp-primitives: 1.1.0
networkx: 2.5
nest-asyncio: 1.4.3
nbsphinx: 0.8.0
nbformat: 5.0.8
nbconvert: 6.0.7
nbclient: 0.5.1
msgpack: 1.0.2
mock: 1.0.1
mistune: 0.8.4
matplotlib: 3.3.3
markupsafe: 1.1.1
loket: 0.2.0
llvmlite: 0.35.0
lightgbm: 3.1.1
kiwisolver: 1.3.1
kaleido: 0.1.0
jupyterlab-widgets: 1.0.0
jupyterlab-pygments: 0.1.2
jupyter-core: 4.7.0
jupyter-client: 6.1.7
jsonschema: 3.2.0
joblib: 1.0.0
jinja2: 2.11.2
jedi: 0.18.0
ipywidgets: 7.6.2
ipython: 7.19.0
ipython-genutils: 0.2.0
ipykernel: 5.4.2
imagesize: 1.2.0
idna: 2.10
heapdict: 1.0.1
graphviz: 0.15
future: 0.18.2
fsspec: 0.8.5
featuretools: 0.22.0
evalml: 0.17.0
entrypoints: 0.3
docutils: 0.14
distributed: 2020.12.0
defusedxml: 0.6.0
decorator: 4.4.2
dask: 2020.12.0
cycller: 0.10.0
commonmark: 0.8.1
colorama: 0.4.4
cloudpickle: 1.6.0
click: 7.1.2
```

(continues on next page)

(continued from previous page)

```
chardet: 4.0.0
cffi: 1.14.4
certifi: 2020.12.5
category-encoders: 2.2.2
catboost: 0.24.4
bleach: 3.2.1
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 and target

Return type `pd.DataFrame`, `pd.Series`

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`) – Data to transform
- **y** (`pd.Series`) – Target data

Returns Transformed X (and y, if passed in) with rows that had a NaN value removed.

Return type `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_state=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.problem_type.all_problem_types` for a full list.
- **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.
- **test_size** (*float*) – What percentage of data points should be included in the test set. Defaults to 0.2 (20%).
- **random_state** (*int*, *np.random.RandomState*) – Seed for the random number generator

Returns Feature and target data each split into train and test sets

Return type *ww.DataTable*, *ww.DataTable*, *ww.DataColumn*, *ww.DataColumn*

5.3 AutoML

5.3.1 AutoML Search Classes

AutoMLSearch

Automated Pipeline search.

evalml.automl.AutoMLSearch

class `evalml.automl.AutoMLSearch(X_train=None, y_train=None, problem_type=None, objective='auto', max_iterations=None, max_time=None, patience=None, tolerance=None, data_splitter=None, allowed_pipelines=None, allowed_model_families=None, start_iteration_callback=None, add_result_callback=None, error_callback=None, additional_objectives=None, random_state=0, n_jobs=-1, tuner_class=None, verbose=True, optimize_thresholds=False, ensembling=False, max_batches=None, problem_configuration=None, train_best_pipeline=True, _pipelines_per_batch=5)`

Automated Pipeline search.

Methods

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

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_state=0, n_jobs=-1, tuner_class=None, verbose=True, optimize_thresholds=False, ensembling=False, max_batches=None, problem_configuration=None, train_best_pipeline=True, _pipelines_per_batch=5)

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.
- **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_state** (*int, np.random.RandomState*) – The random seed/state. 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.
- **verbose** (*boolean*) – If True, turn verbosity on. Defaults to True.
- **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

- **`_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.

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*, *random_state=0*)

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
- **`random_state`** (*int*, *np.random.RandomState*) – The random seed/state. Defaults to 0.

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.search

`AutoMLSearch.search(data_checks='auto', show_iteration_plot=True)`

Find the best pipeline for the data set.

Parameters

- **data_checks** (*DataChecks, list(Datacheck), str, None*) – A collection of data checks to run before automl search. If data checks produce any errors, an exception will be thrown before the search begins. If “disabled” or None, no data checks will be done. If set to “auto”, DefaultDataChecks will be done. Default value is set to “auto”.
- **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.

Attributes

<code>best_pipeline</code>	Returns a trained instance of the best pipeline and parameters found during automl search.
<code>data_check_results</code>	If there are data checks, return any error messages that are found
<code>full_rankings</code>	Returns a pandas.DataFrame with scoring results from all pipelines searched
<code>has_searched</code>	Returns <i>True</i> if search has been ran and <i>False</i> if not
<code>rankings</code>	Returns a pandas.DataFrame with scoring results from the highest-scoring set of parameters used with each pipeline.
<code>results</code>	Class that allows access to a copy of the results from <i>automl_search</i> .

Class Inheritance

evalml.automl.automl_search.AutoMLSearch

5.3.2 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_state=0)`

Given the training data and ML problem parameters, compute a data splitting method to use during AutoML search.

Parameters

- **X** (*pd.DataFrame*, *ww.DataTable*) – The input training data of shape [n_samples, n_features].
- **y** (*pd.Series*, *ww.DataColumn*) – 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.
- **n_splits** (*int*, *None*) – the number of CV splits, if applicable. Default 3.
- **shuffle** (*bool*) – whether or not to shuffle the data before splitting, if applicable. Default True.

- **random_state** (*int*, *np.random.RandomState*) – The random seed/state. Defaults to 0.

Returns data splitting method.

Return type `sklearn.model_selection.BaseCrossValidator`

5.3.3 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_state=0)
```

Base class for the automl algorithms which power evalml.

Methods

<i>__init__</i>	This class represents an automated machine learning (AutoML) algorithm.
<i>add_result</i>	Register results from evaluating a pipeline
<i>next_batch</i>	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_state=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_state** (*int*, *np.random.RandomState*) – The random seed/state. Defaults to 0.

evalml.automl.automl_algorithm.AutoMLAlgorithm.add_result

`AutoMLAlgorithm.add_result(score_to_minimize, pipeline)`

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.

evalml.automl.automl_algorithm.AutoMLAlgorithm.next_batch

abstract `AutoMLAlgorithm.next_batch()`

Get the next batch of pipelines to evaluate

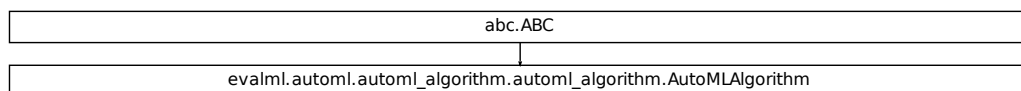
Returns a list of instances of PipelineBase subclasses, ready to be trained and evaluated.

Return type list(*PipelineBase*)

Attributes

<code>batch_number</code>	Returns the number of batches which have been recommended so far.
<code>pipeline_number</code>	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,
                                                         text_columns=None,
                                                         random_state=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, text_columns=None, random_state=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_state** (*int, np.random.RandomState*) – The random seed/state. 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*)

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.

evalml.automl.automl_algorithm.IterativeAlgorithm.next_batch

IterativeAlgorithm.**next_batch**()

Get the next batch of pipelines to evaluate

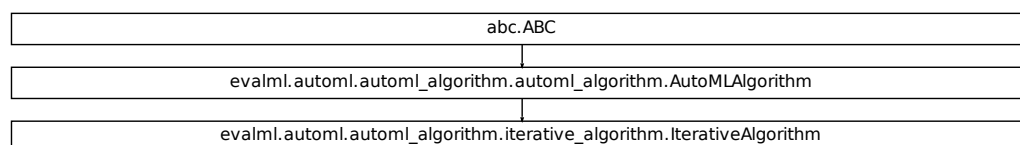
Returns a list of instances of PipelineBase subclasses, ready to be trained and evaluated.

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



5.3.4 AutoML Callbacks

<code>silent_error_callback</code>	No-op.
<code>log_error_callback</code>	Logs the exception thrown as an error.
<code>raise_error_callback</code>	Raises the exception thrown by the AutoMLSearch object.
<code>log_and_save_error_callback</code>	Logs the exception thrown by the AutoMLSearch object as a warning and adds the exception to the ‘errors’ list in AutoMLSearch object results.
<code>raise_and_save_error_callback</code>	Raises the exception thrown by the AutoMLSearch object, logs it as an error, and adds the exception to the ‘errors’ list in AutoMLSearch object results.

`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.

`evalml.automl.callbacks.log_and_save_error_callback`

`evalml.automl.callbacks.log_and_save_error_callback(exception, traceback, automl, **kwargs)`
Logs the exception thrown by the AutoMLSearch object as a warning and adds the exception to the ‘errors’ list in AutoMLSearch object results.

`evalml.automl.callbacks.raise_and_save_error_callback`

`evalml.automl.callbacks.raise_and_save_error_callback(exception, traceback, automl, **kwargs)`
Raises the exception thrown by the AutoMLSearch object, logs it as an error, and adds the exception to the ‘errors’ list in AutoMLSearch object results.

5.4 Pipelines

5.4.1 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_state=0*)

Base class for all pipelines.

name = 'Pipeline Base'

custom_name = None

problem_type = None

Instance attributes

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

Methods:

<i>__init__</i>	Machine learning pipeline made out of transformers and a estimator.
<i>clone</i>	Constructs a new pipeline with the same parameters and components.
<i>compute_estimator_features</i>	Transforms the data by applying all pre-processing components.
<i>describe</i>	Outputs pipeline details including component parameters
<i>fit</i>	Build a model
<i>get_component</i>	Returns component by name
<i>graph</i>	Generate an image representing the pipeline graph
<i>graph_feature_importance</i>	Generate a bar graph of the pipeline's feature importance
<i>load</i>	Loads pipeline at file path

continues on next page

Table 15 – continued from previous page

<i>predict</i>	Make predictions using selected features.
<i>save</i>	Saves pipeline at file path
<i>score</i>	Evaluate model performance on current and additional objectives

`evalml.pipelines.PipelineBase.__init__`

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

Machine learning pipeline made out of transformers and a estimator.

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*, `np.random.RandomState`) – The random seed/state. Defaults to 0.

`evalml.pipelines.PipelineBase.clone`

`PipelineBase.clone(random_state=0)`

Constructs a new pipeline with the same parameters and components.

Parameters `random_state` (*int*) – the value to seed the random state with. Can also be a `RandomState` instance. Defaults to 0.

Returns A new instance of this pipeline with identical parameters and components

`evalml.pipelines.PipelineBase.compute_estimator_features`

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

Transforms the data by applying all pre-processing components.

Parameters `X` (`pd.DataFrame`) – Input data to the pipeline to transform.

Returns `pd.DataFrame` - New transformed features.

`evalml.pipelines.PipelineBase.describe`

`PipelineBase.describe()`

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** pd.Series**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*)

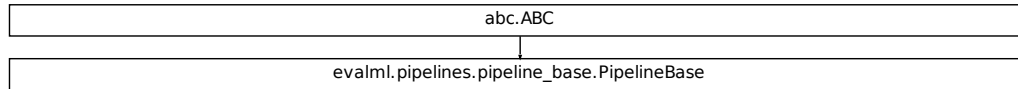
Evaluate model performance on current and additional objectives

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_state=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>clone</code>	Constructs a new pipeline with the same parameters and components.
<code>compute_estimator_features</code>	Transforms the data by applying all pre-processing components.
<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

continues on next page

Table 17 – continued from previous page

<i>score</i>	Evaluate model performance on objectives
--------------	--

evalml.pipelines.ClassificationPipeline.__init__

`ClassificationPipeline.__init__(parameters, random_state=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*, `np.random.RandomState`) – The random seed/state. Defaults to 0.

evalml.pipelines.ClassificationPipeline.clone

`ClassificationPipeline.clone(random_state=0)`

Constructs a new pipeline with the same parameters and components.

Parameters `random_state` (*int*) – the value to seed the random state with. Can also be a `RandomState` instance. Defaults to 0.

Returns A new instance of this pipeline with identical parameters and components

evalml.pipelines.ClassificationPipeline.compute_estimator_features

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

Transforms the data by applying all pre-processing components.

Parameters `X` (`pd.DataFrame`) – Input data to the pipeline to transform.

Returns `pd.DataFrame` - New transformed features.

evalml.pipelines.ClassificationPipeline.describe

`ClassificationPipeline.describe()`

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** pd.Series**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** pd.DataFrame**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*)

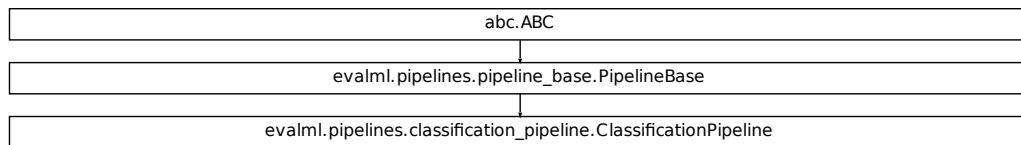
Evaluate model performance on objectives

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_state=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>clone</code>	Constructs a new pipeline with the same parameters and components.
<code>compute_estimator_features</code>	Transforms the data by applying all pre-processing components.
<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.BinaryClassificationPipeline.__init__

`BinaryClassificationPipeline.__init__(parameters, random_state=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*, *np.random.RandomState*) – The random seed/state. Defaults to 0.

evalml.pipelines.BinaryClassificationPipeline.clone

`BinaryClassificationPipeline.clone(random_state=0)`

Constructs a new pipeline with the same parameters and components.

Parameters **random_state** (*int*) – the value to seed the random state with. Can also be a `RandomState` instance. Defaults to 0.

Returns A new instance of this pipeline with identical parameters and components

evalml.pipelines.BinaryClassificationPipeline.compute_estimator_features

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

Transforms the data by applying all pre-processing components.

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

Returns *pd.DataFrame* - New transformed features.

evalml.pipelines.BinaryClassificationPipeline.describe

BinaryClassificationPipeline.**describe**()

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.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 `pd.Series`

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* (*pd.DataFrame* or *np.ndarray*) – Data of shape [n_samples, n_features]

Returns probability estimates

Return type *pd.DataFrame*

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

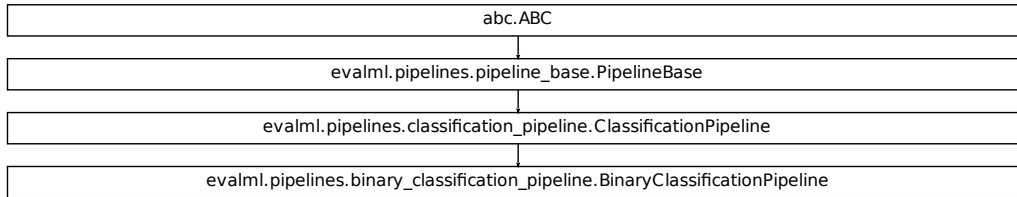
Parameters

- **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_state=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>clone</code>	Constructs a new pipeline with the same parameters and components.
<code>compute_estimator_features</code>	Transforms the data by applying all pre-processing components.
<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 21 – 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.MulticlassClassificationPipeline.__init__`

`MulticlassClassificationPipeline.__init__(parameters, random_state=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*, *np.random.RandomState*) – The random seed/state. Defaults to 0.

`evalml.pipelines.MulticlassClassificationPipeline.clone`

`MulticlassClassificationPipeline.clone(random_state=0)`

Constructs a new pipeline with the same parameters and components.

Parameters `random_state` (*int*) – the value to seed the random state with. Can also be a `RandomState` instance. Defaults to 0.

Returns A new instance of this pipeline with identical parameters and components

`evalml.pipelines.MulticlassClassificationPipeline.compute_estimator_features`

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

Transforms the data by applying all pre-processing components.

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

Returns `pd.DataFrame` - New transformed features.

`evalml.pipelines.MulticlassClassificationPipeline.describe`

`MulticlassClassificationPipeline.describe()`

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 pd.Series

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 pd.DataFrame

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

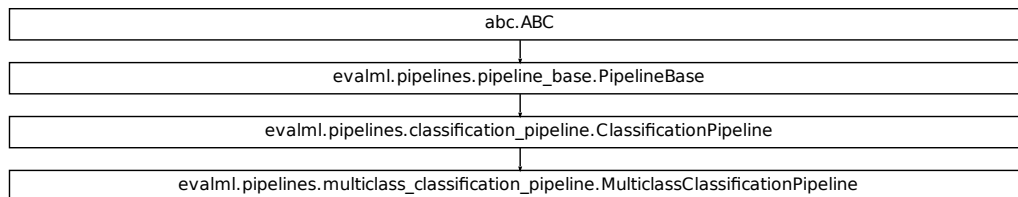
Parameters

- **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_state=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>clone</code>	Constructs a new pipeline with the same parameters and components.
<code>compute_estimator_features</code>	Transforms the data by applying all pre-processing components.
<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_state=0)`
 Machine learning pipeline made out of transformers and a estimator.

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*, `np.random.RandomState`) – The random seed/state. Defaults to 0.

`evalml.pipelines.RegressionPipeline.clone`

`RegressionPipeline.clone(random_state=0)`

Constructs a new pipeline with the same parameters and components.

Parameters `random_state` (*int*) – the value to seed the random state with. Can also be a `RandomState` instance. Defaults to 0.

Returns A new instance of this pipeline with identical parameters and components

`evalml.pipelines.RegressionPipeline.compute_estimator_features`

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

Transforms the data by applying all pre-processing components.

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

Returns `pd.DataFrame` - New transformed features.

`evalml.pipelines.RegressionPipeline.describe`

`RegressionPipeline.describe()`

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 `pd.Series`

`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

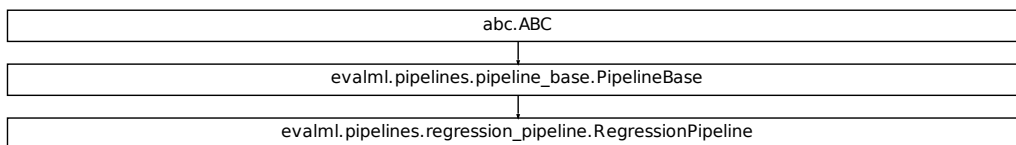
Parameters

- **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_state=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>clone</code>	Constructs a new pipeline with the same parameters and components.
<code>compute_estimator_features</code>	Transforms the data by applying all pre-processing components.
<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_state=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_state** (*int, np.random.RandomState*) – The random seed/state. Defaults to 0.

evalml.pipelines.TimeSeriesClassificationPipeline.clone

TimeSeriesClassificationPipeline.**clone** (*random_state=0*)

Constructs a new pipeline with the same parameters and components.

Parameters **random_state** (*int*) – the value to seed the random state with. Can also be a RandomState instance. Defaults to 0.

Returns A new instance of this pipeline with identical parameters and components

evalml.pipelines.TimeSeriesClassificationPipeline.compute_estimator_features

TimeSeriesClassificationPipeline.**compute_estimator_features** (*X, y=None*)

Transforms the data by applying all pre-processing components.

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

Returns pd.DataFrame - New transformed features.

evalml.pipelines.TimeSeriesClassificationPipeline.describe

TimeSeriesClassificationPipeline.**describe** ()

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 `pd.Series`

`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 `pd.DataFrame`

`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.

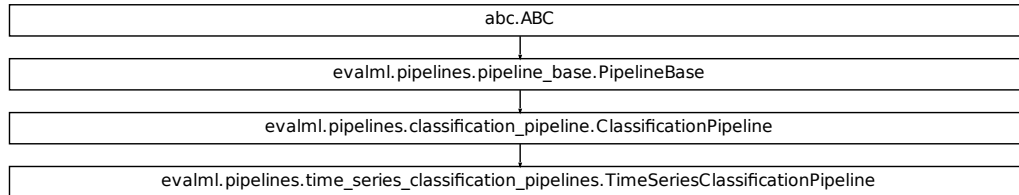
Parameters

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



evalml.pipelines.TimeSeriesBinaryClassificationPipeline

class evalml.pipelines.TimeSeriesBinaryClassificationPipeline(*parameters, random_state=0*)

name = 'Time Series Binary Classification Pipeline'

custom_name = None

problem_type = 'time series 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>	

Methods:

<code>__init__</code>	Machine learning pipeline for time series classification problems made out of transformers and a classifier.
<code>clone</code>	Constructs a new pipeline with the same parameters and components.
<code>compute_estimator_features</code>	Transforms the data by applying all pre-processing components.
<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

continues on next page

Table 27 – continued from previous page

<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.TimeSeriesBinaryClassificationPipeline.__init__`

`TimeSeriesBinaryClassificationPipeline.__init__` (*parameters*, *random_state=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_state** (*int*, *np.random.RandomState*) – The random seed/state. Defaults to 0.

`evalml.pipelines.TimeSeriesBinaryClassificationPipeline.clone`

`TimeSeriesBinaryClassificationPipeline.clone` (*random_state=0*)

Constructs a new pipeline with the same parameters and components.

Parameters `random_state` (*int*) – the value to seed the random state with. Can also be a `RandomState` instance. Defaults to 0.

Returns A new instance of this pipeline with identical parameters and components

`evalml.pipelines.TimeSeriesBinaryClassificationPipeline.compute_estimator_features`

`TimeSeriesBinaryClassificationPipeline.compute_estimator_features` (*X*, *y=None*)

Transforms the data by applying all pre-processing components.

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

Returns `pd.DataFrame` - New transformed features.

evalml.pipelines.TimeSeriesBinaryClassificationPipeline.describe

`TimeSeriesBinaryClassificationPipeline.describe()`

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.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 `pd.Series`

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 `pd.DataFrame`

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*)

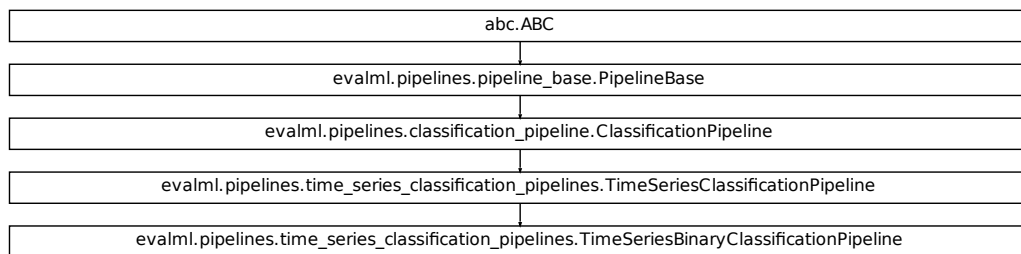
Evaluate model performance on current and additional objectives.

Parameters

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

evalml.pipelines.TimeSeriesMulticlassClassificationPipeline

```
class evalml.pipelines.TimeSeriesMulticlassClassificationPipeline(parameters,  
                                                                ran-  
                                                                dom_state=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>clone</code>	Constructs a new pipeline with the same parameters and components.
<code>compute_estimator_features</code>	Transforms the data by applying all pre-processing components.
<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_state=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_state** (*int*, *np.random.RandomState*) – The random seed/state. Defaults to 0.

evalml.pipelines.TimeSeriesMulticlassClassificationPipeline.clone

`TimeSeriesMulticlassClassificationPipeline.clone(random_state=0)`

Constructs a new pipeline with the same parameters and components.

Parameters `random_state` (*int*) – the value to seed the random state with. Can also be a `RandomState` instance. Defaults to 0.

Returns A new instance of this pipeline with identical parameters and components

evalml.pipelines.TimeSeriesMulticlassClassificationPipeline.compute_estimator_features

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

Transforms the data by applying all pre-processing components.

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

Returns `pd.DataFrame` - New transformed features.

evalml.pipelines.TimeSeriesMulticlassClassificationPipeline.describe

`TimeSeriesMulticlassClassificationPipeline.describe()`

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 *pd.Series*

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 *pd.DataFrame*

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.

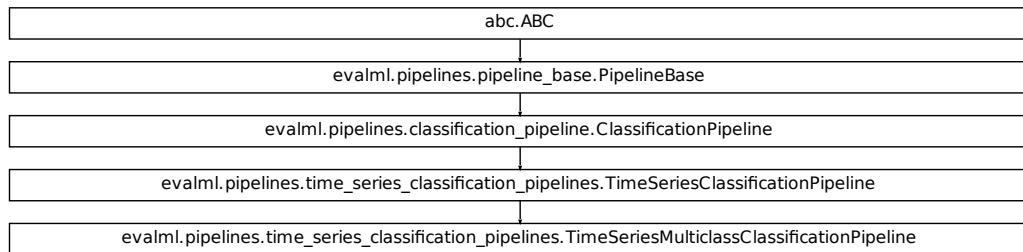
Parameters

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



evalml.pipelines.TimeSeriesRegressionPipeline

class `evalml.pipelines.TimeSeriesRegressionPipeline(parameters, random_state=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>clone</code>	Constructs a new pipeline with the same parameters and components.
<code>compute_estimator_features</code>	Transforms the data by applying all pre-processing components.
<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_state=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_state** (*int*, *np.random.RandomState*) – The random seed/state. Defaults to 0.

evalml.pipelines.TimeSeriesRegressionPipeline.clone

`TimeSeriesRegressionPipeline.clone(random_state=0)`

Constructs a new pipeline with the same parameters and components.

Parameters `random_state` (*int*) – the value to seed the random state with. Can also be a `RandomState` instance. Defaults to 0.

Returns A new instance of this pipeline with identical parameters and components

evalml.pipelines.TimeSeriesRegressionPipeline.compute_estimator_features

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

Transforms the data by applying all pre-processing components.

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

Returns `pd.DataFrame` - New transformed features.

evalml.pipelines.TimeSeriesRegressionPipeline.describe

`TimeSeriesRegressionPipeline.describe()`

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 `pd.Series`

`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.

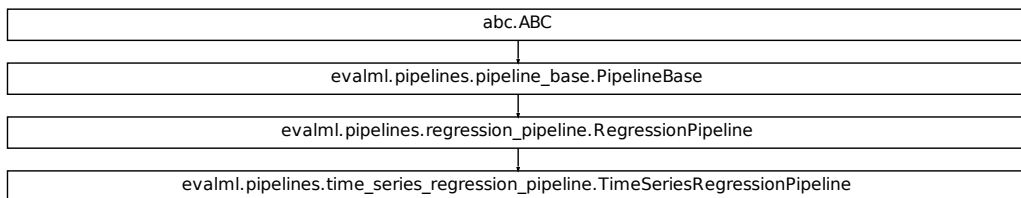
Parameters

- **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



5.4.2 Classification Pipelines

<i>BaselineBinaryPipeline</i>	Baseline Pipeline for binary classification.
<i>BaselineMulticlassPipeline</i>	Baseline Pipeline for multiclass classification.
<i>ModeBaselineBinaryPipeline</i>	Mode Baseline Pipeline for binary classification.
<i>ModeBaselineMulticlassPipeline</i>	Mode Baseline Pipeline for multiclass classification.

evalml.pipelines.BaselineBinaryPipeline

```
class evalml.pipelines.BaselineBinaryPipeline (parameters, random_state=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

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

Methods:

<i>__init__</i>	Machine learning classification pipeline made out of transformers and a classifier.
<i>clone</i>	Constructs a new pipeline with the same parameters and components.
<i>compute_estimator_features</i>	Transforms the data by applying all pre-processing components.
<i>describe</i>	Outputs pipeline details including component parameters
<i>fit</i>	Build a classification model. For string and categorical targets, classes are sorted
<i>get_component</i>	Returns component by name
<i>graph</i>	Generate an image representing the pipeline graph

continues on next page

Table 34 – continued from previous page

<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.BaselineBinaryPipeline.__init__`

`BaselineBinaryPipeline.__init__(parameters, random_state=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*, `np.random.RandomState`) – The random seed/state. Defaults to 0.

`evalml.pipelines.BaselineBinaryPipeline.clone`

`BaselineBinaryPipeline.clone(random_state=0)`

Constructs a new pipeline with the same parameters and components.

Parameters **random_state** (*int*) – the value to seed the random state with. Can also be a `RandomState` instance. Defaults to 0.

Returns A new instance of this pipeline with identical parameters and components

`evalml.pipelines.BaselineBinaryPipeline.compute_estimator_features`

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

Transforms the data by applying all pre-processing components.

Parameters **x** (`pd.DataFrame`) – Input data to the pipeline to transform.

Returns `pd.DataFrame` - New transformed features.

evalml.pipelines.BaselineBinaryPipeline.describe

BaselineBinaryPipeline.**describe**()

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.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 `pd.Series`

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

Returns probability estimates

Return type `pd.DataFrame`

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)`

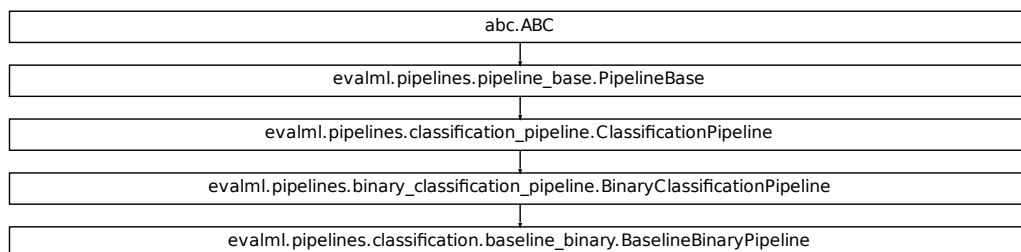
Evaluate model performance on objectives

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_state=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>clone</code>	Constructs a new pipeline with the same parameters and components.
<code>compute_estimator_features</code>	Transforms the data by applying all pre-processing components.
<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.BaselineMulticlassPipeline.__init__

`BaselineMulticlassPipeline.__init__(parameters, random_state=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*, *np.random.RandomState*) – The random seed/state. Defaults to 0.

evalml.pipelines.BaselineMulticlassPipeline.clone

`BaselineMulticlassPipeline.clone(random_state=0)`

Constructs a new pipeline with the same parameters and components.

Parameters `random_state` (*int*) – the value to seed the random state with. Can also be a `RandomState` instance. Defaults to 0.

Returns A new instance of this pipeline with identical parameters and components

evalml.pipelines.BaselineMulticlassPipeline.compute_estimator_features

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

Transforms the data by applying all pre-processing components.

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

Returns `pd.DataFrame` - New transformed features.

evalml.pipelines.BaselineMulticlassPipeline.describe

`BaselineMulticlassPipeline.describe()`

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 `pd.Series`

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 `pd.DataFrame`

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

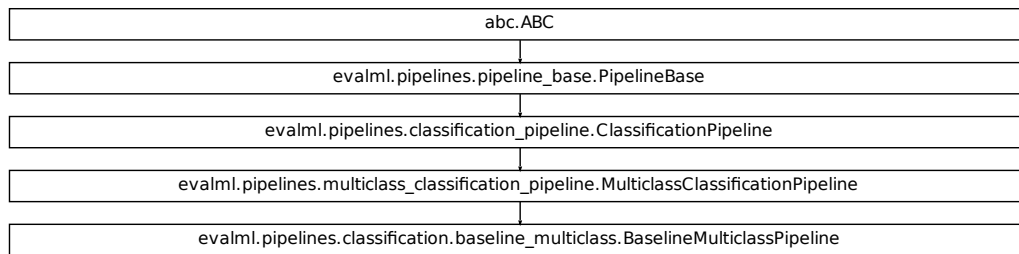
Parameters

- **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_state=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>clone</code>	Constructs a new pipeline with the same parameters and components.
<code>compute_estimator_features</code>	Transforms the data by applying all pre-processing components.
<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.ModeBaselineBinaryPipeline.__init__

`ModeBaselineBinaryPipeline.__init__(parameters, random_state=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*, *np.random.RandomState*) – The random seed/state. Defaults to 0.

`evalml.pipelines.ModeBaselineBinaryPipeline.clone`

`ModeBaselineBinaryPipeline.clone (random_state=0)`

Constructs a new pipeline with the same parameters and components.

Parameters `random_state` (*int*) – the value to seed the random state with. Can also be a `RandomState` instance. Defaults to 0.

Returns A new instance of this pipeline with identical parameters and components

`evalml.pipelines.ModeBaselineBinaryPipeline.compute_estimator_features`

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

Transforms the data by applying all pre-processing components.

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

Returns `pd.DataFrame` - New transformed features.

`evalml.pipelines.ModeBaselineBinaryPipeline.describe`

`ModeBaselineBinaryPipeline.describe ()`

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.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 `pd.Series`

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

Returns probability estimates

Return type `pd.DataFrame`

`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

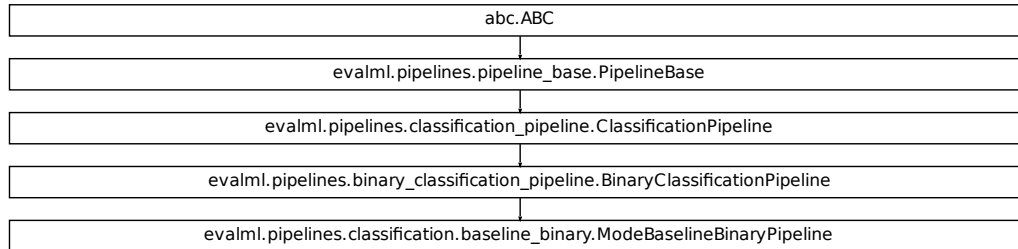
Parameters

- **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_state=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

<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>clone</code>	Constructs a new pipeline with the same parameters and components.
<code>compute_estimator_features</code>	Transforms the data by applying all pre-processing components.
<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_state=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*, *np.random.RandomState*) – The random seed/state. Defaults to 0.

evalml.pipelines.ModeBaselineMulticlassPipeline.clone

ModeBaselineMulticlassPipeline.clone(random_state=0)

Constructs a new pipeline with the same parameters and components.

Parameters **random_state** (*int*) – the value to seed the random state with. Can also be a RandomState instance. Defaults to 0.

Returns A new instance of this pipeline with identical parameters and components

evalml.pipelines.ModeBaselineMulticlassPipeline.compute_estimator_features

ModeBaselineMulticlassPipeline.**compute_estimator_features**(*X*, *y=None*)

Transforms the data by applying all pre-processing components.

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

Returns *pd.DataFrame* - New transformed features.

evalml.pipelines.ModeBaselineMulticlassPipeline.describe

ModeBaselineMulticlassPipeline.**describe**()

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 `pd.Series`

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 *pd.DataFrame*

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

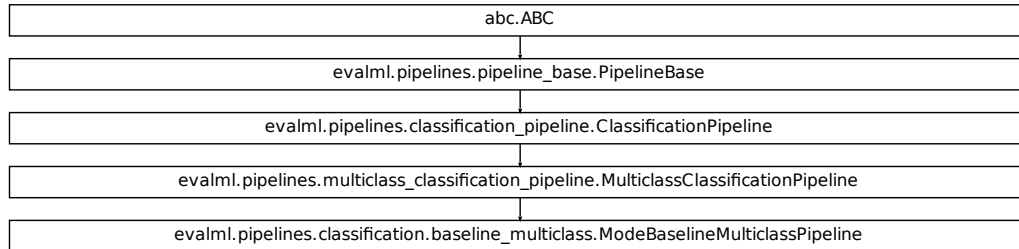
Parameters

- **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



5.4.3 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_state=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>clone</code>	Constructs a new pipeline with the same parameters and components.
<code>compute_estimator_features</code>	Transforms the data by applying all pre-processing components.
<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_state=0)`

Machine learning pipeline made out of transformers and a estimator.

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*, `np.random.RandomState`) – The random seed/state. Defaults to 0.

`evalml.pipelines.BaselineRegressionPipeline.clone`

`BaselineRegressionPipeline.clone(random_state=0)`

Constructs a new pipeline with the same parameters and components.

Parameters `random_state` (*int*) – the value to seed the random state with. Can also be a `RandomState` instance. Defaults to 0.

Returns A new instance of this pipeline with identical parameters and components

`evalml.pipelines.BaselineRegressionPipeline.compute_estimator_features`

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

Transforms the data by applying all pre-processing components.

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

Returns `pd.DataFrame` - New transformed features.

`evalml.pipelines.BaselineRegressionPipeline.describe`

`BaselineRegressionPipeline.describe()`

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 `pd.Series`

`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

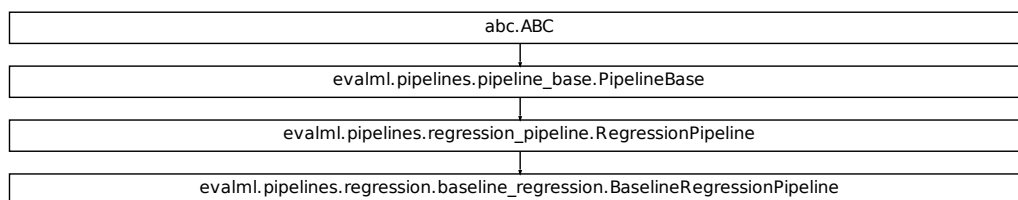
Parameters

- **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_state=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>clone</code>	Constructs a new pipeline with the same parameters and components.
<code>compute_estimator_features</code>	Transforms the data by applying all pre-processing components.
<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_state=0)`

Machine learning pipeline made out of transformers and a estimator.

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*, *np.random.RandomState*) – The random seed/state. Defaults to 0.

`evalml.pipelines.MeanBaselineRegressionPipeline.clone`

`MeanBaselineRegressionPipeline.clone(random_state=0)`

Constructs a new pipeline with the same parameters and components.

Parameters `random_state` (*int*) – the value to seed the random state with. Can also be a `RandomState` instance. Defaults to 0.

Returns A new instance of this pipeline with identical parameters and components

`evalml.pipelines.MeanBaselineRegressionPipeline.compute_estimator_features`

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

Transforms the data by applying all pre-processing components.

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

Returns `pd.DataFrame` - New transformed features.

`evalml.pipelines.MeanBaselineRegressionPipeline.describe`

`MeanBaselineRegressionPipeline.describe()`

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** pd.Series**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*)

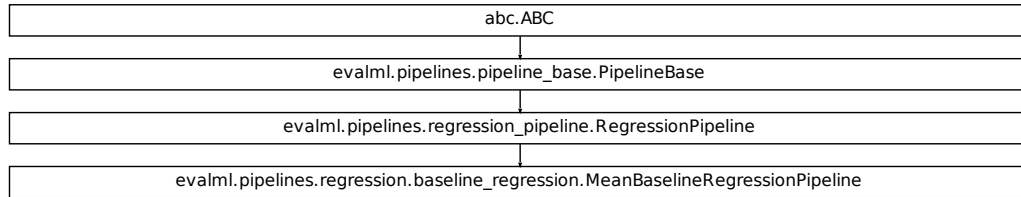
Evaluate model performance on current and additional objectives

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_state=0)
    Baseline Pipeline for time series regression problems.
    name = 'Time Series Baseline Regression Pipeline'
    custom_name = None
    summary = 'Time Series Baseline Regressor'
    component_graph = ['Time Series Baseline Regressor']
    problem_type = 'time series regression'
    model_family = 'baseline'
    hyperparameters = {'Time Series Baseline Regressor': {}}
    custom_hyperparameters = None
    default_parameters = {'Time Series Baseline Regressor': {'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>clone</code>	Constructs a new pipeline with the same parameters and components.
<code>compute_estimator_features</code>	Transforms the data by applying all pre-processing components.
<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_state=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_state** (*int*, *np.random.RandomState*) – The random seed/state. Defaults to 0.

evalml.pipelines.TimeSeriesBaselineRegressionPipeline.clone

`TimeSeriesBaselineRegressionPipeline.clone(random_state=0)`

Constructs a new pipeline with the same parameters and components.

Parameters **random_state** (*int*) – the value to seed the random state with. Can also be a `RandomState` instance. Defaults to 0.

Returns A new instance of this pipeline with identical parameters and components

evalml.pipelines.TimeSeriesBaselineRegressionPipeline.compute_estimator_features

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

Transforms the data by applying all pre-processing components.

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

Returns *pd.DataFrame* - New transformed features.

evalml.pipelines.TimeSeriesBaselineRegressionPipeline.describe

`TimeSeriesBaselineRegressionPipeline.describe()`

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, np.ndarray, or None*) – 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, objective=None*)

Make predictions using selected features.

Parameters

- **X** (*ww.DataTable, pd.DataFrame, np.ndarray, or None*) – Data of shape `[n_samples, n_features]`
- **y** (*ww.DataColumn, pd.Series, np.ndarray*) – The target training targets of length `[n_samples]`. `y` is required for this pipeline.
- **objective** (*Object or string*) – The objective to use to make predictions

Returns Predicted values.

Return type `pd.Series`

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*)

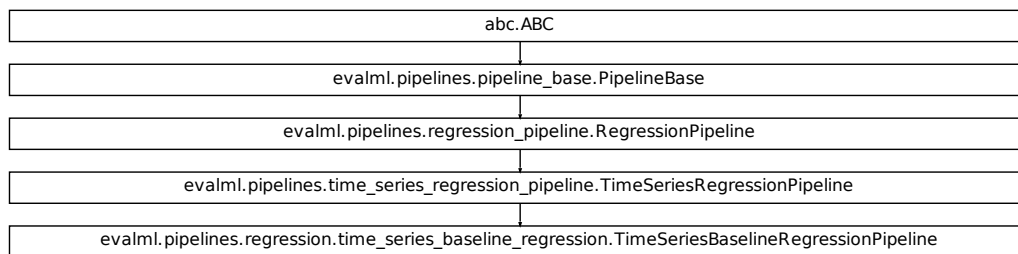
Evaluate model performance on current and additional objectives.

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

5.4.4 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*, *text_columns=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*) – Dictionary of custom hyperparameters, with component name as key and dictionary of parameters as the value
- **text_columns** – feature names which should be treated as text features. Defaults to None.

evalml.pipelines.utils.make_pipeline_from_components

`evalml.pipelines.utils.make_pipeline_from_components` (*component_instances*, *problem_type*, *custom_name=None*, *random_state=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_state** – Random state 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.

5.5 Components

5.5.1 Component Base Classes

Components represent a step in a pipeline.

<i>ComponentBase</i>	Base class for all components.
<i>Transformer</i>	A component that may or may not need fitting that transforms data.
<i>Estimator</i>	A component that fits and predicts given data.

evalml.pipelines.components.ComponentBase

class evalml.pipelines.components.ComponentBase(*parameters=None, component_obj=None, random_state=0, **kwargs*)

Base class for all components.

Methods

<i>__init__</i>	Initialize self.
<i>clone</i>	Constructs a new component with the same parameters
<i>describe</i>	Describe a component and its parameters
<i>fit</i>	Fits component to data
<i>load</i>	Loads component at file path
<i>save</i>	Saves component at file path

evalml.pipelines.components.ComponentBase.__init__

`ComponentBase.__init__ (parameters=None, component_obj=None, random_state=0, **kwargs)`

Initialize self. See help(type(self)) for accurate signature.

evalml.pipelines.components.ComponentBase.clone

`ComponentBase.clone (random_state=0)`

Constructs a new component with the same parameters

Parameters `random_state` (*int*, *RandomState*) – the value to seed the random state with. Can also be a *RandomState* instance. Defaults to 0.

Returns A new instance of this component with identical parameters

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** (*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.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

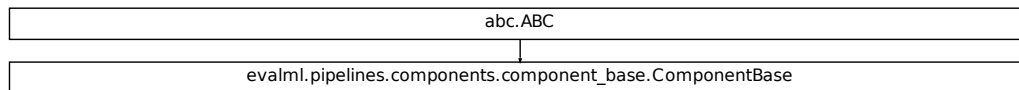
ComponentBase.save(*file_path*, *pickle_protocol*=5)

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_state=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
<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_state=0, **kwargs)`
Initialize self. See `help(type(self))` for accurate signature.

`evalml.pipelines.components.Transformer.clone`

`Transformer.clone(random_state=0)`
Constructs a new component with the same parameters

Parameters `random_state` (*int*, *RandomState*) – the value to seed the random state with. Can also be a *RandomState* instance. Defaults to 0.

Returns A new instance of this component with identical parameters

`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** (*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.Transformer.fit_transform**`Transformer.fit_transform(X, y=None)`

Fits on X and transforms X

Parameters

- **X** (*pd.DataFrame*) – Data to fit and transform
- **y** (*pd.DataFrame*) – Target data

Returns Transformed X**Return type** pd.DataFrame**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

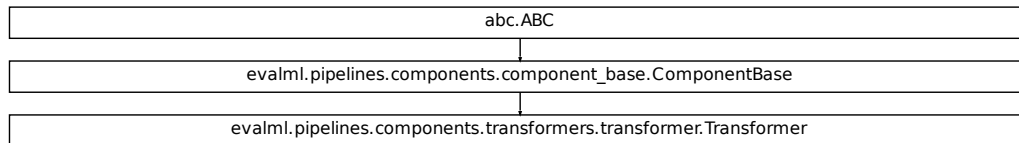
Parameters

- **X** (`pd.DataFrame`) – Data to transform
- **y** (`pd.Series`, *optional*) – Target data

Returns Transformed X

Return type `pd.DataFrame`

Class Inheritance



evalml.pipelines.components.Estimator

class `evalml.pipelines.components.Estimator` (*parameters=None, component_obj=None, random_state=0, **kwargs*)

A component that fits and predicts given data.

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 Estimator component.

Methods

<code>__init__</code>	Initialize self.
<code>clone</code>	Constructs a new component with the same parameters
<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 52 – 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_state=0, **kwargs*)
 Initialize self. See help(type(self)) for accurate signature.

evalml.pipelines.components.Estimator.clone

`Estimator.clone` (*random_state=0*)
 Constructs a new component with the same parameters

Parameters `random_state` (*int, RandomState*) – the value to seed the random state with. Can also be a RandomState instance. Defaults to 0.

Returns A new instance of this component with identical parameters

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** (*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.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** `pd.Series`**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** `pd.DataFrame`**evalml.pipelines.components.Estimator.save**`Estimator.save(file_path, pickle_protocol=5)`

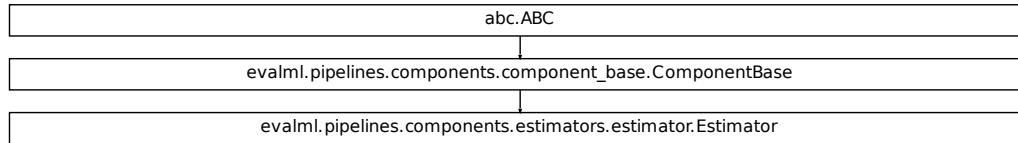
Saves component at file path

Parameters

- `file_path` (*str*) – Location to save file
- `pickle_protocol` (*int*) – The pickle data stream format.

Returns None

Class Inheritance



5.5.2 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.

5.5.3 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

evalml.pipelines.components.DropColumns

```
class evalml.pipelines.components.DropColumns (columns=None, random_state=0,  
                                              **kwargs)
```

Drops specified columns in input data.

```
name = 'Drop Columns Transformer'
```

```
model_family = 'none'
```

```
hyperparameter_ranges = {}
```

```
default_parameters = {'columns': None}
```

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 drops specified columns in input data.
<code>clone</code>	Constructs a new component with the same parameters
<code>describe</code>	Describe a component and its parameters
<code>fit</code>	'Fits' the transformer by checking if the column names are present in the dataset.
<code>fit_transform</code>	Fit transformer to data, then transform data.
<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.

evalml.pipelines.components.DropColumns.__init__

```
DropColumns.__init__ (columns=None, random_state=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(random_state=0)`

Constructs a new component with the same parameters

Parameters `random_state` (*int*, *RandomState*) – the value to seed the random state with. Can also be a *RandomState* instance. Defaults to 0.

Returns A new instance of this component with identical parameters

`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 the column names are present in the dataset.

Parameters

- `X` (*pd.DataFrame*) – Data to check.
- `y` (*pd.Series*, *optional*) – Targets.

Returns None.

`evalml.pipelines.components.DropColumns.fit_transform`

`DropColumns.fit_transform(X, y=None)`

Fit transformer to data, then transform data.

Parameters

- `X` (*pd.DataFrame*) – Data to transform.
- `y` (*pd.Series*, *optional*) – Targets.

Returns Transformed X.

Return type *pd.DataFrame*

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

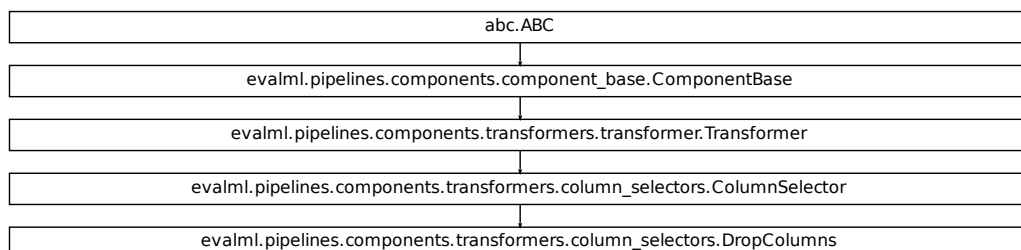
- **file_path** (*str*) – Location to save file
- **pickle_protocol** (*int*) – The pickle data stream format.

Returns None**evalml.pipelines.components.DropColumns.transform**DropColumns.**transform**(*X*, *y=None*)

Transforms data X by dropping columns.

Parameters

- **X** (*pd.DataFrame*) – Data to transform.
- **y** (*pd.Series*, *optional*) – Targets.

Returns Transformed X.**Return type** pd.DataFrame**Class Inheritance**

evalml.pipelines.components.SelectColumns

```
class evalml.pipelines.components.SelectColumns (columns=None, random_state=0,  
                                              **kwargs)
```

Selects specified columns in input data.

```
name = 'Select Columns Transformer'
```

```
model_family = 'none'
```

```
hyperparameter_ranges = {}
```

```
default_parameters = {'columns': None}
```

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 drops specified columns in input data.
<code>clone</code>	Constructs a new component with the same parameters
<code>describe</code>	Describe a component and its parameters
<code>fit</code>	'Fits' the transformer by checking if the column names are present in the dataset.
<code>fit_transform</code>	Fit transformer to data, then transform data.
<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_state=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` (*random_state=0*)

Constructs a new component with the same parameters

Parameters `random_state` (*int, RandomState*) – the value to seed the random state with. Can also be a `RandomState` instance. Defaults to 0.

Returns A new instance of this component with identical parameters

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 the column names are present in the dataset.

Parameters

- **X** (*pd.DataFrame*) – Data to check.
- **y** (*pd.Series, optional*) – Targets.

Returns None.

evalml.pipelines.components.SelectColumns.fit_transform

`SelectColumns.fit_transform` (*X, y=None*)

Fit transformer to data, then transform data.

Parameters

- **X** (*pd.DataFrame*) – Data to transform.
- **y** (*pd.Series, optional*) – Targets.

Returns Transformed X.

Return type `pd.DataFrame`

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

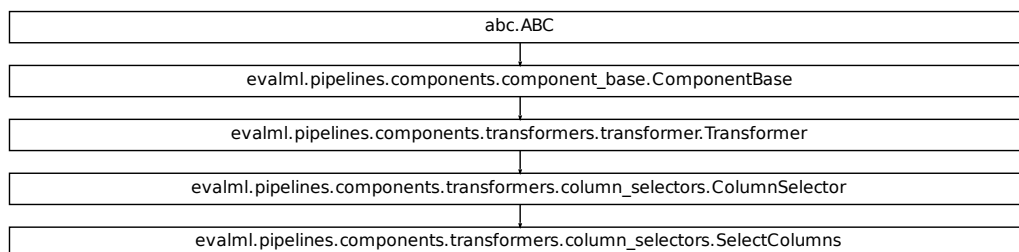
- `file_path` (*str*) – Location to save file
- `pickle_protocol` (*int*) – The pickle data stream format.

Returns None**evalml.pipelines.components.SelectColumns.transform**`SelectColumns.transform(X, y=None)`

Transforms data X by selecting columns.

Parameters

- `X` (*pd.DataFrame*) – Data to transform.
- `y` (*pd.Series, optional*) – Targets.

Returns Transformed X.**Return type** `pd.DataFrame`**Class Inheritance**

evalml.pipelines.components.OneHotEncoder

```
class evalml.pipelines.components.OneHotEncoder(top_n=10, features_to_encode=None,
                                                categories=None, drop=None,
                                                handle_unknown='ignore', handle_missing='error', random_state=0,
                                                **kwargs)
```

One-hot encoder to encode non-numeric data.

```
name = 'One Hot Encoder'
```

```
model_family = 'none'
```

```
hyperparameter_ranges = {}
```

```
default_parameters = {'categories': None, 'drop': None, 'features_to_encode': None, 'handle_unknown': 'ignore', 'handle_missing': 'error', 'random_state': 0, 'top_n': 10, **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 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
<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 DataFrame.

evalml.pipelines.components.OneHotEncoder.__init__

```
OneHotEncoder.__init__(top_n=10, features_to_encode=None, categories=None, drop=None,
                        handle_unknown='ignore', handle_missing='error', random_state=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 method to use for each feature. Defaults to *None*.
- **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”.

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 (random_state=0)`

Constructs a new component with the same parameters

Parameters **random_state** (*int, RandomState*) – the value to seed the random state with. Can also be a *RandomState* instance. Defaults to 0.

Returns A new instance of this component with identical parameters

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** (*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.OneHotEncoder.fit_transform**`OneHotEncoder.fit_transform(X, y=None)`

Fits on X and transforms X

Parameters

- **X** (*pd.DataFrame*) – Data to fit and transform
- **y** (*pd.DataFrame*) – Target data

Returns Transformed X**Return type** pd.DataFrame**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)`

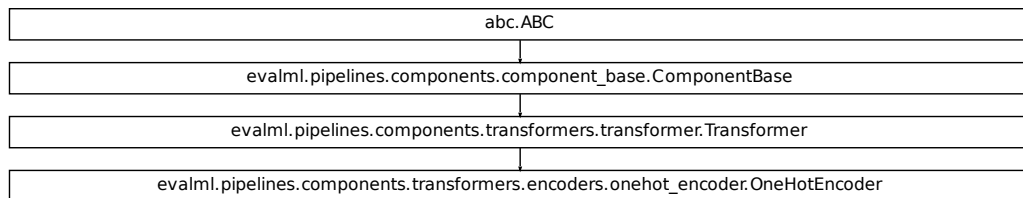
One-hot encode the input DataFrame.

Parameters

- **X** (*pd.DataFrame*) – Dataframe of features.
- **y** (*pd.Series*) – Ignored.

Returns Transformed dataframe, where each categorical feature has been encoded into numerical columns using one-hot encoding.

Class Inheritance



evalml.pipelines.components.TargetEncoder

```
class evalml.pipelines.components.TargetEncoder(cols=None, smoothing=1.0,
                                                handle_unknown='value', handle_missing='value', random_state=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'}
```

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
<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_state=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

`evalml.pipelines.components.TargetEncoder.clone`

`TargetEncoder.clone (random_state=0)`

Constructs a new component with the same parameters

Parameters `random_state` (*int*, *RandomState*) – the value to seed the random state with. Can also be a *RandomState* instance. Defaults to 0.

Returns A new instance of this component with identical parameters

`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** (*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.TargetEncoder.fit_transform`

`TargetEncoder.fit_transform (X, y)`

Fits on X and transforms X

Parameters

- **X** (*pd.DataFrame*) – Data to fit and transform
- **y** (*pd.DataFrame*) – Target data

Returns Transformed X

Return type *pd.DataFrame*

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

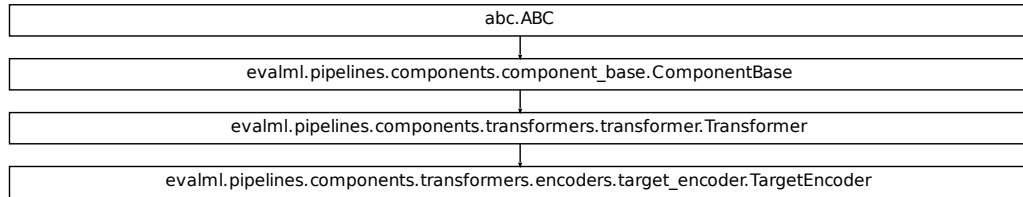
Parameters

- `X` (*pd.DataFrame*) – Data to transform
- `y` (*pd.Series, optional*) – Target data

Returns Transformed X

Return type pd.DataFrame

Class Inheritance



evalml.pipelines.components.PerColumnImputer

```
class evalml.pipelines.components.PerColumnImputer (impute_strategies=None,      de-  
fault_impute_strategy='most_frequent',  
random_state=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
<code>describe</code>	Describe a component and its parameters
<code>fit</code>	Fits imputers on input data
<code>fit_transform</code>	Fits imputer and imputes missing values in input data.
<code>load</code>	Loads component at file path
<code>save</code>	Saves component at file path

continues on next page

Table 64 – continued from previous page

<i>transform</i>	Transforms input data by imputing missing values.
------------------	---

evalml.pipelines.components.PerColumnImputer.__init__

`PerColumnImputer.__init__(impute_strategies=None, default_impute_strategy='most_frequent', random_state=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”

evalml.pipelines.components.PerColumnImputer.clone

`PerColumnImputer.clone(random_state=0)`

Constructs a new component with the same parameters

Parameters `random_state` (*int*, *RandomState*) – the value to seed the random state with. Can also be a *RandomState* instance. Defaults to 0.

Returns A new instance of this component with identical parameters

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 imputer and imputes missing values in input data.

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 `pd.DataFrame`

`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

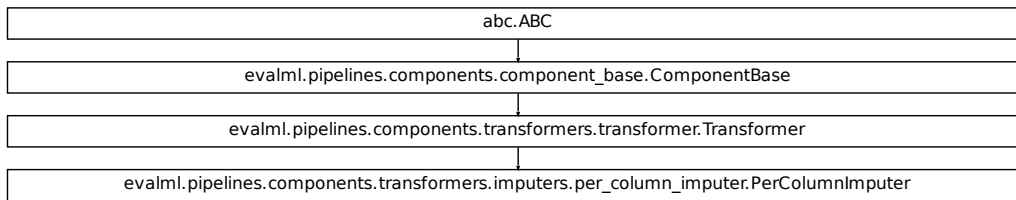
`PerColumnImputer.transform(X, y=None)`
 Transforms input data by imputing missing values.

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 `pd.DataFrame`

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_state=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
<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_state*=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.

evalml.pipelines.components.Imputer.clone

`Imputer.clone(random_state=0)`

Constructs a new component with the same parameters

Parameters `random_state` (*int*, *RandomState*) – the value to seed the random state with. Can also be a *RandomState* instance. Defaults to 0.

Returns A new instance of this component with identical parameters

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** (*pd.DataFrame* or *np.ndarray*) – The input training data of shape [n_samples, n_features]
- **y** (*pd.Series*, *optional*) – The target training data of length [n_samples]

Returns self

evalml.pipelines.components.Imputer.fit_transform

`Imputer.fit_transform(X, y=None)`

Fits on X and transforms X

Parameters

- **X** (*pd.DataFrame*) – Data to fit and transform
- **y** (*pd.DataFrame*) – Target data

Returns Transformed X

Return type *pd.DataFrame*

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.

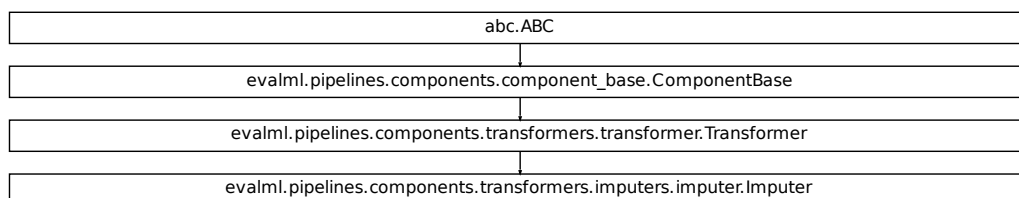
Parameters

- `X` (*pd.DataFrame*) – Data to transform
- `y` (*pd.Series, optional*) – Ignored.

Returns Transformed X

Return type `pd.DataFrame`

Class Inheritance



evalml.pipelines.components.SimpleImputer

```
class evalml.pipelines.components.SimpleImputer(impute_strategy='most_frequent',
                                                fill_value=None, random_state=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
<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.SimpleImputer.__init__

```
SimpleImputer.__init__(impute_strategy='most_frequent', fill_value=None, random_state=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.

evalml.pipelines.components.SimpleImputer.clone

`SimpleImputer.clone(random_state=0)`

Constructs a new component with the same parameters

Parameters `random_state` (*int*, *RandomState*) – the value to seed the random state with. Can also be a *RandomState* instance. Defaults to 0.

Returns A new instance of this component with identical parameters

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 `pd.DataFrame`

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*)

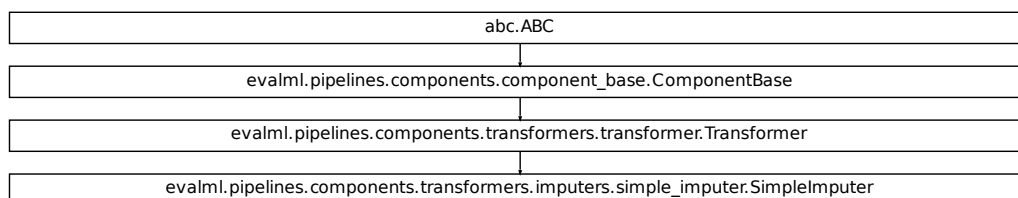
Transforms data *X* by imputing missing values. ‘None’ values are converted to np.nan before imputation and are treated as the same.

Parameters

- *X* (*ww.DataTable*, *pd.DataFrame*) – Data to transform
- *y* (*ww.DataColumn*, *pd.Series*, *optional*) – Ignored.

Returns Transformed *X*

Return type pd.DataFrame

Class Inheritance

evalml.pipelines.components.StandardScaler

class evalml.pipelines.components.**StandardScaler** (*random_state=0, **kwargs*)
Standardize features: removes mean and scales to unit variance.

```
name = 'Standard Scaler'
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>	Initialize self.
<code>clone</code>	Constructs a new component with the same parameters
<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.StandardScaler.__init__

`StandardScaler.__init__` (*random_state=0, **kwargs*)
Initialize self. See `help(type(self))` for accurate signature.

evalml.pipelines.components.StandardScaler.clone

`StandardScaler.clone` (*random_state=0*)
Constructs a new component with the same parameters

Parameters `random_state` (*int, RandomState*) – the value to seed the random state with. Can also be a `RandomState` instance. Defaults to 0.

Returns A new instance of this component with identical parameters

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** (*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.StandardScaler.fit_transform**`StandardScaler.fit_transform` (*X, y=None*)

Fits on X and transforms X

Parameters

- **X** (*pd.DataFrame*) – Data to fit and transform
- **y** (*pd.DataFrame*) – Target data

Returns Transformed X**Return type** pd.DataFrame**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)`

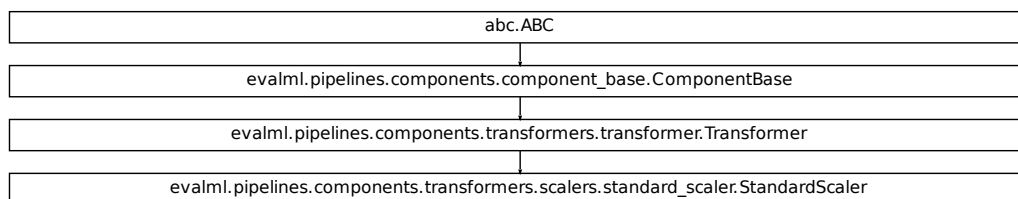
Transforms data X

Parameters

- **X** (*pd.DataFrame*) – Data to transform
- **y** (*pd.Series, optional*) – Target data

Returns Transformed X

Return type `pd.DataFrame`

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_state=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
<code>describe</code>	Describe a component and its parameters
<code>fit</code>	Fits component to data
<code>fit_transform</code>	Fits feature selector on data X then transforms X by selecting features
<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 data X 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_state=0,
                                     **kwargs)
```

Initialize self. See help(type(self)) for accurate signature.

evalml.pipelines.components.RFRegressorSelectFromModel.clone

```
RFRegressorSelectFromModel.clone(random_state=0)
```

Constructs a new component with the same parameters

Parameters **random_state** (*int*, *RandomState*) – the value to seed the random state with. Can also be a *RandomState* instance. Defaults to 0.

Returns A new instance of this component with identical parameters

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** (*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.RFRegressorSelectFromModel.fit_transform

`RFRegressorSelectFromModel.fit_transform(X, y=None)`

Fits feature selector on data X then transforms X by selecting features

Parameters

- **X** (`pd.DataFrame`) – Data to fit and transform
- **y** (`pd.Series`) – Target data

Returns Transformed X

Return type `pd.DataFrame`

evalml.pipelines.components.RFRegressorSelectFromModel.get_names

`RFRegressorSelectFromModel.get_names()`

Get names of selected features.

Returns list of the names of features selected

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 data X by selecting features

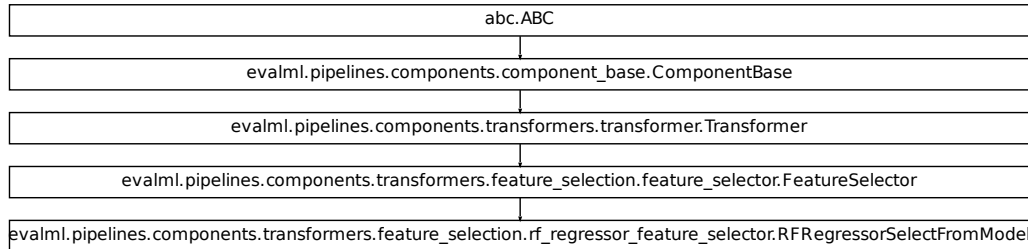
Parameters

- **X** (`pd.DataFrame`) – Data to transform
- **y** (`pd.Series, optional`) – Target data

Returns Transformed X

Return type `pd.DataFrame`

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_state=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

<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
<code>describe</code>	Describe a component and its parameters
<code>fit</code>	Fits component to data
<code>fit_transform</code>	Fits feature selector on data X then transforms X by selecting features
<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 data X 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_state=0*,
***kwargs*)

Initialize self. See help(type(self)) for accurate signature.

evalml.pipelines.components.RFClassifierSelectFromModel.clone

`RFClassifierSelectFromModel.clone` (*random_state=0*)

Constructs a new component with the same parameters

Parameters `random_state` (*int*, *RandomState*) – the value to seed the random state with. Can also be a *RandomState* instance. Defaults to 0.

Returns A new instance of this component with identical parameters

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** (*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.RFClassifierSelectFromModel.fit_transform

`RFClassifierSelectFromModel.fit_transform(X, y=None)`

Fits feature selector on data X then transforms X by selecting features

Parameters

- **X** (*pd.DataFrame*) – Data to fit and transform
- **y** (*pd.Series*) – Target data

Returns Transformed X

Return type *pd.DataFrame*

evalml.pipelines.components.RFClassifierSelectFromModel.get_names

`RFClassifierSelectFromModel.get_names()`

Get names of selected features.

Returns list of the names of features selected

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)`

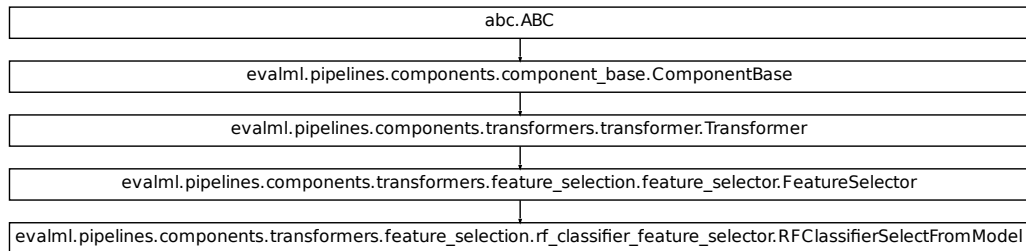
Transforms data X by selecting features

Parameters

- **X** (`pd.DataFrame`) – Data to transform
- **y** (`pd.Series`, *optional*) – Target data

Returns Transformed X

Return type `pd.DataFrame`

Class Inheritance**evalml.pipelines.components.DropNullColumns**

class `evalml.pipelines.components.DropNullColumns` (`pct_null_threshold=1.0`, *ran-*
`dom_state=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
<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_state=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.

`evalml.pipelines.components.DropNullColumns.clone`

`DropNullColumns.clone(random_state=0)`

Constructs a new component with the same parameters

Parameters `random_state` (*int*, *RandomState*) – the value to seed the random state with. Can also be a *RandomState* instance. Defaults to 0.

Returns A new instance of this component with identical parameters

`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** (*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.DropNullColumns.fit_transform`

`DropNullColumns.fit_transform(X, y=None)`

Fits on X and transforms X

Parameters

- **X** (*pd.DataFrame*) – Data to fit and transform
- **y** (*pd.DataFrame*) – Target data

Returns Transformed X

Return type *pd.DataFrame*

`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.

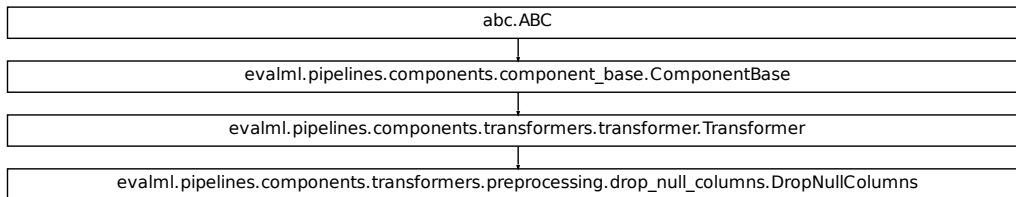
Parameters

- **X** (*pd.DataFrame*) – Data to transform
- **y** (*pd.Series, optional*) – Ignored.

Returns Transformed X

Return type *pd.DataFrame*

Class Inheritance



evalml.pipelines.components.DateTimeFeaturizer

```
class evalml.pipelines.components.DateTimeFeaturizer(features_to_extract=None,  
                                                    encode_as_categories=False,  
                                                    random_state=0, **kwargs)
```

Transformer that can automatically featurize Date/Time columns.

```
name = 'Date/Time Featurization Component'
```

```
model_family = 'none'
```

```
hyperparameter_ranges = {}
```

```
default_parameters = {'encode_as_categories': False, 'features_to_extract': ['year', 'month', 'day', 'hour', 'minute', 'second']}
```

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
<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_state=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_state** (*int*, *np.random.RandomState*) – Seed for the random number generator.

`evalml.pipelines.components.DateTimeFeaturizer.clone`

`DateTimeFeaturizer.clone(random_state=0)`

Constructs a new component with the same parameters

Parameters `random_state` (*int*, *RandomState*) – the value to seed the random state with. Can also be a *RandomState* instance. Defaults to 0.

Returns A new instance of this component with identical parameters

`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** (*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.DateTimeFeaturizer.fit_transform`

`DateTimeFeaturizer.fit_transform` (*X, y=None*)

Fits on X and transforms X

Parameters

- **X** (*pd.DataFrame*) – Data to fit and transform
- **y** (*pd.DataFrame*) – Target data

Returns Transformed X

Return type pd.DataFrame

`evalml.pipelines.components.DateTimeFeaturizer.get_feature_names`

`DateTimeFeaturizer.get_feature_names` ()

Gets the categories of each datetime feature.

Returns Dict. 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)`

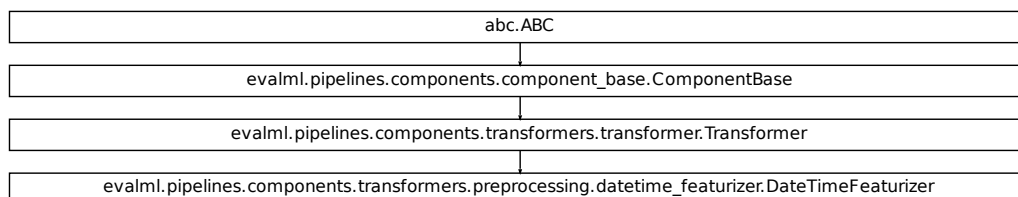
Transforms data X by creating new features using existing DateTime columns, and then dropping those DateTime columns

Parameters

- `X` (*pd.DataFrame*) – Data to transform
- `y` (*pd.Series, optional*) – Ignored.

Returns Transformed X

Return type *pd.DataFrame*

Class Inheritance

evalml.pipelines.components.TextFeaturizer

```
class evalml.pipelines.components.TextFeaturizer (text_columns=None, random_state=0, **kwargs)
```

Transformer that can automatically featurize text columns.

```
name = 'Text Featurization Component'
```

```
model_family = 'none'
```

```
hyperparameter_ranges = {}
```

```
default_parameters = {'text_columns': None}
```

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 featuretools' nlp_primitives
<code>clone</code>	Constructs a new component with the same parameters
<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__ (text_columns=None, random_state=0, **kwargs)
```

Extracts features from text columns using featuretools' nlp_primitives

Parameters

- **text_columns** (*list*) – list of feature names which should be treated as text features.
- **random_state** (*int, np.random.RandomState*) – Seed for the random number generator.

evalml.pipelines.components.TextFeaturizer.clone`TextFeaturizer.clone (random_state=0)`

Constructs a new component with the same parameters

Parameters `random_state` (*int*, *RandomState*) – the value to seed the random state with. Can also be a *RandomState* instance. Defaults to 0.**Returns** A new instance of this component with identical parameters**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** (*pd.DataFrame* or *np.ndarray*) – the input training data of shape [n_samples, n_features]
- **y** (*pd.Series*, *optional*) – the target training labels 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** (*pd.DataFrame*) – Data to fit and transform
- **y** (*pd.DataFrame*) – Target data

Returns Transformed X**Return type** *pd.DataFrame*

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

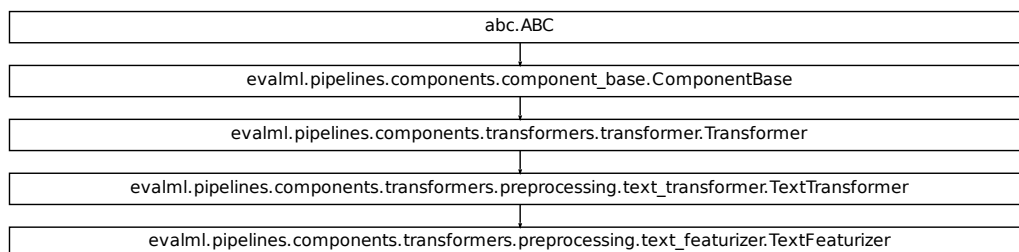
- `file_path` (*str*) – Location to save file
- `pickle_protocol` (*int*) – The pickle data stream format.

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` (*pd.DataFrame*) – Data to transform
- `y` (*pd.Series, optional*) – Ignored.

Returns Transformed X**Return type** `pd.DataFrame`**Class Inheritance**

evalml.pipelines.components.DelayedFeatureTransformer

```
class evalml.pipelines.components.DelayedFeatureTransformer(max_delay=2, delay_features=True,  
                                                         delay_target=True,  
                                                         gap=1, random_state=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

<code>needs_fitting</code>	
<code>parameters</code>	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
<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_state=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_state** (*int*, *np.random.RandomState*) – Seed for the random number generator. There is no randomness in this transformer.

evalml.pipelines.components.DelayedFeatureTransformer.clone

`DelayedFeatureTransformer.clone(random_state=0)`

Constructs a new component with the same parameters

Parameters **random_state** (*int*, *RandomState*) – the value to seed the random state with. Can also be a *RandomState* instance. Defaults to 0.

Returns A new instance of this component with identical parameters

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.

evalml.pipelines.components.DelayedFeatureTransformer.fit_transform

`DelayedFeatureTransformer.fit_transform(X, y=None)`

Fits on X and transforms X

Parameters

- **X** (*pd.DataFrame*) – Data to fit and transform
- **y** (*pd.DataFrame*) – Target data

Returns Transformed X

Return type *pd.DataFrame*

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.

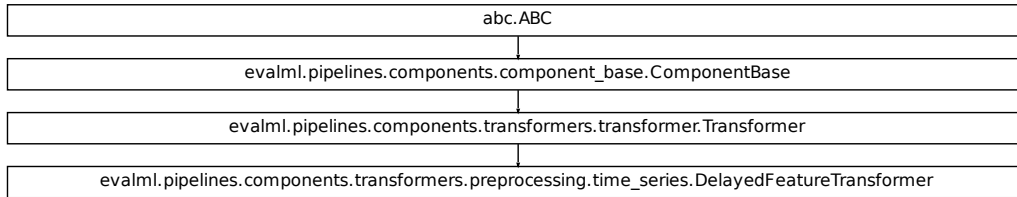
Parameters

- **X** (*pd.DataFrame or None*) – Data to transform. None is expected when only the target variable is being used.
- **y** (*pd.Series, None*) – Target.

Returns Transformed X.

Return type pd.DataFrame

Class Inheritance



evalml.pipelines.components.DFSTransformer

```
class evalml.pipelines.components.DFSTransformer(index='index', random_state=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
<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_state=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_state** (*int*, *np.random.RandomState*) – seed for the random number generator

evalml.pipelines.components.DFSTransformer.clone

`DFSTransformer.clone(random_state=0)`

Constructs a new component with the same parameters

Parameters **random_state** (*int*, *RandomState*) – the value to seed the random state with. Can also be a *RandomState* instance. Defaults to 0.

Returns A new instance of this component with identical parameters

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]

evalml.pipelines.components.DFSTransformer.fit_transform

`DFSTransformer.fit_transform(X, y=None)`

Fits on X and transforms X

Parameters

- **X** (`pd.DataFrame`) – Data to fit and transform
- **y** (`pd.DataFrame`) – Target data

Returns Transformed X

Return type `pd.DataFrame`

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

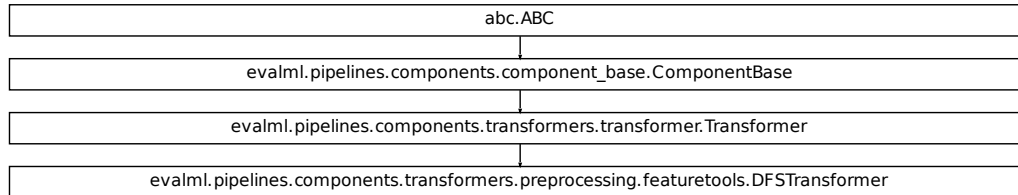
evalml.pipelines.components.DFSTransformer.transform

`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]`

Class Inheritance



5.5.4 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.

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_state=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:
```

```
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
```

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
<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_state=0, **kwargs)
```

Initialize self. See `help(type(self))` for accurate signature.

`evalml.pipelines.components.CatBoostClassifier.clone`

```
CatBoostClassifier.clone(random_state=0)
```

Constructs a new component with the same parameters

Parameters `random_state` (`int`, `RandomState`) – the value to seed the random state with. Can also be a `RandomState` instance. Defaults to 0.

Returns A new instance of this component with identical parameters

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** (*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.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** pd.Series

`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 *pd.DataFrame*

`evalml.pipelines.components.CatBoostClassifier.save`

`CatBoostClassifier.save(file_path, pickle_protocol=5)`

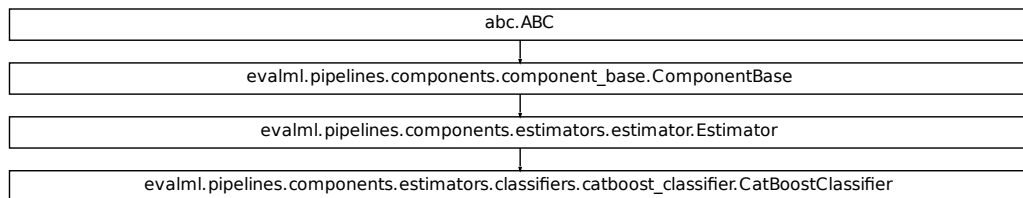
Saves component at file path

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_state=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,
```

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
<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_state=0, penalty='elasticnet', **kwargs)`
 Initialize self. See `help(type(self))` for accurate signature.

`evalml.pipelines.components.ElasticNetClassifier.clone`

`ElasticNetClassifier.clone(random_state=0)`
 Constructs a new component with the same parameters

Parameters `random_state` (*int*, *RandomState*) – the value to seed the random state with. Can also be a *RandomState* instance. Defaults to 0.

Returns A new instance of this component with identical parameters

`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** (*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.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 *pd.Series*

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 *pd.DataFrame*

evalml.pipelines.components.ElasticNetClassifier.save

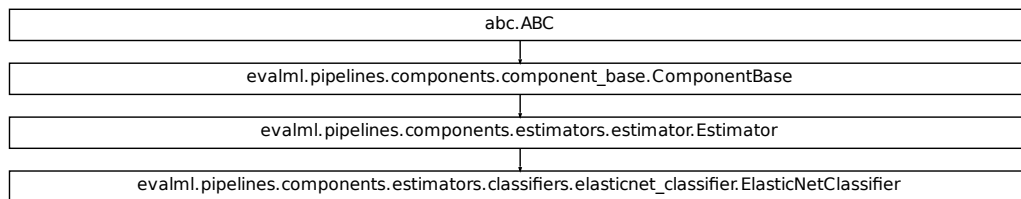
`ElasticNetClassifier.save(file_path, pickle_protocol=5)`

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, random_state=0, **kwargs)

```

Extra Trees Classifier.

name = 'Extra Trees Classifier'

model_family = 'extra_trees'

supported_problem_types = [<ProblemTypes.BINARY: 'binary'>, <ProblemTypes.MULTICLASS: 'multiclass'>]

hyperparameter_ranges = {'max_depth': Integer(low=4, high=10, prior='uniform', transform='log'), 'min_samples_split': Integer(low=2, high=10, prior='uniform', transform='log')}

default_parameters = {'max_depth': 6, 'max_features': 'auto', 'min_samples_split': 2, 'min_weight_fraction_leaf': 0.0, 'n_estimators': 100, 'n_jobs': -1, 'random_state': 0}

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
<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_state=0, **kwargs)`
Initialize self. See `help(type(self))` for accurate signature.

`evalml.pipelines.components.ExtraTreesClassifier.clone`

`ExtraTreesClassifier.clone(random_state=0)`
Constructs a new component with the same parameters

Parameters `random_state` (*int*, *RandomState*) – the value to seed the random state with. Can also be a *RandomState* instance. Defaults to 0.

Returns A new instance of this component with identical parameters

`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** (*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.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 *pd.Series*

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 *pd.DataFrame*

evalml.pipelines.components.ExtraTreesClassifier.save

`ExtraTreesClassifier.save(file_path, pickle_protocol=5)`

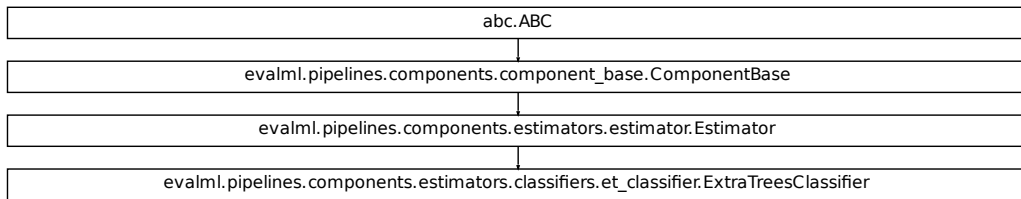
Saves component at file path

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_state=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}
```


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
<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_state=0, **kwargs)`
 Initialize self. See `help(type(self))` for accurate signature.

`evalml.pipelines.components.RandomForestClassifier.clone`

`RandomForestClassifier.clone(random_state=0)`
 Constructs a new component with the same parameters

Parameters `random_state` (*int*, *RandomState*) – the value to seed the random state with. Can also be a *RandomState* instance. Defaults to 0.

Returns A new instance of this component with identical parameters

`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** (*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.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 pd.Series

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 pd.DataFrame

evalml.pipelines.components.RandomForestClassifier.save

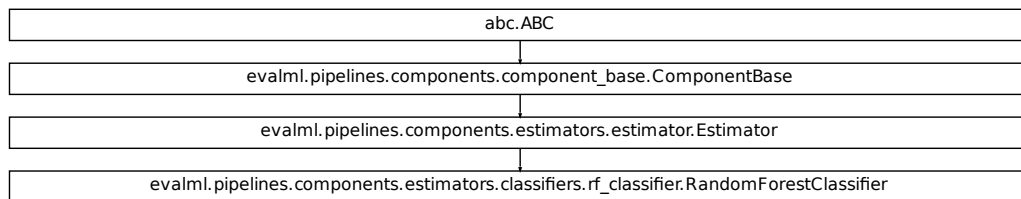
`RandomForestClassifier.save(file_path, pickle_protocol=5)`

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_state=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: 'multiclass'>]

hyperparameter_ranges = {'bagging_fraction': Real(low=1e-06, high=1, prior='uniform', 'lightgbm': 0.9), 'bagging_freq': Integer(low=0, high=5, prior='uniform', 'lightgbm': 5), 'boosting_type': Categorical(values=['gbdt', 'dart', 'goss', 'lgbf'], 'lightgbm': 'gbdt'), 'max_depth': Integer(low=0, high=100, prior='uniform', 'lightgbm': 0), 'min_child_samples': Integer(low=5, high=1000, prior='uniform', 'lightgbm': 20), 'min_child_weight': Real(low=0.01, high=100, prior='log-uniform', 'lightgbm': 0.1), 'num_leaves': Integer(low=31, high=10000, prior='log-uniform', 'lightgbm': 31), 'n_estimators': Integer(low=100, high=10000, prior='log-uniform', 'lightgbm': 100), 'n_jobs': Integer(low=-1, high=16, prior='uniform', 'lightgbm': -1), 'learning_rate': Real(low=0.01, high=1, prior='log-uniform', 'lightgbm': 0.1)}

default_parameters = {'bagging_fraction': 0.9, 'bagging_freq': 0, 'boosting_type': 'gbdt', 'max_depth': 0, 'min_child_samples': 20, 'min_child_weight': 0.01, 'num_leaves': 31, 'n_estimators': 100, 'n_jobs': -1, 'learning_rate': 0.1}

Instance attributes

<code>SEED_MAX</code>	
<code>SEED_MIN</code>	
<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
<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_state=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(random_state=0)`

Constructs a new component with the same parameters

Parameters `random_state` (`int`, `RandomState`) – the value to seed the random state with. Can also be a `RandomState` instance. Defaults to 0.

Returns A new instance of this component with identical parameters

`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** (*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.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 pd.Series

`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 pd.DataFrame

`evalml.pipelines.components.LightGBMClassifier.save`

`LightGBMClassifier.save` (*file_path*, *pickle_protocol=5*)

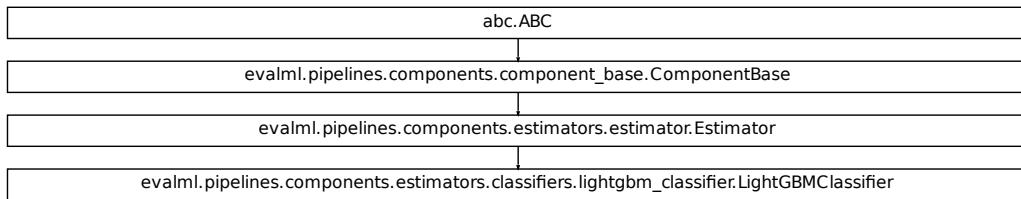
Saves component at file path

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',
```

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
<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_state=0, **kwargs)`

Initialize self. See `help(type(self))` for accurate signature.

evalml.pipelines.components.LogisticRegressionClassifier.clone

`LogisticRegressionClassifier.clone(random_state=0)`

Constructs a new component with the same parameters

Parameters `random_state` (*int*, *RandomState*) – the value to seed the random state with. Can also be a *RandomState* instance. Defaults to 0.

Returns A new instance of this component with identical parameters

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** (*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.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 *pd.Series*

`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 *pd.DataFrame*

evalml.pipelines.components.LogisticRegressionClassifier.save

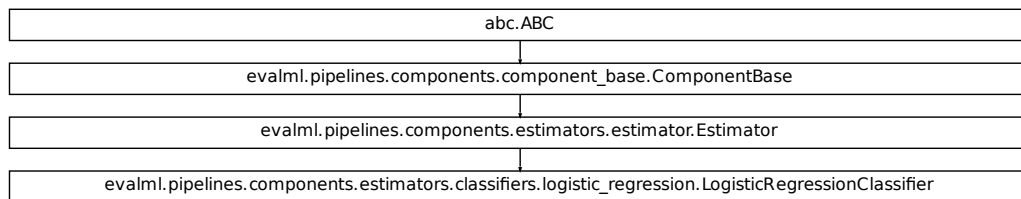
`LogisticRegressionClassifier.save(file_path, pickle_protocol=5)`

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_state=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
```

Instance attributes

<code>SEED_MAX</code>	
<code>SEED_MIN</code>	
<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
<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_state=0, **kwargs)`
Initialize self. See `help(type(self))` for accurate signature.

`evalml.pipelines.components.XGBoostClassifier.clone`

`XGBoostClassifier.clone(random_state=0)`
Constructs a new component with the same parameters

Parameters `random_state` (*int*, *RandomState*) – the value to seed the random state with. Can also be a *RandomState* instance. Defaults to 0.

Returns A new instance of this component with identical parameters

`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** (*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.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 pd.Series

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 pd.DataFrame

`evalml.pipelines.components.XGBoostClassifier.save`

`XGBoostClassifier.save(file_path, pickle_protocol=5)`

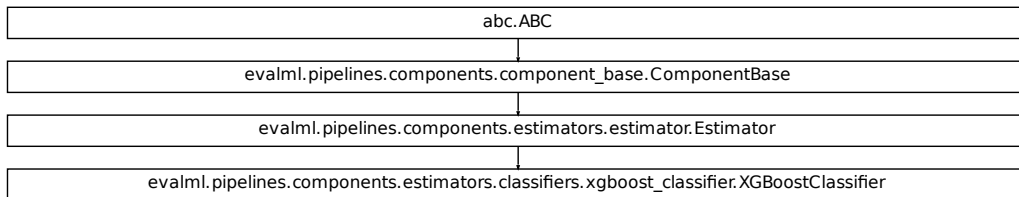
Saves component at file path

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_state=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'}

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
<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_state=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_state** (*int*, *np.random.RandomState*) – Seed for the random number generator

`evalml.pipelines.components.BaselineClassifier.clone`

`BaselineClassifier.clone(random_state=0)`

Constructs a new component with the same parameters

Parameters **random_state** (*int*, *RandomState*) – the value to seed the random state with. Can also be a *RandomState* instance. Defaults to 0.

Returns A new instance of this component with identical parameters

`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** (*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.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 pd.Series

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 *pd.DataFrame*

evalml.pipelines.components.BaselineClassifier.save

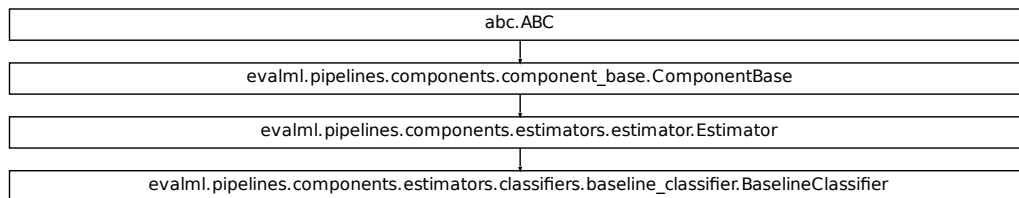
`BaselineClassifier.save(file_path, pickle_protocol=5)`

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,
                                                            final_estimator=None,
                                                            cv=None, n_jobs=1,
                                                            random_state=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}
```

Instance attributes

<code>feature_importance</code>	Not implemented for StackedEnsembleClassifier and StackedEnsembleRegressor
<code>needs_fitting</code>	
<code>parameters</code>	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
<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_state=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_state** (*int*, *np.random.RandomState*) – seed for the random number generator

evalml.pipelines.components.StackedEnsembleClassifier.clone

`StackedEnsembleClassifier.clone(random_state=0)`

Constructs a new component with the same parameters

Parameters **random_state** (*int*, *RandomState*) – the value to seed the random state with. Can also be a *RandomState* instance. Defaults to 0.

Returns A new instance of this component with identical parameters

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** (*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.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 `pd.Series`

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 `pd.DataFrame`

evalml.pipelines.components.StackedEnsembleClassifier.save

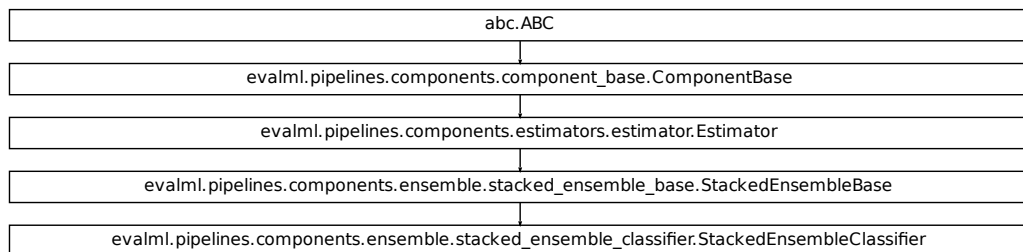
`StackedEnsembleClassifier.save(file_path, pickle_protocol=5)`

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_state=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
```

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
<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_state=0, **kwargs)
```

Initialize self. See help(type(self)) for accurate signature.

evalml.pipelines.components.DecisionTreeClassifier.clone

`DecisionTreeClassifier.clone (random_state=0)`

Constructs a new component with the same parameters

Parameters `random_state` (*int*, *RandomState*) – the value to seed the random state with. Can also be a *RandomState* instance. Defaults to 0.

Returns A new instance of this component with identical parameters

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** (*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.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** *pd.Series***evalml.pipelines.components.DecisionTreeClassifier.predict_proba**`DecisionTreeClassifier.predict_proba(X)`

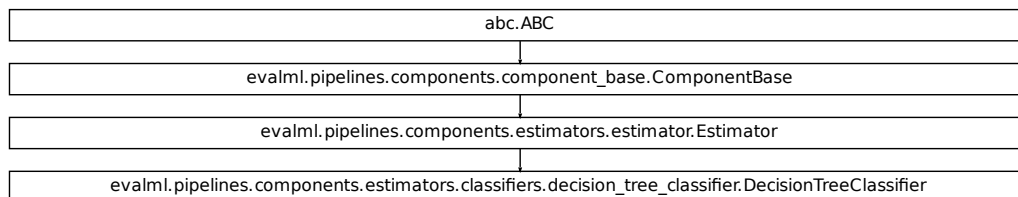
Make probability estimates for labels.

Parameters *X* (*ww.DataTable*, *pd.DataFrame*, or *np.ndarray*) – Features**Returns** Probability estimates**Return type** *pd.DataFrame***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**

Regressors

Regressors are components that output a predicted target value.

<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>TimeSeriesBaselineRegressor</i>	Time series regressor that predicts using the naive forecasting approach.
<i>StackedEnsembleRegressor</i>	Stacked Ensemble Regressor.
<i>DecisionTreeRegressor</i>	Decision Tree Regressor.
<i>LightGBMRegressor</i>	LightGBM Regressor

evalml.pipelines.components.CatBoostRegressor

```
class evalml.pipelines.components.CatBoostRegressor (n_estimators=10,      eta=0.03,
                                                    max_depth=6,          boot-
                                                    strap_type=None,  silent=False,
                                                    allow_writing_files=False,
                                                    random_state=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
```

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
<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_state=0, **kwargs)`

Initialize self. See help(type(self)) for accurate signature.

evalml.pipelines.components.CatBoostRegressor.clone

`CatBoostRegressor.clone(random_state=0)`

Constructs a new component with the same parameters

Parameters `random_state` (*int*, *RandomState*) – the value to seed the random state with. Can also be a *RandomState* instance. Defaults to 0.

Returns A new instance of this component with identical parameters

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** (*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.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 *pd.Series*

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 *pd.DataFrame*

evalml.pipelines.components.CatBoostRegressor.save

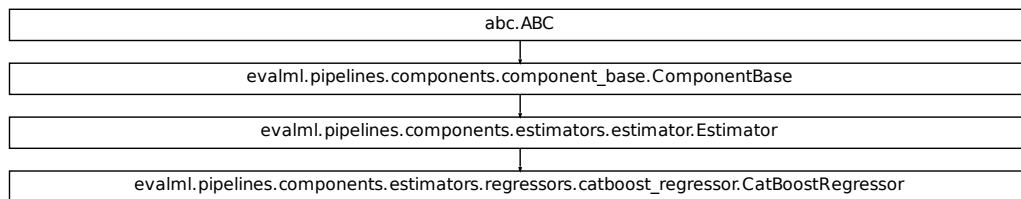
`CatBoostRegressor.save(file_path, pickle_protocol=5)`

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_state=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
```

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
<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_state=0, **kwargs)`
Initialize self. See `help(type(self))` for accurate signature.

`evalml.pipelines.components.ElasticNetRegressor.clone`

`ElasticNetRegressor.clone(random_state=0)`
Constructs a new component with the same parameters

Parameters `random_state` (*int*, *RandomState*) – the value to seed the random state with. Can also be a *RandomState* instance. Defaults to 0.

Returns A new instance of this component with identical parameters

`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** (*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.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 *pd.Series*

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 *pd.DataFrame*

`evalml.pipelines.components.ElasticNetRegressor.save`

`ElasticNetRegressor.save` (*file_path*, *pickle_protocol=5*)

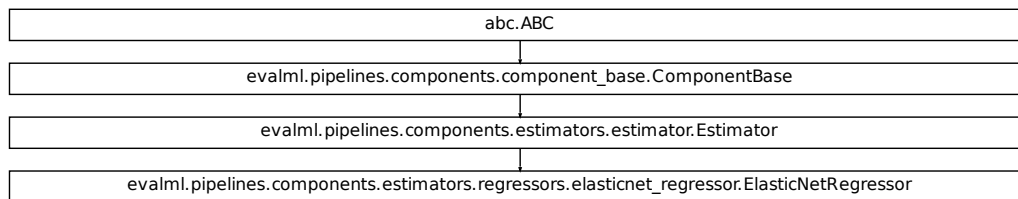
Saves component at file path

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_state=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}
```

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
<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_state=0, **kwargs)`
 Initialize self. See `help(type(self))` for accurate signature.

`evalml.pipelines.components.LinearRegressor.clone`

`LinearRegressor.clone(random_state=0)`
 Constructs a new component with the same parameters

Parameters `random_state` (`int`, `RandomState`) – the value to seed the random state with. Can also be a `RandomState` instance. Defaults to 0.

Returns A new instance of this component with identical parameters

`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** (*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.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 *pd.Series*

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 *pd.DataFrame*

evalml.pipelines.components.LinearRegressor.save

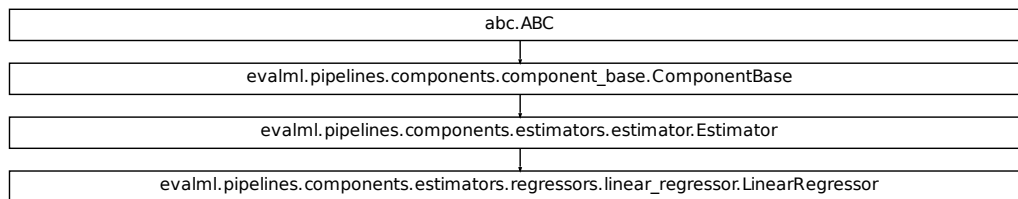
`LinearRegressor.save(file_path, pickle_protocol=5)`

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_state=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,

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
<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_state=0, **kwargs)`
Initialize self. See `help(type(self))` for accurate signature.

`evalml.pipelines.components.ExtraTreesRegressor.clone`

`ExtraTreesRegressor.clone(random_state=0)`
Constructs a new component with the same parameters

Parameters `random_state` (*int*, *RandomState*) – the value to seed the random state with. Can also be a *RandomState* instance. Defaults to 0.

Returns A new instance of this component with identical parameters

`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** (*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.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 *pd.Series*

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 *pd.DataFrame*

evalml.pipelines.components.ExtraTreesRegressor.save

`ExtraTreesRegressor.save(file_path, pickle_protocol=5)`

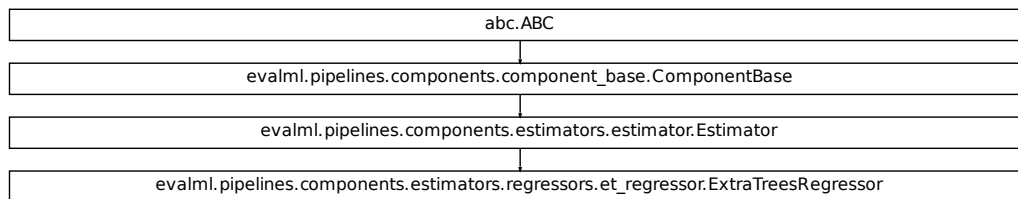
Saves component at file path

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_state=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}
```

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
<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_state=0, **kwargs)`
 Initialize self. See `help(type(self))` for accurate signature.

`evalml.pipelines.components.RandomForestRegressor.clone`

`RandomForestRegressor.clone(random_state=0)`
 Constructs a new component with the same parameters

Parameters `random_state` (*int*, *RandomState*) – the value to seed the random state with. Can also be a *RandomState* instance. Defaults to 0.

Returns A new instance of this component with identical parameters

`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** (*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.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 pd.Series

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 pd.DataFrame

evalml.pipelines.components.RandomForestRegressor.save

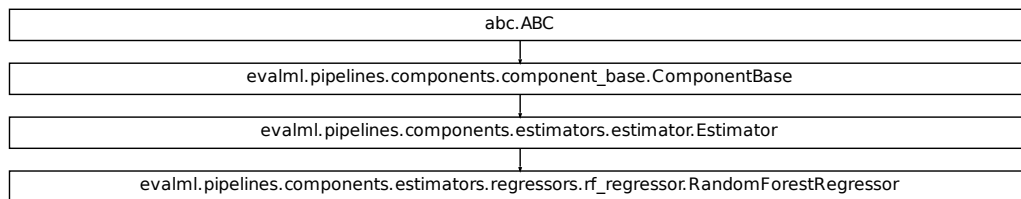
`RandomForestRegressor.save(file_path, pickle_protocol=5)`

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_state=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

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
<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_state=0, **kwargs)`
Initialize self. See `help(type(self))` for accurate signature.

`evalml.pipelines.components.XGBoostRegressor.clone`

`XGBoostRegressor.clone(random_state=0)`
Constructs a new component with the same parameters

Parameters `random_state` (*int*, *RandomState*) – the value to seed the random state with. Can also be a *RandomState* instance. Defaults to 0.

Returns A new instance of this component with identical parameters

`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** (*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.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 *pd.Series*

`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 *pd.DataFrame*

`evalml.pipelines.components.XGBoostRegressor.save`

`XGBoostRegressor.save(file_path, pickle_protocol=5)`

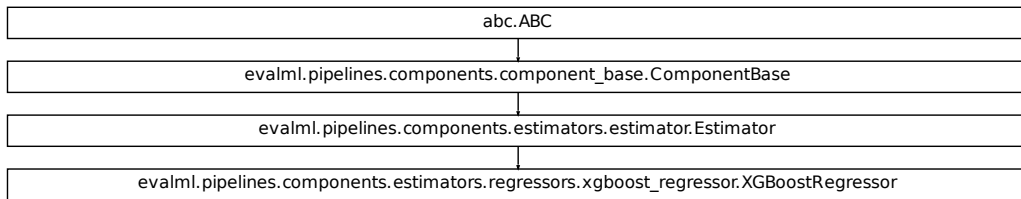
Saves component at file path

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_state=0, **kwargs*) *ran-*

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'}

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 regressor that uses a simple strategy to make predictions.
<code>clone</code>	Constructs a new component with the same parameters
<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_state=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_state** (*int*, *np.random.RandomState*) – seed for the random number generator

evalml.pipelines.components.BaselineRegressor.clone

`BaselineRegressor.clone(random_state=0)`

Constructs a new component with the same parameters

Parameters **random_state** (*int*, *RandomState*) – the value to seed the random state with. Can also be a *RandomState* instance. Defaults to 0.

Returns A new instance of this component with identical parameters

`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** (*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.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 pd.Series

evalml.pipelines.components.BaselineRegressor.predict_proba`BaselineRegressor.predict_proba(X)`

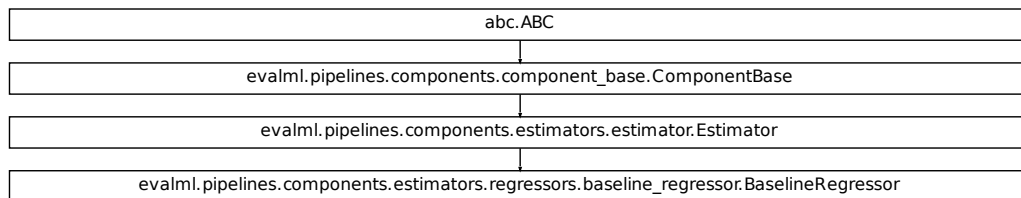
Make probability estimates for labels.

Parameters *X* (*ww.DataTable*, *pd.DataFrame*, or *np.ndarray*) – Features**Returns** Probability estimates**Return type** *pd.DataFrame***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.TimeSeriesBaselineRegressor**

```

class evalml.pipelines.components.TimeSeriesBaselineRegressor(gap=1, random_state=0,
**kwargs)

```

Time series regressor that predicts using the naive forecasting approach.

This is useful as a simple baseline regressor for time series problems

name = 'Time Series Baseline Regressor'**model_family** = 'baseline'**supported_problem_types** = [`ProblemTypes.TIME_SERIES_REGRESSION`: 'time series regression']**hyperparameter_ranges** = {}

```
default_parameters = {'gap': 1}
```

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 regressor that predicts using the naive forecasting approach.
<code>clone</code>	Constructs a new component with the same parameters
<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.TimeSeriesBaselineRegressor.__init__`

`TimeSeriesBaselineRegressor.__init__(gap=1, random_state=0, **kwargs)`

Baseline time series regressor 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** (*int*, *np.random.RandomState*) – seed for the random number generator

`evalml.pipelines.components.TimeSeriesBaselineRegressor.clone`

`TimeSeriesBaselineRegressor.clone(random_state=0)`

Constructs a new component with the same parameters

Parameters **random_state** (*int*, *RandomState*) – the value to seed the random state with. Can also be a *RandomState* instance. Defaults to 0.

Returns A new instance of this component with identical parameters

evalml.pipelines.components.TimeSeriesBaselineRegressor.describe

`TimeSeriesBaselineRegressor.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.TimeSeriesBaselineRegressor.fit

`TimeSeriesBaselineRegressor.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.TimeSeriesBaselineRegressor.load

static `TimeSeriesBaselineRegressor.load` (*file_path*)

Loads component at file path

Parameters **file_path** (*str*) – Location to load file

Returns ComponentBase object

evalml.pipelines.components.TimeSeriesBaselineRegressor.predict

`TimeSeriesBaselineRegressor.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 pd.Series

evalml.pipelines.components.TimeSeriesBaselineRegressor.predict_proba

`TimeSeriesBaselineRegressor.predict_proba(X)`

Make probability estimates for labels.

Parameters **X** (*ww.DataTable*, *pd.DataFrame*, or *np.ndarray*) – Features

Returns Probability estimates

Return type *pd.DataFrame*

evalml.pipelines.components.TimeSeriesBaselineRegressor.save

`TimeSeriesBaselineRegressor.save(file_path, pickle_protocol=5)`

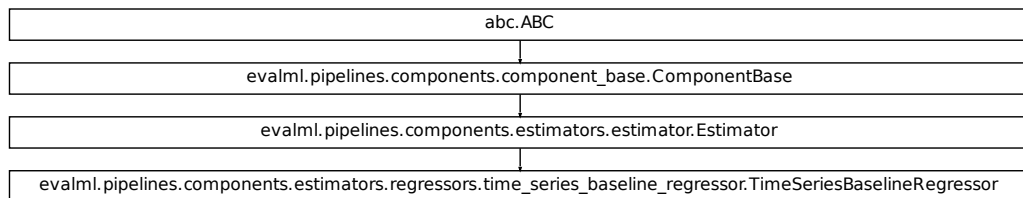
Saves component at file path

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,
                                                            final_estimator=None,
                                                            cv=None, n_jobs=1,
                                                            random_state=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}
```

Instance attributes

<code>feature_importance</code>	Not implemented for StackedEnsembleClassifier and StackedEnsembleRegressor
<code>needs_fitting</code>	
<code>parameters</code>	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
<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_state=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_state** (*int, np.random.RandomState*) – Seed for the random number generator

`evalml.pipelines.components.StackedEnsembleRegressor.clone`

`StackedEnsembleRegressor.clone (random_state=0)`

Constructs a new component with the same parameters

Parameters `random_state` (*int*, *RandomState*) – the value to seed the random state with. Can also be a *RandomState* instance. Defaults to 0.

Returns A new instance of this component with identical parameters

`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** (*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.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 *pd.Series*

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 *pd.DataFrame*

evalml.pipelines.components.StackedEnsembleRegressor.save

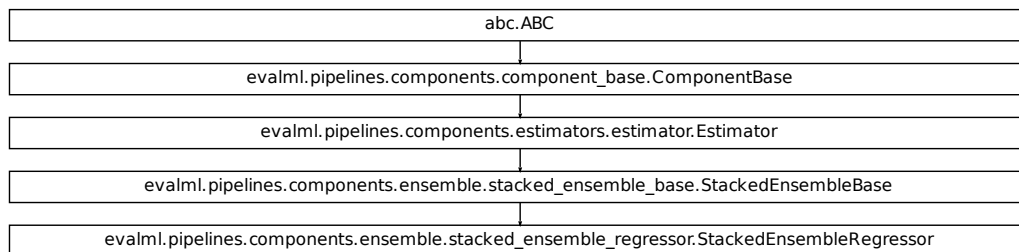
StackedEnsembleRegressor.**save** (*file_path*, *pickle_protocol=5*)

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_state=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
```

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
<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_state=0, **kwargs)
```

Initialize self. See help(type(self)) for accurate signature.

evalml.pipelines.components.DecisionTreeRegressor.clone

`DecisionTreeRegressor.clone (random_state=0)`

Constructs a new component with the same parameters

Parameters `random_state` (*int*, *RandomState*) – the value to seed the random state with. Can also be a *RandomState* instance. Defaults to 0.

Returns A new instance of this component with identical parameters

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** (*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.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 *pd.Series*

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 *pd.DataFrame*

evalml.pipelines.components.DecisionTreeRegressor.save

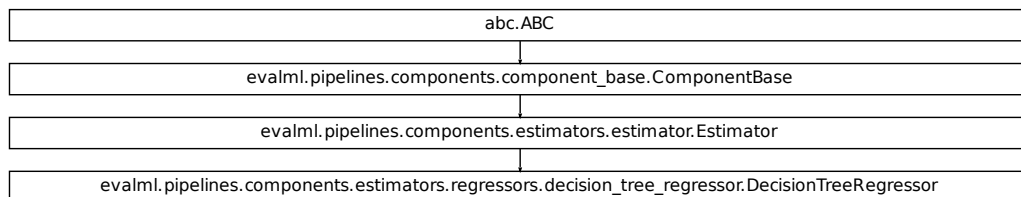
`DecisionTreeRegressor.save(file_path, pickle_protocol=5)`

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',    learn-
                                                    ing_rate=0.1, n_estimators=20,
                                                    max_depth=0, num_leaves=31,
                                                    min_child_samples=20,
                                                    n_jobs=- 1, random_state=0,
                                                    bagging_fraction=0.9,    bag-
                                                    ging_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

```

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
<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_state=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` (*random_state=0*)

Constructs a new component with the same parameters

Parameters `random_state` (*int, RandomState*) – the value to seed the random state with. Can also be a `RandomState` instance. Defaults to 0.

Returns A new instance of this component with identical parameters

`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** (*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.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 `pd.Series`

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 `pd.DataFrame`

evalml.pipelines.components.LightGBMRegressor.save

`LightGBMRegressor.save(file_path, pickle_protocol=5)`

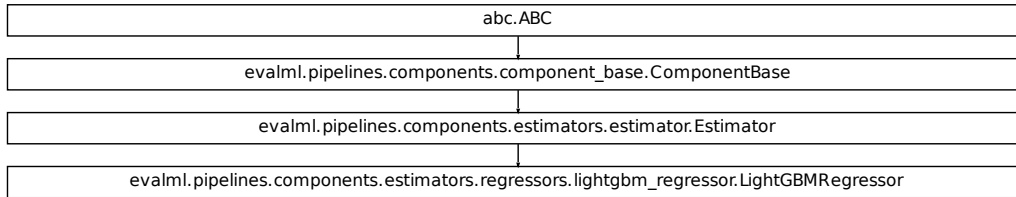
Saves component at file path

Parameters

- `file_path` (*str*) – Location to save file
- `pickle_protocol` (*int*) – The pickle data stream format.

Returns None

Class Inheritance



5.6 Model Understanding

5.6.1 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 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.

`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_state=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_state** (*int*, *np.random.RandomState*) – The random seed/state. Defaults to 0.

Returns 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, feature,
                                              grid_resolution=100)
```

Calculates partial dependence.

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

- **feature** (*int, string*) – The target features for which to create the partial dependence plot for. If feature is an int, it must be the index of the feature to use. If feature is a string, it must be a valid column name in X.

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. 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.

Return type `pd.DataFrame`

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

5.6.2 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.

continues on next page

Table 130 – continued from previous page

<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 partial dependence plot.

`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 `plotly.Figure` showing the prediction vs actual over time.

`evalml.model_understanding.graph_partial_dependence`

`evalml.model_understanding.graph_partial_dependence(pipeline, X, feature, class_label=None, grid_resolution=100)`

Create an one-way partial dependence plot.

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
- **feature** (*int, string*) – The target feature for which to create the partial dependence plot for. If feature is an int, it must be the index of the feature to use. If feature is a string, it must be a valid column name 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.

Returns

pd.DataFrame with averaged predictions for all points in the grid averaged over all samples of X and the values used to calculate those predictions.

Return type pd.DataFrame

5.6.3 Prediction Explanations

<code>explain_prediction</code>	Creates table summarizing the top_k positive and top_k negative contributing features to the prediction of a single datapoint.
<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_prediction

```
evalml.model_understanding.prediction_explanations.explain_prediction (pipeline,
                                                                    in-
                                                                    put_features,
                                                                    top_k=3,
                                                                    train-
                                                                    ing_data=None,
                                                                    in-
                                                                    clude_shap_values=False,
                                                                    out-
                                                                    put_format='text')
```

Creates table summarizing the top_k positive and top_k negative contributing features to the prediction of a single datapoint.

XGBoost models and 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 features - needs to correspond to data the pipeline was fit on.
- **top_k** (*int*) – How many of the highest/lowest features to include in the table.

- **training_data** (*pd.DataFrame*) – Training data the pipeline was fit on. This is required for non-tree estimators because we need a sample of training data for the KernelSHAP algorithm.
- **include_shap_values** (*bool*) – Whether the SHAP values should be included in an extra column in the output. Default is False.
- **output_format** (*str*) – Either “text” or “dict”. Default is “text”.

Returns *str* or *dict* - A report explaining the most positive/negative contributing features to the predictions.

evalml.model_understanding.prediction_explanations.explain_predictions

```
evalml.model_understanding.prediction_explanations.explain_predictions(pipeline,
                                                                    in-
                                                                    put_features,
                                                                    train-
                                                                    ing_data=None,
                                                                    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 models and 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.
- **training_data** (*ww.DataTable, pd.DataFrame*) – Dataframe of data the pipeline was fit on. This can be omitted for pipelines with tree-based estimators.
- **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.
- **output_format** (*str*) – Either “text” or “dict”. Default is “text”.

Returns

str or dict - A report explaining the top contributing features to each prediction for each row of *input_features*. The report will include the feature names, prediction contribution, and SHAP Value (optional).

evalml.model_understanding.prediction_explanations.explain_predictions_best_worst

`evalml.model_understanding.prediction_explanations.explain_predictions_best_worst` (*pipeline*,
input_features,
y_true,
num_to_explain,
top_k_features,
include_shap_values,
metric=None,
output_format='text')

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 models and 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 or dict - A report explaining the top contributing features for the best/worst predictions in the input_features
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.

5.7 Objective Functions

5.7.1 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.

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. This will be the difference normalized by the baseline 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*)

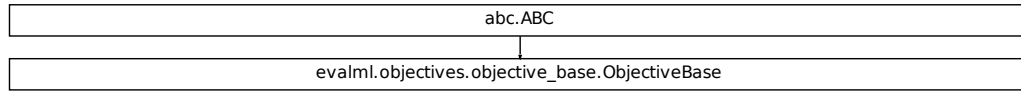
Validates the input based on a few simple checks.

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 (list(ProblemType)): List of problem types that this objective is defined for. Set to [ProblemTypes.BINARY]. **can_optimize_threshold** (bool): Determines if threshold used by objective can be optimized or not.

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*,
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. This will be the difference normalized by the baseline 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(y_pred_proba, y_true, X=None)`

Learn a binary classification threshold which optimizes the current objective.

Parameters

- **y_pred_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)`

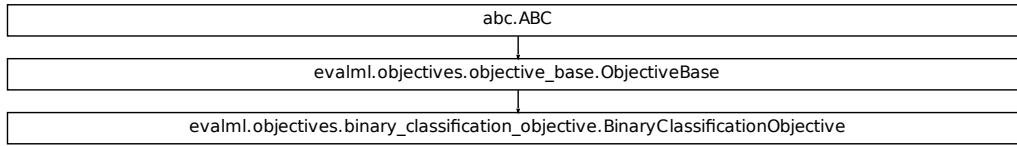
Validates the input based on a few simple checks.

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 (list(ProblemType)): List of problem types that this objective is defined for. Set to [ProblemTypes.MULTICLASS].

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. This will be the difference normalized by the baseline 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)`

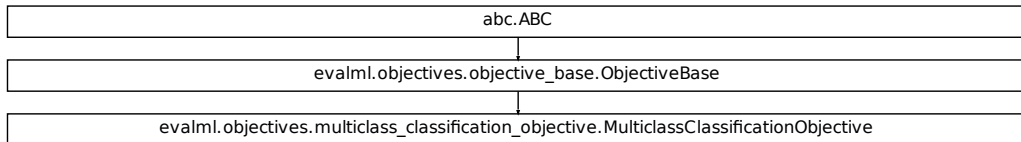
Validates the input based on a few simple checks.

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 (*list(ProblemType)*): List of problem types that this objective is defined for. Set to [ProblemTypes.REGRESSION, ProblemTypes.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. This will be the difference normalized by the baseline 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)`

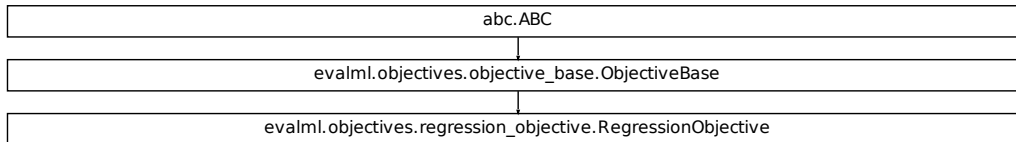
Validates the input based on a few simple checks.

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



5.7.2 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.

Methods

<code>__init__</code>	Create instance of FraudCost
<code>calculate_percent_difference</code>	Calculate the percent difference between scores.
<code>decision_function</code>	Determine if a transaction is fraud given predicted probabilities, threshold, and dataframe with transaction amount.
<code>is_defined_for_problem_type</code>	
<code>objective_function</code>	Calculate amount lost to fraud per transaction given predictions, true values, and dataframe with transaction amount.
<code>optimize_threshold</code>	Learn a binary classification threshold which optimizes the current objective.
<code>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.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. This will be the difference normalized by the baseline 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)`

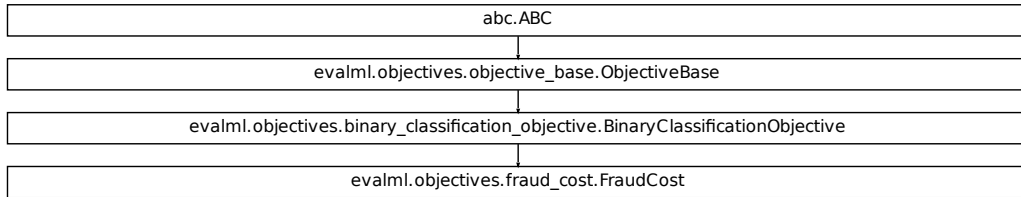
Validates the input based on a few simple checks.

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.

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. This will be the difference normalized by the baseline 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)`

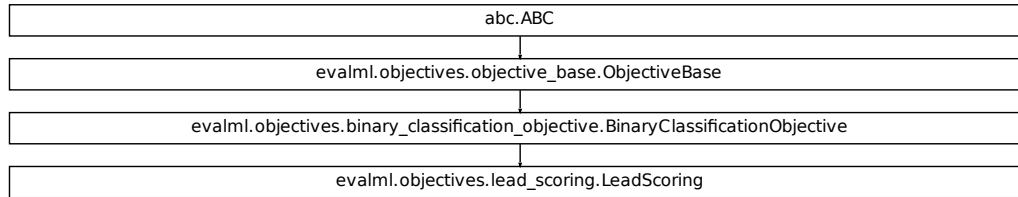
Validates the input based on a few simple checks.

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.

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. This will be the difference normalized by the baseline 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(y_pred_proba, y_true, X=None)`

Learn a binary classification threshold which optimizes the current objective.

Parameters

- **y_pred_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)`

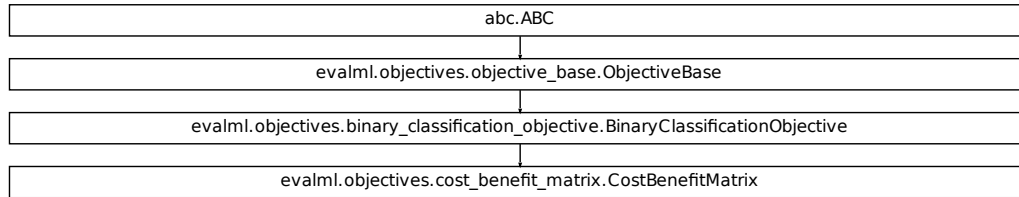
Validates the input based on a few simple checks.

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



5.7.3 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 141 – 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.

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. This will be the difference normalized by the baseline 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*)

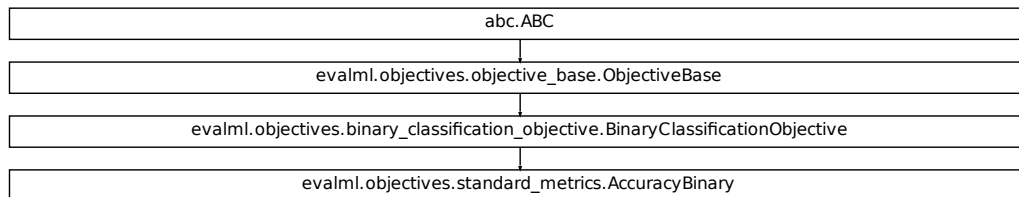
Validates the input based on a few simple checks.

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.

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. This will be the difference normalized by the baseline 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)`

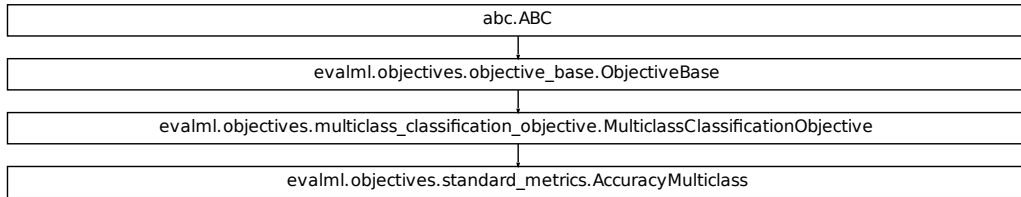
Validates the input based on a few simple checks.

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.

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. This will be the difference normalized by the baseline 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*)

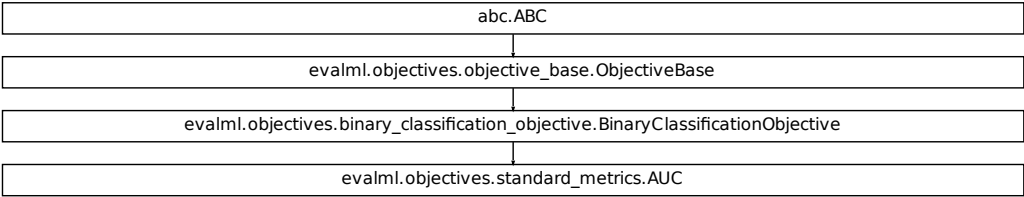
Validates the input based on a few simple checks.

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.

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. This will be the difference normalized by the baseline 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

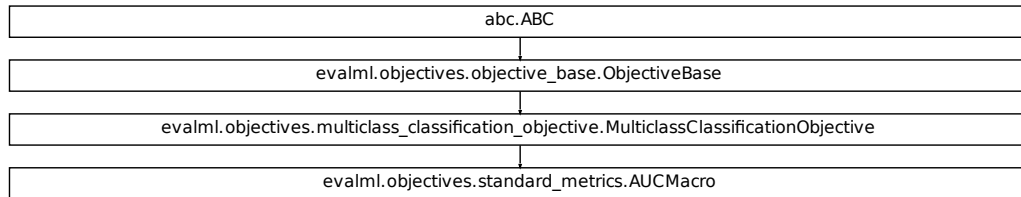
AUCMacro.validate_inputs (*y_true*, *y_predicted*)

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.

Methods

<code><u>__init__</u></code>	Initialize self.
<code><u>calculate_percent_difference</u></code>	Calculate the percent difference between scores.
<code><u>is_defined_for_problem_type</u></code>	
<code><u>objective_function</u></code>	Computes the relative value of the provided predictions compared to the actual labels, according a specified metric
<code><u>score</u></code>	Returns a numerical score indicating performance based on the differences between the predicted and actual values.
<code><u>validate_inputs</u></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. This will be the difference normalized by the baseline 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)`

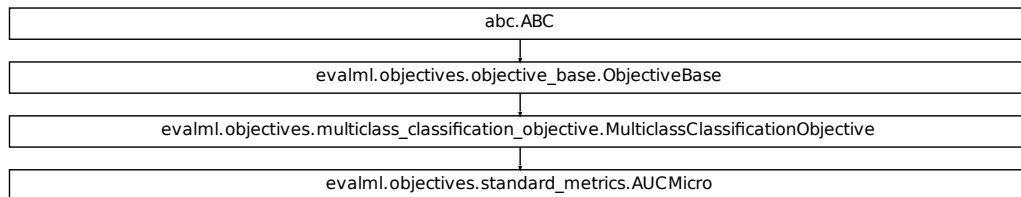
Validates the input based on a few simple checks.

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.

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. This will be the difference normalized by the baseline 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*)

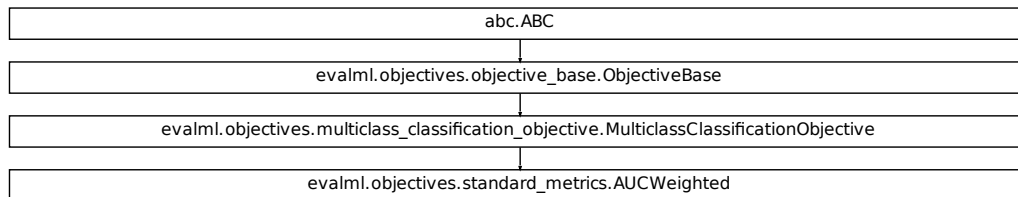
Validates the input based on a few simple checks.

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.

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.

continues on next page

Table 148 – continued from previous page

<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.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. This will be the difference normalized by the baseline 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

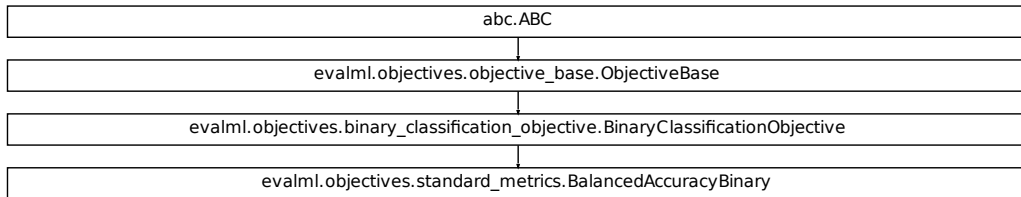
`BalancedAccuracyBinary.validate_inputs(y_true, y_predicted)`

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.

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*,
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. This will be the difference normalized by the baseline 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*)

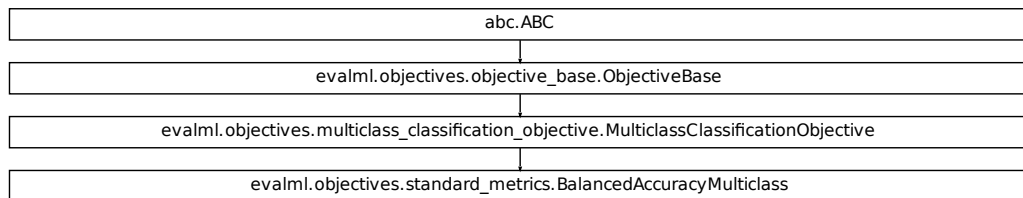
Validates the input based on a few simple checks.

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.

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

continues on next page

Table 150 – continued from previous page

<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. This will be the difference normalized by the baseline 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

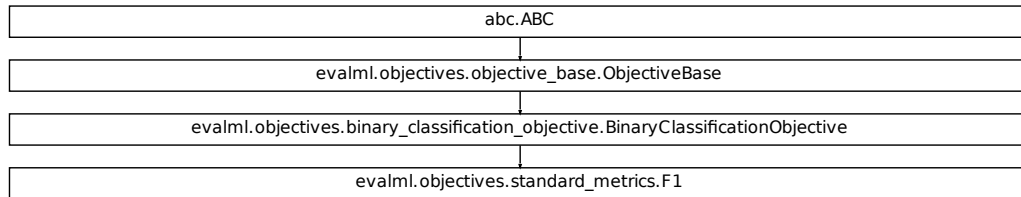
Returns score

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.

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. This will be the difference normalized by the baseline 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)`

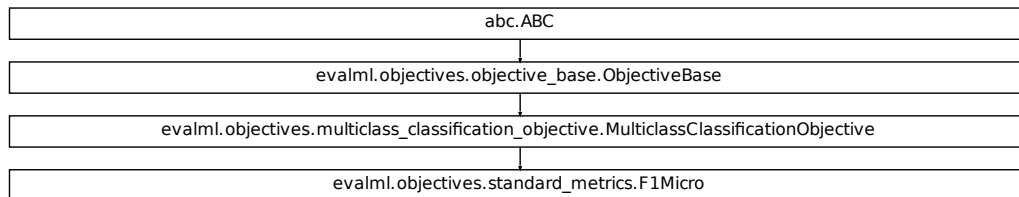
Validates the input based on a few simple checks.

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.

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. This will be the difference normalized by the baseline 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)`

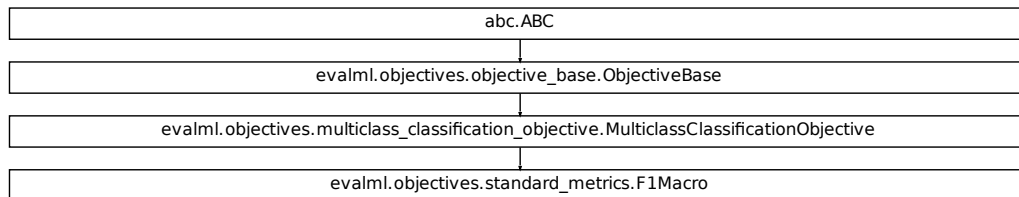
Validates the input based on a few simple checks.

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.

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. This will be the difference normalized by the baseline 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)`

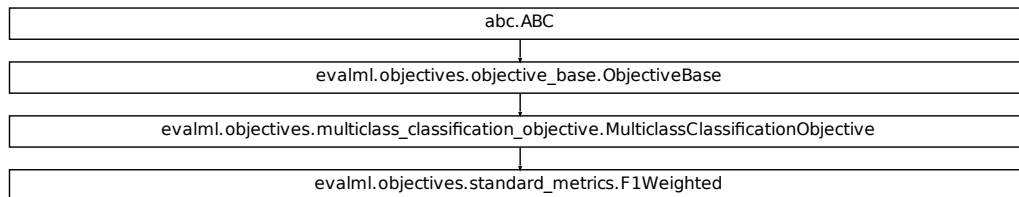
Validates the input based on a few simple checks.

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.

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.

continues on next page

Table 154 – continued from previous page

<code>score</code>	Returns a numerical score indicating performance based on the differences between the predicted and actual values.
<code>validate_inputs</code>	Validates the input based on a few simple checks.

`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. This will be the difference normalized by the baseline 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

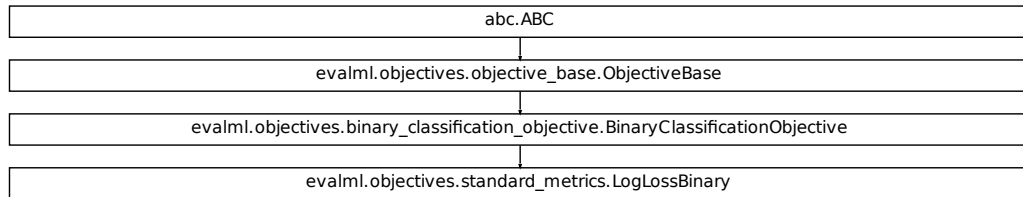
`LogLossBinary.validate_inputs(y_true, y_predicted)`

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.

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. This will be the difference normalized by the baseline 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

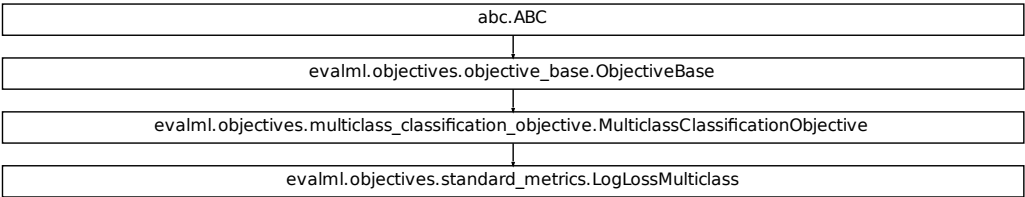
LogLossMulticlass.**validate_inputs** (*y_true*, *y_predicted*)
Validates the input based on a few simple checks.

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.

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.

continues on next page

Table 156 – continued from previous page

<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.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. This will be the difference normalized by the baseline 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

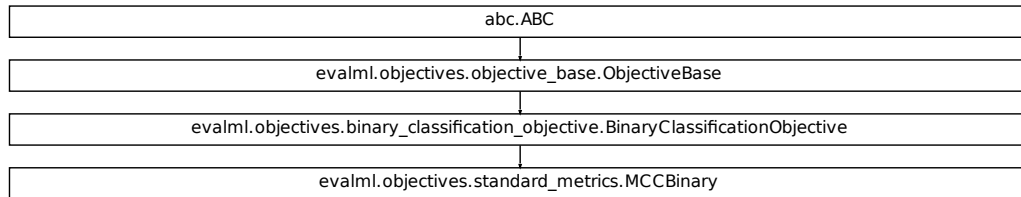
MCCBinary.**validate_inputs**(*y_true*, *y_predicted*)

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.

Methods

<code><u>__init__</u></code>	Initialize self.
<code><u>calculate_percent_difference</u></code>	Calculate the percent difference between scores.
<code><u>is_defined_for_problem_type</u></code>	
<code><u>objective_function</u></code>	Computes the relative value of the provided predictions compared to the actual labels, according a specified metric
<code><u>score</u></code>	Returns a numerical score indicating performance based on the differences between the predicted and actual values.
<code><u>validate_inputs</u></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. This will be the difference normalized by the baseline 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*)

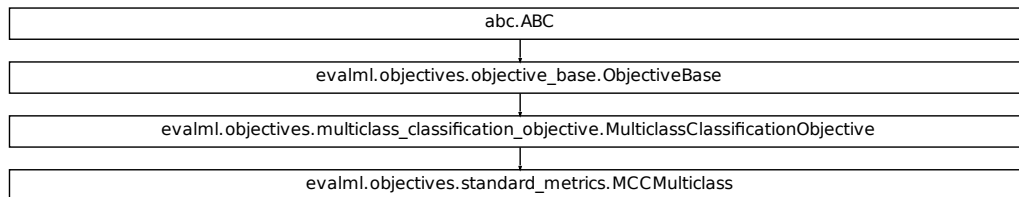
Validates the input based on a few simple checks.

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.

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.

continues on next page

Table 158 – continued from previous page

<code>score</code>	Returns a numerical score indicating performance based on the differences between the predicted and actual values.
<code>validate_inputs</code>	Validates the input based on a few simple checks.

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. This will be the difference normalized by the baseline 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 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)`

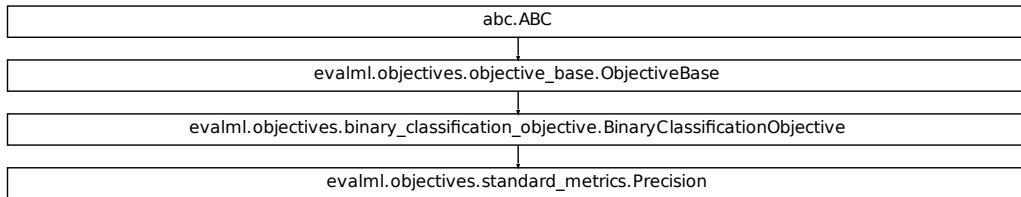
Validates the input based on a few simple checks.

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.

Methods

<code><u>__init__</u></code>	Initialize self.
<code><u>calculate_percent_difference</u></code>	Calculate the percent difference between scores.
<code><u>is_defined_for_problem_type</u></code>	
<code><u>objective_function</u></code>	Computes the relative value of the provided predictions compared to the actual labels, according a specified metric
<code><u>score</u></code>	Returns a numerical score indicating performance based on the differences between the predicted and actual values.
<code><u>validate_inputs</u></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. This will be the difference normalized by the baseline 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)`

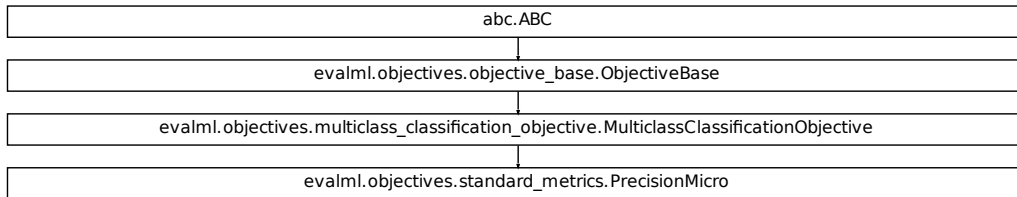
Validates the input based on a few simple checks.

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.

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. This will be the difference normalized by the baseline 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)`

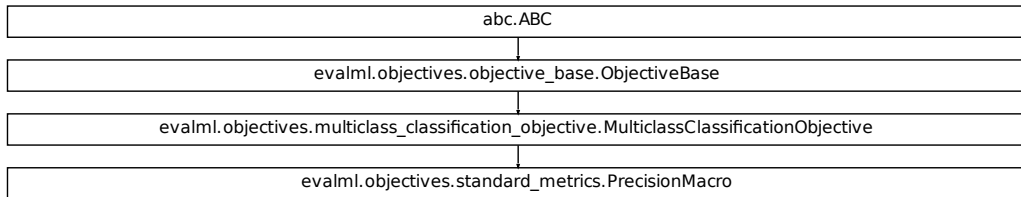
Validates the input based on a few simple checks.

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.

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. This will be the difference normalized by the baseline 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)`

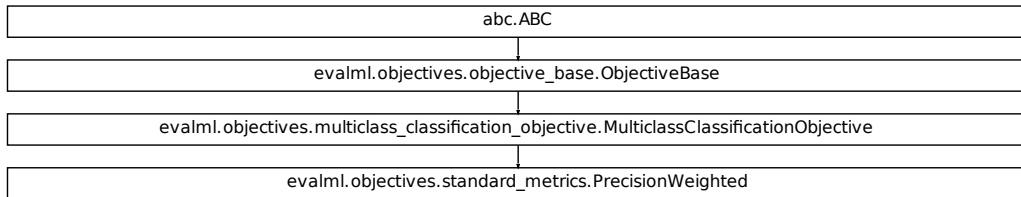
Validates the input based on a few simple checks.

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.

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.

continues on next page

Table 162 – continued from previous page

<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.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. This will be the difference normalized by the baseline 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

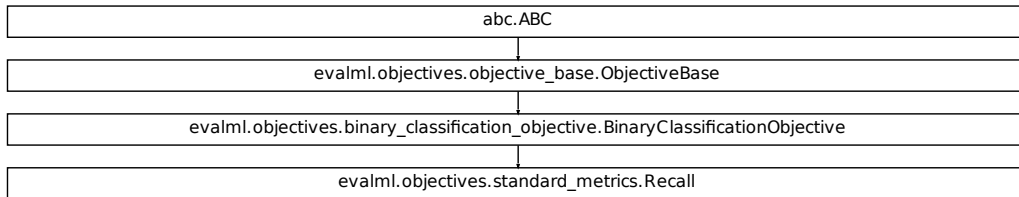
`Recall.validate_inputs(y_true, y_predicted)`

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.

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. This will be the difference normalized by the baseline 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*)

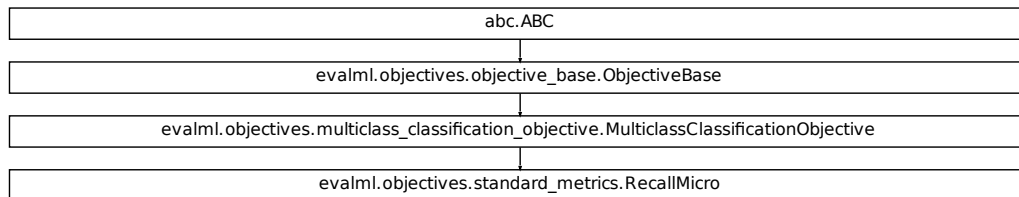
Validates the input based on a few simple checks.

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.

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. This will be the difference normalized by the baseline 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*)

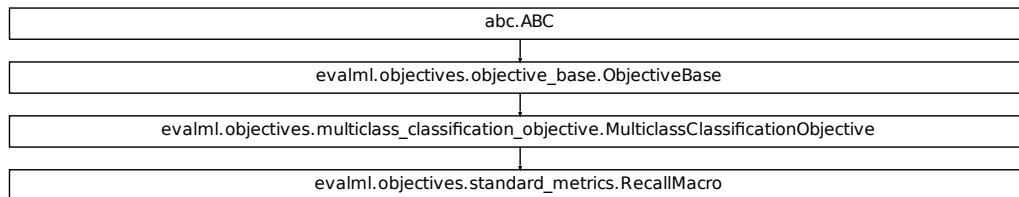
Validates the input based on a few simple checks.

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.

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. This will be the difference normalized by the baseline 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*)

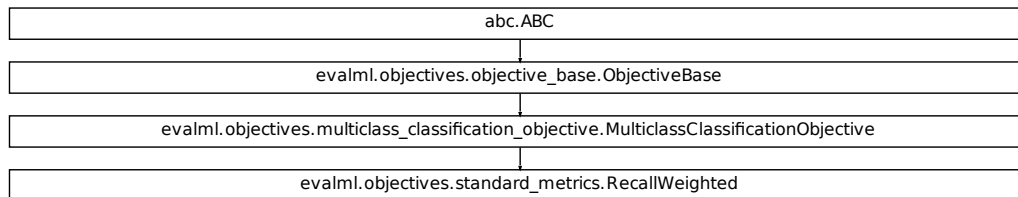
Validates the input based on a few simple checks.

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



5.7.4 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.

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. This will be the difference normalized by the baseline 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)`

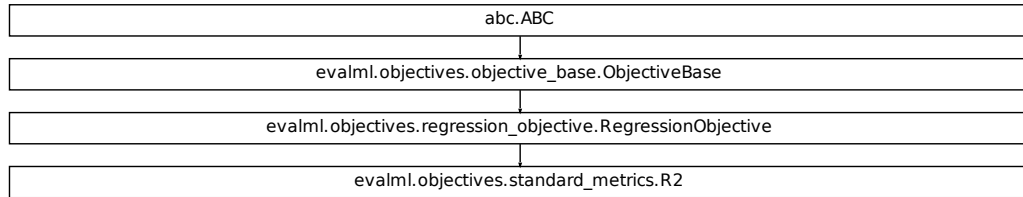
Validates the input based on a few simple checks.

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.

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.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. This will be the difference normalized by the baseline 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*)

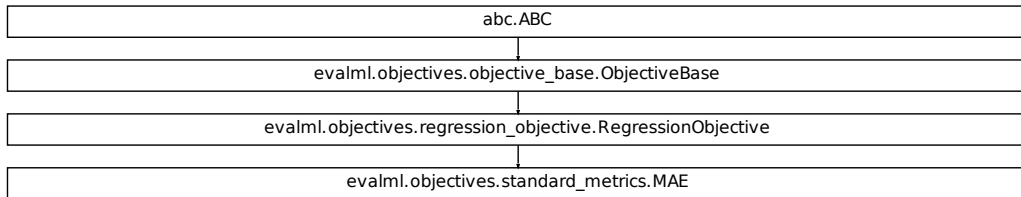
Validates the input based on a few simple checks.

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

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. This will be the difference normalized by the baseline 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

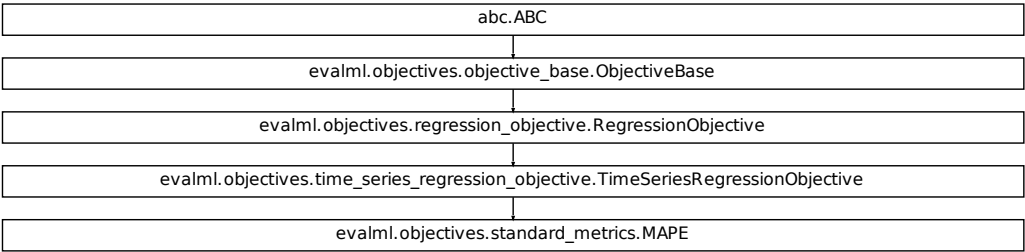
MAPE.validate_inputs (*y_true*, *y_predicted*)
Validates the input based on a few simple checks.

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.

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.

continues on next page

Table 170 – continued from previous page

<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. This will be the difference normalized by the baseline 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*)

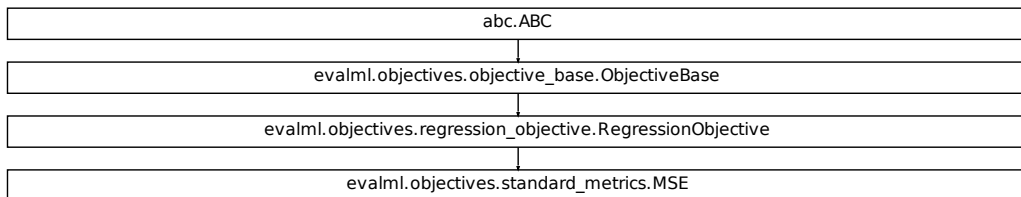
Validates the input based on a few simple checks.

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

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. This will be the difference normalized by the baseline 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)`

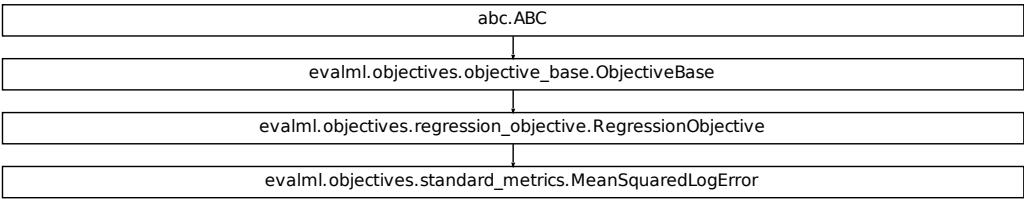
Validates the input based on a few simple checks.

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.

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. This will be the difference normalized by the baseline 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.scoreMedianAE.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

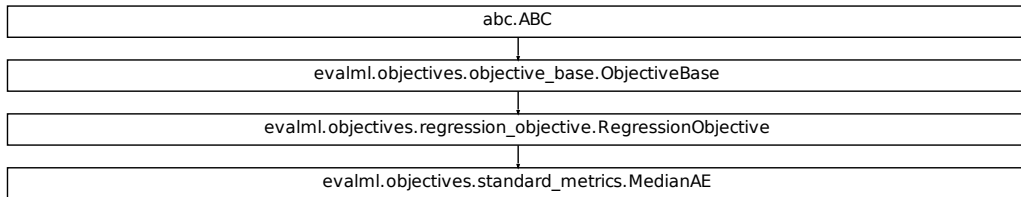
MedianAE.**validate_inputs** (*y_true*, *y_predicted*)

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.

Methods

<code><u>__init__</u></code>	Initialize self.
<code><u>calculate_percent_difference</u></code>	Calculate the percent difference between scores.
<code><u>is_defined_for_problem_type</u></code>	
<code><u>objective_function</u></code>	Computes the relative value of the provided predictions compared to the actual labels, according a specified metric
<code><u>score</u></code>	Returns a numerical score indicating performance based on the differences between the predicted and actual values.
<code><u>validate_inputs</u></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. This will be the difference normalized by the baseline 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*)

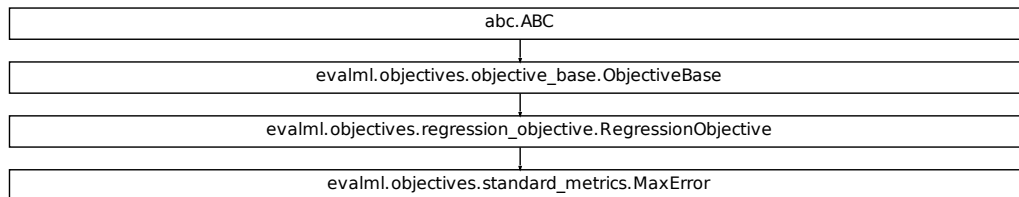
Validates the input based on a few simple checks.

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.

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. This will be the difference normalized by the baseline 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)`

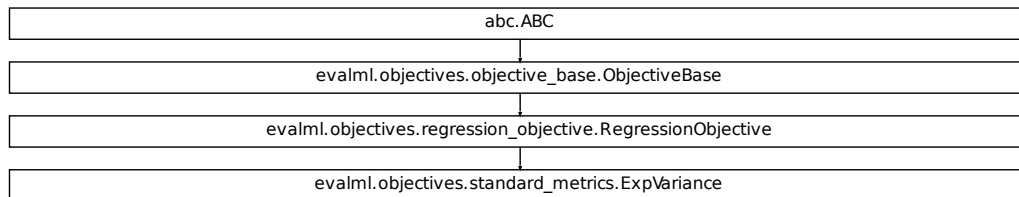
Validates the input based on a few simple checks.

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.

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. This will be the difference normalized by the baseline 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*)

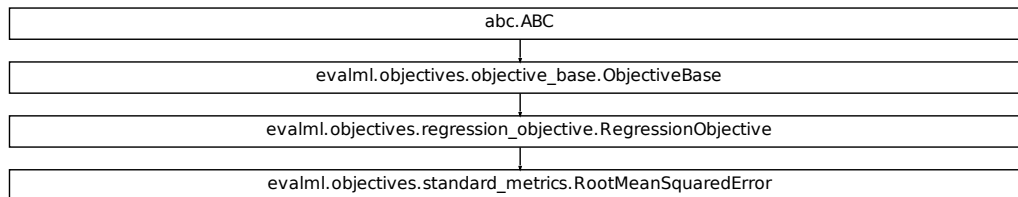
Validates the input based on a few simple checks.

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.

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.

continues on next page

Table 176 – continued from previous page

<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*,
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. This will be the difference normalized by the baseline 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

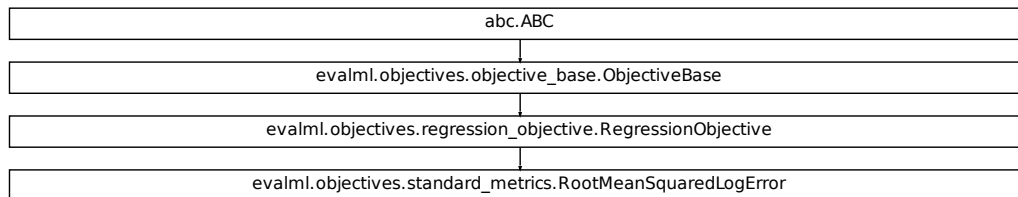
`RootMeanSquaredLogError.validate_inputs` (*y_true*, *y_predicted*)

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

5.7.5 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.

5.8 Problem Types

<code>handle_problem_types</code>	Handles <code>problem_type</code> by either returning the <code>ProblemTypes</code> or converting from a <code>str</code> .
<code>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.

5.8.1 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`

5.8.2 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
```

5.8.3 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.

5.9 Model Family

<i>handle_model_family</i>	Handles model_family by either returning the ModelFamily or converting from a string
<i>ModelFamily</i>	Enum for family of machine learning models.

5.9.1 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

5.9.2 evalml.model_family.ModelFamily

class evalml.model_family.**ModelFamily** (*value*)
Enum for family of machine learning models.

Attributes

BASELINE	Baseline model family.
CATBOOST	CatBoost model family.
DECISION_TREE	Decision Tree model family.
ENSEMBLE	Ensemble model family.
EXTRA_TREES	Extra Trees model family.
LIGHTGBM	LightGBM model family.
LINEAR_MODEL	Linear model family.
NONE	None
RANDOM_FOREST	Random Forest model family.
XGBOOST	XGBoost model family.

5.10 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.

5.10.1 evalml.tuners.Tuner

class evalml.tuners.**Tuner** (*pipeline_hyperparameter_ranges*, *random_state=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_state=0)`
Base Tuner class

Parameters

- **`pipeline_hyperparameter_ranges`** (*dict*) – a set of hyperparameter ranges corresponding to a pipeline’s parameters
- **`random_state`** (*int*, *np.random.RandomState*) – The random state

`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`

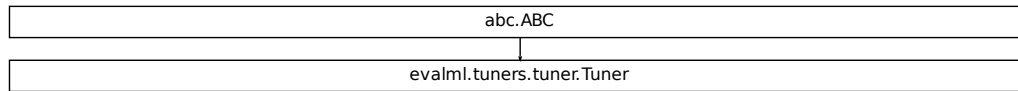
abstract `Tuner.propose()`

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



5.10.2 evalml.tuners.SKOptTuner

class `evalml.tuners.SKOptTuner` (*pipeline_hyperparameter_ranges*, *random_state=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_state=0*)
 Init SKOptTuner

Parameters

- **`pipeline_hyperparameter_ranges`** (*dict*) – a set of hyperparameter ranges corresponding to a pipeline’s parameters
- **`random_state`** (*int*, *np.random.RandomState*) – The random state

`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.

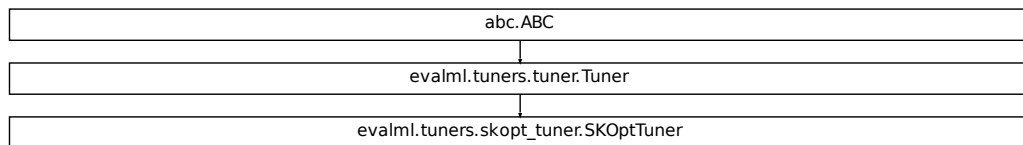
Return type bool

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**5.10.3 evalml.tuners.GridSearchTuner**

class evalml.tuners.**GridSearchTuner** (*pipeline_hyperparameter_ranges, n_points=10, random_state=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_state=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`** – The number of points to sample from along each dimension defined in the `space` argument
- **`random_state`** – Unused in this class

`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

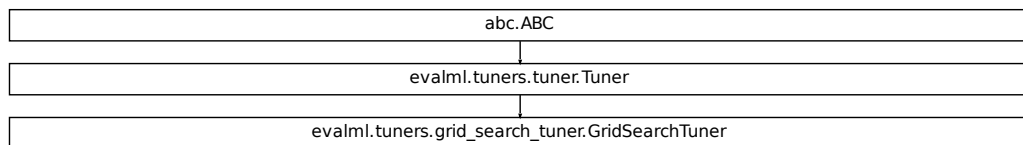
`evalml.tuners.GridSearchTuner.is_search_space_exhausted`

`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` iterationsIf all possible combinations of parameters have been scored, then `NoParamsException` is raised.**Returns** proposed pipeline parameters**Return type** dict**Class Inheritance****5.10.4 evalml.tuners.RandomSearchTuner**

class `evalml.tuners.RandomSearchTuner` (*pipeline_hyperparameter_ranges*, *random_state=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_state=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_state=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** – Unused in this class
- **with_replacement** – If false, only unique hyperparameters will be shown
- **replacement_max_attempts** – The maximum number of tries to get a unique set of random parameters. Only used if tuner is initialized with `with_replacement=True`

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()`

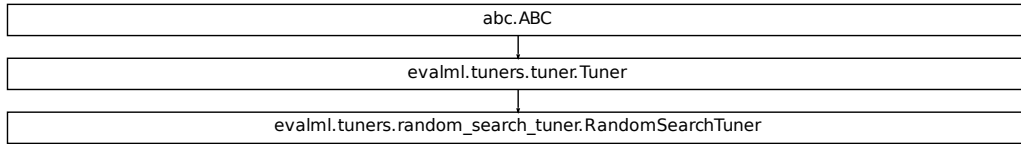
Generate a unique set of parameters.

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



5.11 Data Checks

5.11.1 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.
<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.

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'

Instance attributes

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.

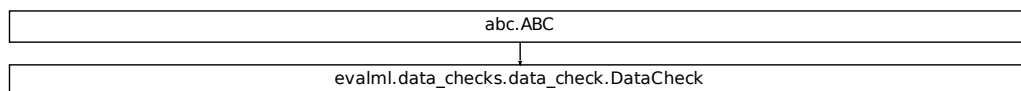
Parameters

- **X** (`pd.DataFrame`) – the input data of shape `[n_samples, n_features]`
- **y** (`pd.Series`, *optional*) – the target data of length `[n_samples]`

Returns Dictionary of `DataCheckError` and `DataCheckWarning` messages

Return type `dict (DataCheckMessage)`

Class Inheritance



evalml.data_checks.InvalidTargetDataCheck

class evalml.data_checks.InvalidTargetDataCheck(*problem_type*, *n_unique*=100)
Checks if the target data contains missing or invalid values.

name = 'InvalidTargetDataCheck'

Instance attributes

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*, *n_unique*=100)
Check if the target is invalid for the specified problem type.

Parameters **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.

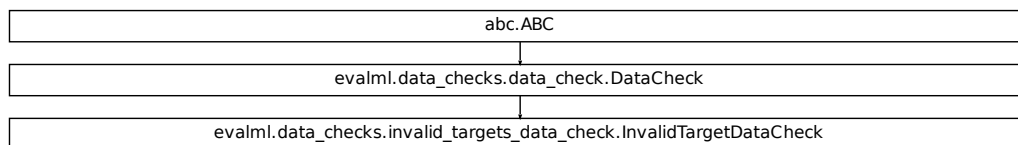
Returns List with DataCheckErrors if any invalid values are found in the target data.

Return type dict (*DataCheckError*)

Example

```
>>> import pandas as pd
>>> X = pd.DataFrame({})
>>> y = pd.Series([0, 1, None, None])
>>> target_check = InvalidTargetDataCheck('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": []}
```

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'

Instance attributes

—

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

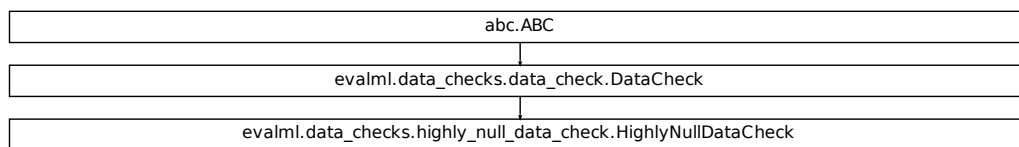
- **X** (*ww.DataTable*, *pd.DataFrame*, *np.ndarray*) – Features
- **y** (*ww.DataColumn*, *pd.Series*, *np.ndarray*) – Ignored.

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"}]}]}
```

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'

Instance attributes**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 not float / boolean)

Parameters **X** (*ww.DataTable, pd.DataFrame, np.ndarray*) – The input features to check

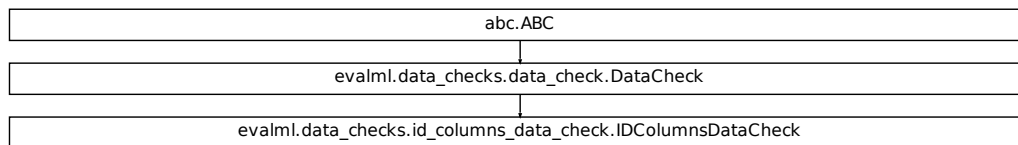
Returns A dictionary of features with column name or index and their probability of being ID columns

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"}}]}
↪
```

Class Inheritance



evalml.data_checks.TargetLeakageDataCheck

class evalml.data_checks.**TargetLeakageDataCheck** (*pct_corr_threshold=0.95*)

Check if any of the features are highly correlated with the target.

name = 'TargetLeakageDataCheck'

Instance attributes

Methods:

<code>__init__</code>	Check if any of the features are highly correlated with the target.
<code>validate</code>	Check if any of the features are highly correlated with the target.

evalml.data_checks.TargetLeakageDataCheck.__init__

`TargetLeakageDataCheck.__init__(pct_corr_threshold=0.95)`

Check if any of the features are highly correlated with the target.

Currently only supports binary and numeric targets and features.

Parameters `pct_corr_threshold` (*float*) – The correlation threshold to be considered leakage. Defaults to 0.95.

evalml.data_checks.TargetLeakageDataCheck.validate

`TargetLeakageDataCheck.validate(X, y)`

Check if any of the features are highly correlated with the target.

Currently only supports binary and numeric targets and features.

Parameters

- **X** (*ww.DataTable*, *pd.DataFrame*, *np.ndarray*) – The input features to check
- **y** (*ww.DataColumn*, *pd.Series*, *np.ndarray*) – The target data

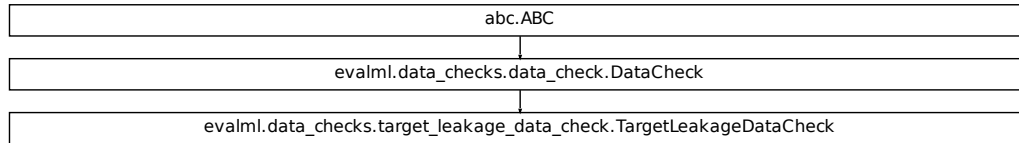
Returns dict with a `DataCheckWarning` if target leakage is detected.

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': [12, 5, 13, 74, 24],
... })
>>> y = pd.Series([10, 42, 31, 51, 40])
>>> target_leakage_check = TargetLeakageDataCheck(pct_corr_threshold=0.8)
>>> assert target_leakage_check.validate(X, y) == {"warnings": [{"message":
↪ "Column 'leak' is 80.0% or more correlated with the target",
↪                                     "data_check_
↪ name": "TargetLeakageDataCheck",
↪                                     "level": "warning",
↪                                     "code": "TARGET_LEAKAGE
↪ ",
↪     "details": {"column": "leak"}},
↪     "errors": []}]}
```

Class Inheritance



evalml.data_checks.OutliersDataCheck

class evalml.data_checks.OutliersDataCheck (*random_state=0*)

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'

Instance attributes

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__** (*random_state=0*)

Checks if there are any outliers in the input data.

Parameters **random_state** (*int, np.random.RandomState*) – The random seed/state. Defaults to 0.

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

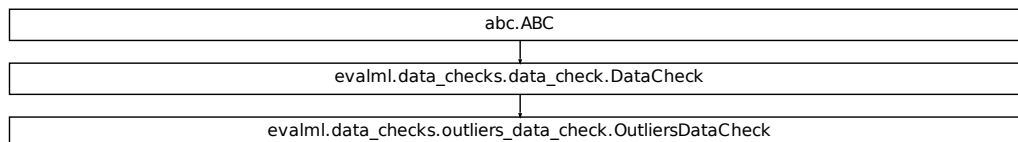
- **X** (`ww.DataTable`, `pd.DataFrame`, `np.ndarray`) – Features
- **y** (`ww.DataColumn`, `pd.Series`, `np.ndarray`) – Ignored.

Returns A dictionary with warnings if any columns have outliers.

Return type dict

Example

```
>>> 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": []}]}
```

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'
```

Instance attributes

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).
```

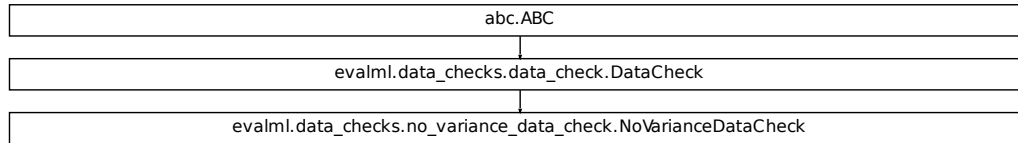
Parameters

- **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, num_cv_folds=3*)
 Checks if any target labels are imbalanced beyond a threshold. Use for classification problems

name = 'ClassImbalanceDataCheck'

Instance attributes

—

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, 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. A perfectly balanced dataset would have a threshold of $(1/n_classes)$, ie 0.50 for binary classes. Defaults to 0.10
- **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 $2 * \text{num_cv_folds}$.

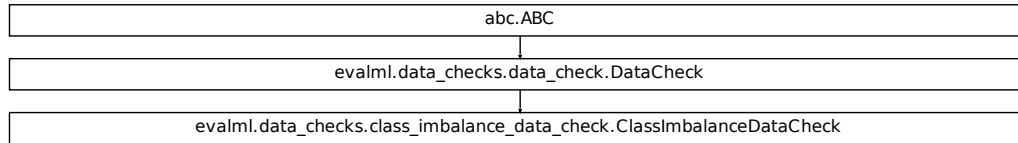
Return type dict

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]}}]}
```

Class Inheritance



<code>DataChecks</code>	A collection of data checks.
<code>DefaultDataChecks</code>	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

<code>__init__</code>	A collection of 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.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*)

A collection of basic data checks that is used by AutoML by default. Includes:

- *HighlyNullDataCheck*
- *IDColumnsDataCheck*
- *TargetLeakageDataCheck*
- *InvalidTargetDataCheck*
- *NoVarianceDataCheck*
- *ClassImbalanceDataCheck* (for classification problem types)

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*)

A collection of basic data checks.

Parameters **problem_type** (*str*) – The problem type that is being validated. Can be regression, binary, or multiclass.

evalml.data_checks.DefaultDataChecks.validate

`DefaultDataChecks.validate(X, y=None)`

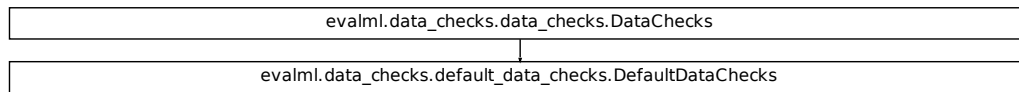
Inspects and validates the input data against data checks and returns a list of warnings and errors if applicable.

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**5.11.2 Data Check Messages**

<i>DataCheckMessage</i>	Base class for all <code>DataCheckMessages</code> .
<i>DataCheckError</i>	<code>DataCheckMessage</code> subclass for errors returned by data checks.
<i>DataCheckWarning</i>	<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 <code>self.message</code> 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__

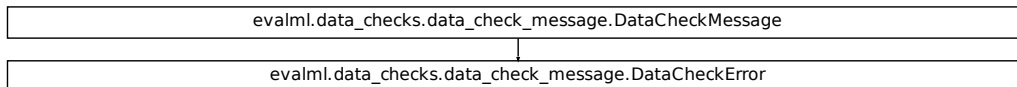
DataCheckError.**__str__**()
 String representation of data check message, equivalent to self.message attribute.

`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 <code>DataCheck</code> , 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

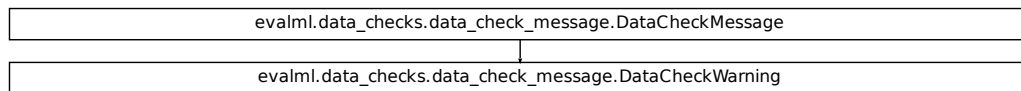
- **message** (*str*) – Message string
- **data_check_name** (*str*) – Name of data check
- **message_code** (`DataCheckMessageCode`, *optional*) – Message code associated with message.
- **details** (*dict*, *optional*) – Additional useful information associated with the message

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**5.11.3 Data Check Message Types***DataCheckMessageType*

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.

5.11.4 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.
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 multi-collinear.
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_LEAKAGE	Message code for when target leakage is detected.
TARGET_UNSUPPORTED_TYPE	Message code for target data that is of an unsupported type.

5.12 Utils

5.12.1 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.
<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 both <code>pd_data_1</code> and <code>pd_data_2</code> .
<code>infer_feature_types</code>	Create a Woodwork structure from the given pandas or numpy input, with specified types for columns.

`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_1, pd_data_2)`

Drop rows that have any NaNs in both `pd_data_1` and `pd_data_2`.

Parameters

- **pd_data_1** (*pd.DataFrame* or *pd.Series*) – Data to subset.
- **pd_data_2** (*pd.DataFrame* or *pd.Series*) – Data to subset.

Returns tuple of *pd.DataFrame* or *pd.Series*

evalml.utils.infer_feature_types

`evalml.utils.infer_feature_types(data, feature_types=None)`

Create a Woodwork structure from the given 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.

RELEASE NOTES

Future Releases

- Enhancements
- Fixes
- Changes
- Documentation Changes
- Testing Changes

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 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 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](#)
- **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 `HighVarianceCVDDataCheck` and replaced synonymous warning in AutoMLSearch [#1254](#)
- Added *PCA Transformer* component for dimensionality reduction [#1270](#)
- Added `generate_pipeline_code` and `generate_component_code` to allow for code generation given a pipeline or component instance [#1306](#)
- Added *PCA Transformer* component for dimensionality reduction [#1270](#)

- Updated `AutoMLSearch` to support Woodwork data structures [#1299](#)
- Added `cv_folds` to `ClassImbalanceDataCheck` and added this check to `DefaultDataChecks` [#1333](#)
- Make `max_batches` argument to `AutoMLSearch.search` public [#1320](#)
- Added text support to automl search [#1062](#)
- Added `_pipelines_per_batch` as a private argument to `AutoMLSearch` [#1355](#)
- **Fixes**
 - Fixed ML performance issue with ordered datasets: always shuffle data in automl’s default CV splits [#1265](#)
 - Fixed broken `evalml info` CLI command [#1293](#)
 - Fixed boosting `type='rf'` for LightGBM Classifier, as well as `num_leaves` error [#1302](#)
 - Fixed bug in `explain_predictions_best_worst` where a custom index in the target variable would cause a `ValueError` [#1318](#)
 - Added stacked ensemble estimators to `evalml.pipelines.__init__` file [#1326](#)
 - Fixed bug in OHE where calls to transform were not deterministic if `top_n` was less than the number of categories in a column [#1324](#)
 - Fixed LightGBM warning messages during `AutoMLSearch` [#1342](#)
 - Fix warnings thrown during `AutoMLSearch` in `HighVarianceCVDataCheck` [#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](#)
- Standarizing inputs as pd.DataFrame / pd.Series [#130](#)
- Enforcing that pipelines must have an estimator as last component [#277](#)
- Added ipywidgets as a dependency in requirements.txt [#278](#)
- Added Random and Grid Search Tuners [#240](#)

- **Documentation Changes**

- Adding class properties to API reference [#244](#)
- Fix and filter FutureWarnings from scikit-learn [#249](#), [#257](#)
- Adding Linear Regression to API reference and cleaning up some Sphinx warnings [#227](#)

- **Testing Changes**

- Added support for testing on Windows with CircleCI [#226](#)
- Added support for doctests [#233](#)

Warning: Breaking Changes

- The `fit()` method for `AutoClassifier` and `AutoRegressor` has been renamed to `search()`.
- `AutoClassifier` has been renamed to `AutoClassificationSearch`
- `AutoRegressor` has been renamed to `AutoRegressionSearch`

- `AutoClassificationSearch.results` and `AutoRegressionSearch.results` now is a dictionary with `pipeline_results` and `search_order` keys. `pipeline_results` can be used to access a dictionary that is identical to the old `.results` dictionary. Whereas, `search_order` returns a list of the search order in terms of `pipeline_id`.
- Pipelines now require an estimator as the last component in `component_list`. Slicing pipelines now throws an `NotImplementedError` to avoid returning pipelines without an estimator.

v0.5.2 Nov. 18, 2019

- **Enhancements**
 - Adding basic pipeline structure visualization #211
- **Documentation Changes**
 - Added notebooks to build process #212

v0.5.1 Nov. 15, 2019

- **Enhancements**
 - Added basic outlier detection guardrail #151
 - Added basic ID column guardrail #135
 - Added support for unlimited pipelines with a `max_time` limit #70
 - Updated `.readthedocs.yaml` to successfully build #188
- **Fixes**
 - Removed MSLE from default additional objectives #203
 - Fixed `random_state` passed in pipelines #204
 - Fixed slow down in `RFRegressor` #206
- **Changes**
 - Pulled information for `describe_pipeline` from pipeline's new `describe` method #190
 - Refactored pipelines #108
 - Removed guardrails from `Auto(*)` #202, #208
- **Documentation Changes**
 - Updated documentation to show `max_time` enhancements #189
 - Updated release instructions for RTD #193
 - Added notebooks to build process #212
 - Added contributing instructions #213
 - Added new content #222

v0.5.0 Oct. 29, 2019

- **Enhancements**
 - Added basic one hot encoding #73
 - Use enums for `model_type` #110
 - Support for splitting regression datasets #112

- Auto-infer multiclass classification [#99](#)
- Added support for other units in `max_time` [#125](#)
- Detect highly null columns [#121](#)
- Added additional regression objectives [#100](#)
- Show an interactive iteration vs. score plot when using `fit()` [#134](#)
- **Fixes**
 - Reordered `describe_pipeline` [#94](#)
 - Added type check for `model_type` [#109](#)
 - Fixed `s` units when setting string `max_time` [#132](#)
 - Fix objectives not appearing in API documentation [#150](#)
- **Changes**
 - Reorganized tests [#93](#)
 - Moved logging to its own module [#119](#)
 - Show progress bar history [#111](#)
 - Using `cloudpickle` instead of `pickle` to allow unloading of custom objectives [#113](#)
 - Removed `render.py` [#154](#)
- **Documentation Changes**
 - Update release instructions [#140](#)
 - Include `additional_objectives` parameter [#124](#)
 - Added Changelog [#136](#)
- **Testing Changes**
 - Code coverage [#90](#)
 - Added CircleCI tests for other Python versions [#104](#)
 - Added doc notebooks as tests [#139](#)
 - Test metadata for CircleCI and 2 core parallelism [#137](#)

v0.4.1 Sep. 16, 2019

- **Enhancements**
 - Added AutoML for classification and regressor using Autobase and Skopt [#7](#) [#9](#)
 - Implemented standard classification and regression metrics [#7](#)
 - Added logistic regression, random forest, and XGBoost pipelines [#7](#)
 - Implemented support for custom objectives [#15](#)
 - Feature importance for pipelines [#18](#)
 - Serialization for pipelines [#19](#)
 - Allow fitting on objectives for optimal threshold [#27](#)
 - Added detect label leakage [#31](#)
 - Implemented callbacks [#42](#)

- Allow for multiclass classification #21
 - Added support for additional objectives #79
- **Fixes**
 - Fixed feature selection in pipelines #13
 - Made `random_seed` usage consistent #45
- **Documentation Changes**
 - Documentation Changes
 - Added docstrings #6
 - Created notebooks for docs #6
 - Initialized readthedocs EvalML #6
 - Added favicon #38
- **Testing Changes**
 - Added testing for loading data #39

v0.2.0 Aug. 13, 2019

- **Enhancements**
 - Created fraud detection objective #4

v0.1.0 July. 31, 2019

- *First Release*
- **Enhancements**
 - Added lead scoring objective #1
 - Added basic classifier #1
- **Documentation Changes**
 - Initialized Sphinx for docs #1

Symbols

`__eq__()` (*evalml.data_checks.DataCheckError* method), 446
`__eq__()` (*evalml.data_checks.DataCheckMessage* method), 444
`__eq__()` (*evalml.data_checks.DataCheckWarning* method), 447
`__init__()` (*evalml.automl.AutoMLSearch* method), 101
`__init__()` (*evalml.automl.automl_algorithm.AutoMLAlgorithm* method), 106
`__init__()` (*evalml.automl.automl_algorithm.IterativeAlgorithm* method), 108
`__init__()` (*evalml.data_checks.ClassImbalanceDataCheck* method), 439
`__init__()` (*evalml.data_checks.DataCheck* method), 429
`__init__()` (*evalml.data_checks.DataCheckError* method), 445
`__init__()` (*evalml.data_checks.DataCheckMessage* method), 444
`__init__()` (*evalml.data_checks.DataCheckWarning* method), 446
`__init__()` (*evalml.data_checks.DataChecks* method), 441
`__init__()` (*evalml.data_checks.DefaultDataChecks* method), 442
`__init__()` (*evalml.data_checks.HighlyNullDataCheck* method), 432
`__init__()` (*evalml.data_checks.IDColumnsDataCheck* method), 433
`__init__()` (*evalml.data_checks.InvalidTargetDataCheck* method), 430
`__init__()` (*evalml.data_checks.NoVarianceDataCheck* method), 438
`__init__()` (*evalml.data_checks.OutliersDataCheck* method), 436
`__init__()` (*evalml.data_checks.TargetLeakageDataCheck* method), 435
`__init__()` (*evalml.objectives.AUC* method), 344
`__init__()` (*evalml.objectives.AUCMacro* method), 347
`__init__()` (*evalml.objectives.AUCMicro* method), 350
`__init__()` (*evalml.objectives.AUCWeighted* method), 352
`__init__()` (*evalml.objectives.AccuracyBinary* method), 339
`__init__()` (*evalml.objectives.AccuracyMulticlass* method), 342
`__init__()` (*evalml.objectives.BalancedAccuracyBinary* method), 354
`__init__()` (*evalml.objectives.BalancedAccuracyMulticlass* method), 357
`__init__()` (*evalml.objectives.BinaryClassificationObjective* method), 321
`__init__()` (*evalml.objectives.CostBenefitMatrix* method), 335
`__init__()` (*evalml.objectives.ExpVariance* method), 412
`__init__()` (*evalml.objectives.F1* method), 359
`__init__()` (*evalml.objectives.F1Macro* method), 364
`__init__()` (*evalml.objectives.F1Micro* method), 362
`__init__()` (*evalml.objectives.F1Weighted* method), 366
`__init__()` (*evalml.objectives.FraudCost* method), 329
`__init__()` (*evalml.objectives.LeadScoring* method), 332
`__init__()` (*evalml.objectives.LogLossBinary* method), 368
`__init__()` (*evalml.objectives.LogLossMulticlass* method), 371
`__init__()` (*evalml.objectives.MAE* method), 398
`__init__()` (*evalml.objectives.MAPE* method), 401
`__init__()` (*evalml.objectives.MCCBinary* method), 373
`__init__()` (*evalml.objectives.MCCMulticlass* method), 376
`__init__()` (*evalml.objectives.MSE* method), 403
`__init__()` (*evalml.objectives.MaxError* method), 410
`__init__()` (*evalml.objectives.MeanSquaredLogError* method), 405

[__init__\(\) \(evalml.objectives.MedianAE method\), 407](#)
[__init__\(\) \(evalml.objectives.MulticlassClassificationObjective method\), 324](#)
[__init__\(\) \(evalml.objectives.ObjectiveBase method\), 319](#)
[__init__\(\) \(evalml.objectives.Precision method\), 378](#)
[__init__\(\) \(evalml.objectives.PrecisionMacro method\), 383](#)
[__init__\(\) \(evalml.objectives.PrecisionMicro method\), 381](#)
[__init__\(\) \(evalml.objectives.PrecisionWeighted method\), 385](#)
[__init__\(\) \(evalml.objectives.R2 method\), 396](#)
[__init__\(\) \(evalml.objectives.Recall method\), 387](#)
[__init__\(\) \(evalml.objectives.RecallMacro method\), 392](#)
[__init__\(\) \(evalml.objectives.RecallMicro method\), 390](#)
[__init__\(\) \(evalml.objectives.RecallWeighted method\), 394](#)
[__init__\(\) \(evalml.objectives.RegressionObjective method\), 327](#)
[__init__\(\) \(evalml.objectives.RootMeanSquaredError method\), 414](#)
[__init__\(\) \(evalml.objectives.RootMeanSquaredLogError method\), 416](#)
[__init__\(\) \(evalml.pipelines.BaselineBinaryPipeline method\), 152](#)
[__init__\(\) \(evalml.pipelines.BaselineMulticlassPipeline method\), 157](#)
[__init__\(\) \(evalml.pipelines.BaselineRegressionPipeline method\), 171](#)
[__init__\(\) \(evalml.pipelines.BinaryClassificationPipeline method\), 120](#)
[__init__\(\) \(evalml.pipelines.ClassificationPipeline method\), 116](#)
[__init__\(\) \(evalml.pipelines.MeanBaselineRegressionPipeline method\), 176](#)
[__init__\(\) \(evalml.pipelines.ModeBaselineBinaryPipeline method\), 161](#)
[__init__\(\) \(evalml.pipelines.ModeBaselineMulticlassPipeline method\), 166](#)
[__init__\(\) \(evalml.pipelines.MulticlassClassificationPipeline method\), 125](#)
[__init__\(\) \(evalml.pipelines.PipelineBase method\), 112](#)
[__init__\(\) \(evalml.pipelines.RegressionPipeline method\), 129](#)
[__init__\(\) \(evalml.pipelines.TimeSeriesBaselineRegressionPipeline method\), 180](#)
[__init__\(\) \(evalml.pipelines.TimeSeriesBinaryClassificationPipeline method\), 138](#)
[__init__\(\) \(evalml.pipelines.TimeSeriesClassificationPipeline method\), 133](#)
[__init__\(\) \(evalml.pipelines.TimeSeriesMulticlassClassificationPipeline method\), 143](#)
[__init__\(\) \(evalml.pipelines.TimeSeriesRegressionPipeline method\), 147](#)
[__init__\(\) \(evalml.pipelines.components.BaselineClassifier method\), 265](#)
[__init__\(\) \(evalml.pipelines.components.BaselineRegressor method\), 293](#)
[__init__\(\) \(evalml.pipelines.components.CatBoostClassifier method\), 244](#)
[__init__\(\) \(evalml.pipelines.components.CatBoostRegressor method\), 275](#)
[__init__\(\) \(evalml.pipelines.components.ComponentBase method\), 186](#)
[__init__\(\) \(evalml.pipelines.components.DFSTransformer method\), 241](#)
[__init__\(\) \(evalml.pipelines.components.DateTimeFeaturizer method\), 231](#)
[__init__\(\) \(evalml.pipelines.components.DecisionTreeClassifier method\), 271](#)
[__init__\(\) \(evalml.pipelines.components.DecisionTreeRegressor method\), 302](#)
[__init__\(\) \(evalml.pipelines.components.DelayedFeatureTransformer method\), 237](#)
[__init__\(\) \(evalml.pipelines.components.DropColumns method\), 195](#)
[__init__\(\) \(evalml.pipelines.components.DropNullColumns method\), 228](#)
[__init__\(\) \(evalml.pipelines.components.ElasticNetClassifier method\), 247](#)
[__init__\(\) \(evalml.pipelines.components.ElasticNetRegressor method\), 278](#)
[__init__\(\) \(evalml.pipelines.components.Estimator method\), 191](#)
[__init__\(\) \(evalml.pipelines.components.ExtraTreesClassifier method\), 250](#)
[__init__\(\) \(evalml.pipelines.components.ExtraTreesRegressor method\), 284](#)
[__init__\(\) \(evalml.pipelines.components.Imputer method\), 212](#)
[__init__\(\) \(evalml.pipelines.components.LightGBMClassifier method\), 256](#)
[__init__\(\) \(evalml.pipelines.components.LightGBMRegressor method\), 306](#)
[__init__\(\) \(evalml.pipelines.components.LinearRegressor method\), 281](#)
[__init__\(\) \(evalml.pipelines.components.LogisticRegressionClassifier method\), 259](#)
[__init__\(\) \(evalml.pipelines.components.OneHotEncoder method\), 201](#)
[__init__\(\) \(evalml.pipelines.components.PerColumnImputer method\), 209](#)
[__init__\(\) \(evalml.pipelines.components.RFClassifierSelectFromModel method\), 133](#)

`method`), 225
`__init__` () (`evalml.pipelines.components.RFRegressorSelectFromModel` method), 109
`method`), 222
`__init__` () (`evalml.pipelines.components.RandomForestClassifier` method), 103
`method`), 253
`__init__` () (`evalml.pipelines.components.RandomForestRegressor` method), 193
`method`), 287
`__init__` () (`evalml.pipelines.components.SelectColumns` method), 198
`method`), 198
`__init__` () (`evalml.pipelines.components.SimpleImputer` method), 215
`method`), 215
`__init__` () (`evalml.pipelines.components.StackedEnsembleClassifier` method), 268
`method`), 268
`__init__` () (`evalml.pipelines.components.StackedEnsembleRegressor` method), 299
`method`), 299
`__init__` () (`evalml.pipelines.components.StandardScaler` method), 218
`method`), 218
`__init__` () (`evalml.pipelines.components.TargetEncoder` method), 205
`method`), 205
`__init__` () (`evalml.pipelines.components.TextFeaturizer` method), 234
`method`), 234
`__init__` () (`evalml.pipelines.components.TimeSeriesBaselineRegressor` method), 296
`method`), 296
`__init__` () (`evalml.pipelines.components.Transformer` method), 188
`method`), 188
`__init__` () (`evalml.pipelines.components.XGBoostClassifier` method), 262
`method`), 262
`__init__` () (`evalml.pipelines.components.XGBoostRegressor` method), 290
`method`), 290
`__init__` () (`evalml.tuners.GridSearchTuner` method), 425
`method`), 425
`__init__` () (`evalml.tuners.RandomSearchTuner` method), 427
`method`), 427
`__init__` () (`evalml.tuners.SKOptTuner` method), 423
`method`), 423
`__init__` () (`evalml.tuners.Tuner` method), 422
`method`), 422
`__str__` () (`evalml.data_checks.DataCheckError` method), 445
`method`), 445
`__str__` () (`evalml.data_checks.DataCheckMessage` method), 444
`method`), 444
`__str__` () (`evalml.data_checks.DataCheckWarning` method), 447
`method`), 447

A

`AccuracyBinary` (class in `evalml.objectives`), 339
`AccuracyMulticlass` (class in `evalml.objectives`), 342
`add` () (`evalml.tuners.GridSearchTuner` method), 425
`add` () (`evalml.tuners.RandomSearchTuner` method), 427
`add` () (`evalml.tuners.SKOptTuner` method), 423
`add` () (`evalml.tuners.Tuner` method), 422
`add_result` () (`evalml.automl.automl_algorithm.AutoMLAlgorithm` method), 107
`add_result` () (`evalml.automl.automl_algorithm.IterativeAlgorithm` method), 107
`add_to_rankings` () (`evalml.automl.AutoMLSearch` method), 100
`allowed_model_families` () (in module `evalml.pipelines.components.utils`), 193
`AUC` (class in `evalml.objectives`), 344
`AUCMacro` (class in `evalml.objectives`), 347
`AUCMicro` (class in `evalml.objectives`), 349
`AUCWeighted` (class in `evalml.objectives`), 351
`AutoMLAlgorithm` (class in `evalml.automl.automl_algorithm`), 106
`AutoMLSearch` (class in `evalml.automl`), 100

B

`BalancedAccuracyBinary` (class in `evalml.objectives`), 353
`BalancedAccuracyMulticlass` (class in `evalml.objectives`), 356
`BaselineBinaryPipeline` (class in `evalml.pipelines`), 151
`BaselineBinaryClassifier` (class in `evalml.pipelines.components`), 264
`BaselineMulticlassPipeline` (class in `evalml.pipelines`), 156
`BaselineRegressionPipeline` (class in `evalml.pipelines`), 170
`BaselineRegressor` (class in `evalml.pipelines.components`), 292
`binary_objective_vs_threshold` () (in module `evalml.model_understanding`), 311
`BinaryClassificationObjective` (class in `evalml.objectives`), 321
`BinaryClassificationPipeline` (class in `evalml.pipelines`), 119

C

`calculate_percent_difference` () (`evalml.objectives.AccuracyBinary` class method), 339
`calculate_percent_difference` () (`evalml.objectives.AccuracyMulticlass` class method), 342
`calculate_percent_difference` () (`evalml.objectives.AUC` class method), 345
`calculate_percent_difference` () (`evalml.objectives.AUCMacro` class method), 348
`calculate_percent_difference` () (`evalml.objectives.AUCMicro` class method), 350
`calculate_percent_difference` () (`evalml.objectives.AUCWeighted` class method), 352

`calculate_percent_difference()`
 (*evalml.objectives.BalancedAccuracyBinary*
 class method), 354
`calculate_percent_difference()`
 (*evalml.objectives.BalancedAccuracyMulticlass*
 class method), 357
`calculate_percent_difference()`
 (*evalml.objectives.BinaryClassificationObjective*
 class method), 322
`calculate_percent_difference()`
 (*evalml.objectives.CostBenefitMatrix* *class*
 method), 336
`calculate_percent_difference()`
 (*evalml.objectives.ExpVariance* *class method*),
 412
`calculate_percent_difference()`
 (*evalml.objectives.F1* *class method*), 359
`calculate_percent_difference()`
 (*evalml.objectives.F1Macro* *class method*),
 364
`calculate_percent_difference()`
 (*evalml.objectives.F1Micro* *class method*),
 362
`calculate_percent_difference()`
 (*evalml.objectives.F1Weighted* *class method*),
 366
`calculate_percent_difference()`
 (*evalml.objectives.FraudCost* *class method*),
 330
`calculate_percent_difference()`
 (*evalml.objectives.LeadScoring* *class method*),
 333
`calculate_percent_difference()`
 (*evalml.objectives.LogLossBinary* *class*
 method), 368
`calculate_percent_difference()`
 (*evalml.objectives.LogLossMulticlass* *class*
 method), 371
`calculate_percent_difference()`
 (*evalml.objectives.MAE* *class method*), 399
`calculate_percent_difference()`
 (*evalml.objectives.MAPE* *class method*),
 401
`calculate_percent_difference()`
 (*evalml.objectives.MaxError* *class method*),
 410
`calculate_percent_difference()`
 (*evalml.objectives.MCCBinary* *class method*),
 373
`calculate_percent_difference()`
 (*evalml.objectives.MCCMulticlass* *class*
 method), 376
`calculate_percent_difference()`
 (*evalml.objectives.MeanSquaredLogError*
 class method), 405
`calculate_percent_difference()`
 (*evalml.objectives.MedianAE* *class method*),
 408
`calculate_percent_difference()`
 (*evalml.objectives.MSE* *class method*), 403
`calculate_percent_difference()`
 (*evalml.objectives.MulticlassClassificationObjective*
 class method), 325
`calculate_percent_difference()`
 (*evalml.objectives.ObjectiveBase* *class*
 method), 319
`calculate_percent_difference()`
 (*evalml.objectives.Precision* *class method*),
 378
`calculate_percent_difference()`
 (*evalml.objectives.PrecisionMacro* *class*
 method), 383
`calculate_percent_difference()`
 (*evalml.objectives.PrecisionMicro* *class*
 method), 381
`calculate_percent_difference()`
 (*evalml.objectives.PrecisionWeighted* *class*
 method), 385
`calculate_percent_difference()`
 (*evalml.objectives.R2* *class method*), 396
`calculate_percent_difference()`
 (*evalml.objectives.Recall* *class method*),
 387
`calculate_percent_difference()`
 (*evalml.objectives.RecallMacro* *class method*),
 392
`calculate_percent_difference()`
 (*evalml.objectives.RecallMicro* *class method*),
 390
`calculate_percent_difference()`
 (*evalml.objectives.RecallWeighted* *class*
 method), 394
`calculate_percent_difference()`
 (*evalml.objectives.RegressionObjective* *class*
 method), 327
`calculate_percent_difference()`
 (*evalml.objectives.RootMeanSquaredError*
 class method), 414
`calculate_percent_difference()`
 (*evalml.objectives.RootMeanSquaredLogError*
 class method), 416
`calculate_permutation_importance()` (in
 module evalml.model_understanding), 310
`CatBoostClassifier` (*class* in
 evalml.pipelines.components), 243
`CatBoostRegressor` (*class* in
 evalml.pipelines.components), 274
`categories()` (*evalml.pipelines.components.OneHotEncoder*

method), 202

ClassificationPipeline (class in clone() (evalml.pipelines.components.LinearRegressor method), 281

evalml.pipelines), 115

ClassImbalanceDataCheck (class in clone() (evalml.pipelines.components.LogisticRegressionClassifier method), 259

evalml.data_checks), 439

clone() (evalml.pipelines.BaselineBinaryPipeline method), 152

clone() (evalml.pipelines.BaselineMulticlassPipeline method), 157

clone() (evalml.pipelines.BaselineRegressionPipeline method), 172

clone() (evalml.pipelines.BinaryClassificationPipeline method), 120

clone() (evalml.pipelines.ClassificationPipeline method), 116

clone() (evalml.pipelines.components.BaselineClassifier method), 265

clone() (evalml.pipelines.components.BaselineRegressor method), 293

clone() (evalml.pipelines.components.CatBoostClassifier method), 244

clone() (evalml.pipelines.components.CatBoostRegressor method), 275

clone() (evalml.pipelines.components.ComponentBase method), 186

clone() (evalml.pipelines.components.DateTimeFeaturizer method), 231

clone() (evalml.pipelines.components.DecisionTreeClassifier method), 272

clone() (evalml.pipelines.components.DecisionTreeRegressor method), 303

clone() (evalml.pipelines.components.DelayedFeatureTransformer method), 238

clone() (evalml.pipelines.components.DFSTransformer method), 241

clone() (evalml.pipelines.components.DropColumns method), 196

clone() (evalml.pipelines.components.DropNullColumns method), 228

clone() (evalml.pipelines.components.ElasticNetClassifier method), 247

clone() (evalml.pipelines.components.ElasticNetRegressor method), 278

clone() (evalml.pipelines.components.Estimator method), 191

clone() (evalml.pipelines.components.ExtraTreesClassifier method), 250

clone() (evalml.pipelines.components.ExtraTreesRegressor method), 284

clone() (evalml.pipelines.components.Imputer method), 213

clone() (evalml.pipelines.components.LightGBMClassifier method), 256

clone() (evalml.pipelines.components.LightGBMRegressor method), 306

clone() (evalml.pipelines.components.LogisticRegressionClassifier method), 259

clone() (evalml.pipelines.components.OneHotEncoder method), 202

clone() (evalml.pipelines.components.PerColumnImputer method), 209

clone() (evalml.pipelines.components.RandomForestClassifier method), 253

clone() (evalml.pipelines.components.RandomForestRegressor method), 287

clone() (evalml.pipelines.components.RFClassifierSelectFromModel method), 225

clone() (evalml.pipelines.components.RFRegressorSelectFromModel method), 222

clone() (evalml.pipelines.components.SelectColumns method), 199

clone() (evalml.pipelines.components.SimpleImputer method), 216

clone() (evalml.pipelines.components.StackedEnsembleClassifier method), 269

clone() (evalml.pipelines.components.StackedEnsembleRegressor method), 300

clone() (evalml.pipelines.components.StandardScaler method), 218

clone() (evalml.pipelines.components.TargetEncoder method), 206

clone() (evalml.pipelines.components.TextFeaturizer method), 235

clone() (evalml.pipelines.components.TimeSeriesBaselineRegressor method), 296

clone() (evalml.pipelines.components.Transformer method), 188

clone() (evalml.pipelines.components.XGBoostClassifier method), 262

clone() (evalml.pipelines.components.XGBoostRegressor method), 290

clone() (evalml.pipelines.MeanBaselineRegressionPipeline method), 176

clone() (evalml.pipelines.ModeBaselineBinaryPipeline method), 162

clone() (evalml.pipelines.ModeBaselineMulticlassPipeline method), 166

clone() (evalml.pipelines.MulticlassClassificationPipeline method), 125

clone() (evalml.pipelines.PipelineBase method), 112

clone() (evalml.pipelines.RegressionPipeline method), 130

clone() (evalml.pipelines.TimeSeriesBaselineRegressionPipeline method), 180

clone() (evalml.pipelines.TimeSeriesBinaryClassificationPipeline method), 138

<code>clone()</code> (<i>evalml.pipelines.TimeSeriesClassificationPipeline</i> method), 134	<code>compute_estimator_features()</code> (<i>evalml.pipelines.TimeSeriesBaselineRegressionPipeline</i> method), 181
<code>clone()</code> (<i>evalml.pipelines.TimeSeriesMulticlassClassificationPipeline</i> method), 143	<code>compute_estimator_features()</code> (<i>evalml.pipelines.TimeSeriesBinaryClassificationPipeline</i> method), 138
<code>clone()</code> (<i>evalml.pipelines.TimeSeriesRegressionPipeline</i> method), 148	<code>compute_estimator_features()</code> (<i>evalml.pipelines.TimeSeriesClassificationPipeline</i> method), 134
<code>component_graph</code> (<i>evalml.pipelines.BaselineBinaryPipeline</i> attribute), 151	<code>compute_estimator_features()</code> (<i>evalml.pipelines.TimeSeriesMulticlassClassificationPipeline</i> method), 143
<code>component_graph</code> (<i>evalml.pipelines.BaselineMulticlassPipeline</i> attribute), 156	<code>compute_estimator_features()</code> (<i>evalml.pipelines.TimeSeriesRegressionPipeline</i> method), 148
<code>component_graph</code> (<i>evalml.pipelines.BaselineRegressionPipeline</i> attribute), 170	<code>compute_estimator_features()</code> (<i>evalml.pipelines.TimeSeriesBaselineRegressionPipeline</i> method), 179
<code>component_graph</code> (<i>evalml.pipelines.ModeBaselineBinaryPipeline</i> attribute), 160	<code>compute_estimator_features()</code> (<i>evalml.pipelines.TimeSeriesBaselineRegressionPipeline</i> method), 179
<code>component_graph</code> (<i>evalml.pipelines.ModeBaselineMulticlassPipeline</i> attribute), 165	<code>compute_estimator_features()</code> (<i>evalml.pipelines.TimeSeriesBaselineRegressionPipeline</i> method), 179
<code>component_graph</code> (<i>evalml.pipelines.TimeSeriesBaselineRegressionPipeline</i> attribute), 179	<code>compute_estimator_features()</code> (<i>evalml.pipelines.TimeSeriesBaselineRegressionPipeline</i> method), 179
<code>ComponentBase</code> (class in <i>CostBenefitMatrix</i> (class in <i>evalml.objectives</i>), <i>evalml.pipelines.components</i>), 185	<code>compute_estimator_features()</code> (<i>evalml.pipelines.BaselineBinaryPipeline</i> method), 152
<code>compute_estimator_features()</code> (<i>evalml.pipelines.BaselineBinaryPipeline</i> method), 152	<code>compute_estimator_features()</code> (<i>evalml.pipelines.BaselineMulticlassPipeline</i> method), 157
<code>compute_estimator_features()</code> (<i>evalml.pipelines.BaselineMulticlassPipeline</i> method), 157	<code>compute_estimator_features()</code> (<i>evalml.pipelines.BaselineRegressionPipeline</i> method), 172
<code>compute_estimator_features()</code> (<i>evalml.pipelines.BaselineRegressionPipeline</i> method), 172	<code>compute_estimator_features()</code> (<i>evalml.pipelines.BinaryClassificationPipeline</i> method), 121
<code>compute_estimator_features()</code> (<i>evalml.pipelines.BinaryClassificationPipeline</i> method), 121	<code>compute_estimator_features()</code> (<i>evalml.pipelines.ClassificationPipeline</i> method), 116
<code>compute_estimator_features()</code> (<i>evalml.pipelines.ClassificationPipeline</i> method), 116	<code>compute_estimator_features()</code> (<i>evalml.pipelines.ModeBaselineBinaryPipeline</i> method), 162
<code>compute_estimator_features()</code> (<i>evalml.pipelines.ModeBaselineBinaryPipeline</i> method), 162	<code>compute_estimator_features()</code> (<i>evalml.pipelines.ModeBaselineMulticlassPipeline</i> method), 165
<code>compute_estimator_features()</code> (<i>evalml.pipelines.ModeBaselineMulticlassPipeline</i> method), 165	<code>compute_estimator_features()</code> (<i>evalml.pipelines.TimeSeriesBaselineRegressionPipeline</i> method), 179
<code>compute_estimator_features()</code> (<i>evalml.pipelines.MulticlassClassificationPipeline</i> method), 125	<code>compute_estimator_features()</code> (<i>evalml.pipelines.BaselineBinaryPipeline</i> method), 151
<code>compute_estimator_features()</code> (<i>evalml.pipelines.PipelineBase</i> method), 112	<code>compute_estimator_features()</code> (<i>evalml.pipelines.BaselineMulticlassPipeline</i> method), 156
<code>compute_estimator_features()</code> (<i>evalml.pipelines.RegressionPipeline</i> method), 112	<code>compute_estimator_features()</code> (<i>evalml.pipelines.BaselineRegressionPipeline</i> method), 170
<code>compute_estimator_features()</code> (<i>evalml.pipelines.RegressionPipeline</i> method), 112	<code>compute_estimator_features()</code> (<i>evalml.pipelines.BinaryClassificationPipeline</i> method), 119
<code>compute_estimator_features()</code> (<i>evalml.pipelines.RegressionPipeline</i> method), 112	<code>compute_estimator_features()</code> (<i>evalml.pipelines.ClassificationPipeline</i> method), 115
<code>compute_estimator_features()</code> (<i>evalml.pipelines.RegressionPipeline</i> method), 112	<code>compute_estimator_features()</code> (<i>evalml.pipelines.MeanBaselineRegressionPipeline</i> method), 175

[attribute](#)), 175
[custom_name \(evalml.pipelines.ModeBaselineBinaryPipeline attribute\)](#), 160
[custom_name \(evalml.pipelines.ModeBaselineMulticlassPipeline attribute\)](#), 165
[custom_name \(evalml.pipelines.MulticlassClassificationPipeline attribute\)](#), 124
[custom_name \(evalml.pipelines.PipelineBase attribute\)](#), 111
[custom_name \(evalml.pipelines.RegressionPipeline attribute\)](#), 128
[custom_name \(evalml.pipelines.TimeSeriesBaselineRegressionPipeline attribute\)](#), 179
[custom_name \(evalml.pipelines.TimeSeriesBinaryClassificationPipeline attribute\)](#), 137
[custom_name \(evalml.pipelines.TimeSeriesClassificationPipeline attribute\)](#), 133
[custom_name \(evalml.pipelines.TimeSeriesMulticlassClassificationPipeline attribute\)](#), 142
[custom_name \(evalml.pipelines.TimeSeriesRegressionPipeline attribute\)](#), 146
[decision_function\(\)](#)
[decision_function\(\) \(evalml.objectives.LeadScoring method\)](#), 333
[decision_function\(\) \(evalml.objectives.LogLossBinary method\)](#), 368
[decision_function\(\) \(evalml.objectives.MCCBinary method\)](#), 373
[decision_function\(\) \(evalml.objectives.Precision method\)](#), 378
[decision_function\(\) \(evalml.objectives.Recall method\)](#), 387
[DecisionTreeClassifier \(class in evalml.pipelines.components\)](#), 271
[DecisionTreeRegressor \(class in evalml.pipelines.components\)](#), 302
[default_parameters \(evalml.pipelines.BaselineBinaryPipeline attribute\)](#), 151
[default_parameters \(evalml.pipelines.BaselineMulticlassPipeline attribute\)](#), 156
[default_parameters \(evalml.pipelines.BaselineRegressionPipeline attribute\)](#), 170
[default_parameters \(evalml.pipelines.components.BaselineClassifier attribute\)](#), 264
[default_parameters \(evalml.pipelines.components.BaselineRegressor attribute\)](#), 292
[default_parameters \(evalml.pipelines.components.CatBoostClassifier attribute\)](#), 243
[default_parameters \(evalml.pipelines.components.CatBoostRegressor attribute\)](#), 274
[default_parameters \(evalml.pipelines.components.DateTimeFeaturizer attribute\)](#), 230
[default_parameters \(evalml.pipelines.components.DecisionTreeClassifier attribute\)](#), 271
[default_parameters \(evalml.pipelines.components.DecisionTreeRegressor attribute\)](#), 302
[default_parameters \(evalml.pipelines.components.DelayedFeatureTransformer attribute\)](#), 237
[default_parameters \(evalml.pipelines.components.DFSTransformer attribute\)](#), 240
[default_parameters](#)
D
[DataCheck \(class in evalml.data_checks\)](#), 428
[DataCheckError \(class in evalml.data_checks\)](#), 445
[DataCheckMessage \(class in evalml.data_checks\)](#), 443
[DataCheckMessageCode \(class in evalml.data_checks\)](#), 448
[DataCheckMessageType \(class in evalml.data_checks\)](#), 447
[DataChecks \(class in evalml.data_checks\)](#), 441
[DataCheckWarning \(class in evalml.data_checks\)](#), 446
[DateTimeFeaturizer \(class in evalml.pipelines.components\)](#), 230
[decision_function\(\) \(evalml.objectives.AccuracyBinary method\)](#), 340
[decision_function\(\) \(evalml.objectives.AUC method\)](#), 345
[decision_function\(\) \(evalml.objectives.BalancedAccuracyBinary method\)](#), 354
[decision_function\(\) \(evalml.objectives.BinaryClassificationObjective method\)](#), 322
[decision_function\(\) \(evalml.objectives.CostBenefitMatrix method\)](#), 336
[decision_function\(\) \(evalml.objectives.F1 method\)](#), 359
[decision_function\(\) \(evalml.objectives.FraudCost method\)](#), 330

<code>(evalml.pipelines.components.DropColumns attribute), 195</code>	<code>(evalml.pipelines.components.SimpleImputer attribute), 215</code>
<code>default_parameters (evalml.pipelines.components.DropNullColumns attribute), 227</code>	<code>default_parameters (evalml.pipelines.components.StackedEnsembleClassifier attribute), 268</code>
<code>default_parameters (evalml.pipelines.components.ElasticNetClassifier attribute), 246</code>	<code>default_parameters (evalml.pipelines.components.StackedEnsembleRegressor attribute), 299</code>
<code>default_parameters (evalml.pipelines.components.ElasticNetRegressor attribute), 277</code>	<code>default_parameters (evalml.pipelines.components.StandardScaler attribute), 218</code>
<code>default_parameters (evalml.pipelines.components.ExtraTreesClassifier attribute), 249</code>	<code>default_parameters (evalml.pipelines.components.TargetEncoder attribute), 204</code>
<code>default_parameters (evalml.pipelines.components.ExtraTreesRegressor attribute), 283</code>	<code>default_parameters (evalml.pipelines.components.TextFeaturizer attribute), 234</code>
<code>default_parameters (evalml.pipelines.components.Imputer attribute), 211</code>	<code>default_parameters (evalml.pipelines.components.TimeSeriesBaselineRegressor attribute), 295</code>
<code>default_parameters (evalml.pipelines.components.LightGBMClassifier attribute), 255</code>	<code>default_parameters (evalml.pipelines.components.XGBoostClassifier attribute), 261</code>
<code>default_parameters (evalml.pipelines.components.LightGBMRegressor attribute), 305</code>	<code>default_parameters (evalml.pipelines.components.XGBoostRegressor attribute), 289</code>
<code>default_parameters (evalml.pipelines.components.LinearRegressor attribute), 280</code>	<code>default_parameters (evalml.pipelines.MeanBaselineRegressionPipeline attribute), 175</code>
<code>default_parameters (evalml.pipelines.components.LogisticRegressionClassifier attribute), 258</code>	<code>default_parameters (evalml.pipelines.ModeBaselineBinaryPipeline attribute), 160</code>
<code>default_parameters (evalml.pipelines.components.OneHotEncoder attribute), 201</code>	<code>default_parameters (evalml.pipelines.ModeBaselineMulticlassPipeline attribute), 165</code>
<code>default_parameters (evalml.pipelines.components.PerColumnImputer attribute), 208</code>	<code>default_parameters (evalml.pipelines.TimeSeriesBaselineRegressionPipeline attribute), 179</code>
<code>default_parameters (evalml.pipelines.components.RandomForestClassifier attribute), 252</code>	<code>DefaultDataChecks (class in evalml.data_checks), 442</code>
<code>default_parameters (evalml.pipelines.components.RandomForestRegressor attribute), 286</code>	<code>DelayedFeatureTransformer (class in evalml.pipelines.components), 237</code>
<code>default_parameters (evalml.pipelines.components.RFClassifierSelectFromModel attribute), 224</code>	<code>describe() (evalml.pipelines.BaselineBinaryPipeline method), 153</code>
<code>default_parameters (evalml.pipelines.components.RFRegressorSelectFromModel attribute), 221</code>	<code>describe() (evalml.pipelines.BaselineMulticlassPipeline method), 157</code>
<code>default_parameters (evalml.pipelines.components.SelectColumns attribute), 198</code>	<code>describe() (evalml.pipelines.BaselineRegressionPipeline method), 172</code>
<code>default_parameters</code>	<code>describe() (evalml.pipelines.BinaryClassificationPipeline method), 121</code>
	<code>describe() (evalml.pipelines.ClassificationPipeline method), 116</code>
	<code>describe() (evalml.pipelines.components.BaselineClassifier method), 266</code>

- 450
- `DropColumns` (class in `evalml.pipelines.components`), 195
- `DropNullColumns` (class in `evalml.pipelines.components`), 227
- ## E
- `ElasticNetClassifier` (class in `evalml.pipelines.components`), 246
- `ElasticNetRegressor` (class in `evalml.pipelines.components`), 277
- `Estimator` (class in `evalml.pipelines.components`), 190
- `explain_prediction()` (in module `evalml.model_understanding.prediction_explanations`), 316
- `explain_predictions()` (in module `evalml.model_understanding.prediction_explanations`), 317
- `explain_predictions_best_worst()` (in module `evalml.model_understanding.prediction_explanations`), 318
- `ExpVariance` (class in `evalml.objectives`), 411
- `ExtraTreesClassifier` (class in `evalml.pipelines.components`), 249
- `ExtraTreesRegressor` (class in `evalml.pipelines.components`), 283
- ## F
- `F1` (class in `evalml.objectives`), 358
- `F1Macro` (class in `evalml.objectives`), 363
- `F1Micro` (class in `evalml.objectives`), 361
- `F1Weighted` (class in `evalml.objectives`), 365
- `fit()` (`evalml.pipelines.BaselineBinaryPipeline` method), 153
- `fit()` (`evalml.pipelines.BaselineMulticlassPipeline` method), 158
- `fit()` (`evalml.pipelines.BaselineRegressionPipeline` method), 172
- `fit()` (`evalml.pipelines.BinaryClassificationPipeline` method), 121
- `fit()` (`evalml.pipelines.ClassificationPipeline` method), 117
- `fit()` (`evalml.pipelines.components.BaselineClassifier` method), 266
- `fit()` (`evalml.pipelines.components.BaselineRegressor` method), 294
- `fit()` (`evalml.pipelines.components.CatBoostClassifier` method), 245
- `fit()` (`evalml.pipelines.components.CatBoostRegressor` method), 276
- `fit()` (`evalml.pipelines.components.ComponentBase` method), 186
- `fit()` (`evalml.pipelines.components.DateTimeFeaturizer` method), 232
- `fit()` (`evalml.pipelines.components.DecisionTreeClassifier` method), 272
- `fit()` (`evalml.pipelines.components.DecisionTreeRegressor` method), 303
- `fit()` (`evalml.pipelines.components.DelayedFeatureTransformer` method), 238
- `fit()` (`evalml.pipelines.components.DFSTransformer` method), 241
- `fit()` (`evalml.pipelines.components.DropColumns` method), 196
- `fit()` (`evalml.pipelines.components.DropNullColumns` method), 229
- `fit()` (`evalml.pipelines.components.ElasticNetClassifier` method), 248
- `fit()` (`evalml.pipelines.components.ElasticNetRegressor` method), 279
- `fit()` (`evalml.pipelines.components.Estimator` method), 191
- `fit()` (`evalml.pipelines.components.ExtraTreesClassifier` method), 251
- `fit()` (`evalml.pipelines.components.ExtraTreesRegressor` method), 285
- `fit()` (`evalml.pipelines.components.Imputer` method), 213
- `fit()` (`evalml.pipelines.components.LightGBMClassifier` method), 257
- `fit()` (`evalml.pipelines.components.LightGBMRegressor` method), 306
- `fit()` (`evalml.pipelines.components.LinearRegressor` method), 282
- `fit()` (`evalml.pipelines.components.LogisticRegressionClassifier` method), 260
- `fit()` (`evalml.pipelines.components.OneHotEncoder` method), 203
- `fit()` (`evalml.pipelines.components.PerColumnImputer` method), 210
- `fit()` (`evalml.pipelines.components.RandomForestClassifier` method), 254
- `fit()` (`evalml.pipelines.components.RandomForestRegressor` method), 288
- `fit()` (`evalml.pipelines.components.RFClassifierSelectFromModel` method), 226
- `fit()` (`evalml.pipelines.components.RFRegressorSelectFromModel` method), 222
- `fit()` (`evalml.pipelines.components.SelectColumns` method), 199
- `fit()` (`evalml.pipelines.components.SimpleImputer` method), 216
- `fit()` (`evalml.pipelines.components.StackedEnsembleClassifier` method), 269
- `fit()` (`evalml.pipelines.components.StackedEnsembleRegressor` method), 300
- `fit()` (`evalml.pipelines.components.StandardScaler` method), 219

`fit()` (`evalml.pipelines.components.TargetEncoder` method), 199
`fit()` (`evalml.pipelines.components.TextFeaturizer` method), 235
`fit()` (`evalml.pipelines.components.TimeSeriesBaselineRegressor` method), 219
`fit()` (`evalml.pipelines.components.TimeSeriesBaselineRegressor` method), 297
`fit()` (`evalml.pipelines.components.Transformer` method), 189
`fit()` (`evalml.pipelines.components.XGBoostClassifier` method), 263
`fit()` (`evalml.pipelines.components.XGBoostRegressor` method), 291
`fit()` (`evalml.pipelines.MeanBaselineRegressionPipeline` method), 177
`fit()` (`evalml.pipelines.ModeBaselineBinaryPipeline` method), 162
`fit()` (`evalml.pipelines.ModeBaselineMulticlassPipeline` method), 167
`fit()` (`evalml.pipelines.MulticlassClassificationPipeline` method), 126
`fit()` (`evalml.pipelines.PipelineBase` method), 113
`fit()` (`evalml.pipelines.RegressionPipeline` method), 130
`fit()` (`evalml.pipelines.TimeSeriesBaselineRegressionPipeline` method), 181
`fit()` (`evalml.pipelines.TimeSeriesBinaryClassificationPipeline` method), 139
`fit()` (`evalml.pipelines.TimeSeriesClassificationPipeline` method), 134
`fit()` (`evalml.pipelines.TimeSeriesMulticlassClassificationPipeline` method), 144
`fit()` (`evalml.pipelines.TimeSeriesRegressionPipeline` method), 148
`fit_transform()` (`evalml.pipelines.components.DateFeatureTransformer` method), 232
`fit_transform()` (`evalml.pipelines.components.DelayedFeatureTransformer` method), 238
`fit_transform()` (`evalml.pipelines.components.DFSTransformer` method), 242
`fit_transform()` (`evalml.pipelines.components.DropColumns` method), 196
`fit_transform()` (`evalml.pipelines.components.DropNullColumns` method), 229
`fit_transform()` (`evalml.pipelines.components.Imputer` method), 213
`fit_transform()` (`evalml.pipelines.components.OneHotEncoder` method), 203
`fit_transform()` (`evalml.pipelines.components.PerColumnImputer` method), 210
`fit_transform()` (`evalml.pipelines.components.RFClassifierSelectFromModel` method), 226
`fit_transform()` (`evalml.pipelines.components.RFRegressorSelectFromModel` method), 223
`fit_transform()` (`evalml.pipelines.components.SelectColumns` method), 199
`fit_transform()` (`evalml.pipelines.components.SimpleImputer` method), 216
`fit_transform()` (`evalml.pipelines.components.StandardScaler` method), 219
`fit_transform()` (`evalml.pipelines.components.TargetEncoder` method), 206
`fit_transform()` (`evalml.pipelines.components.TextFeaturizer` method), 235
`fit_transform()` (`evalml.pipelines.components.Transformer` method), 189
`FraudCost` (class in `evalml.objectives`), 329

G

`generate_component_code()` (in module `evalml.pipelines.components.utils`), 194
`generate_pipeline_code()` (in module `evalml.pipelines.utils`), 185
`get_all_objective_names()` (in module `evalml.objectives`), 418
`get_component()` (`evalml.pipelines.BaselineBinaryPipeline` method), 153
`get_component()` (`evalml.pipelines.BaselineMulticlassPipeline` method), 158
`get_component()` (`evalml.pipelines.BaselineRegressionPipeline` method), 173
`get_component()` (`evalml.pipelines.BinaryClassificationPipeline` method), 121
`get_component()` (`evalml.pipelines.ClassificationPipeline` method), 117
`get_component()` (`evalml.pipelines.MeanBaselineRegressionPipeline` method), 177
`get_component()` (`evalml.pipelines.ModeBaselineBinaryPipeline` method), 163
`get_component()` (`evalml.pipelines.ModeBaselineMulticlassPipeline` method), 167
`get_component()` (`evalml.pipelines.MulticlassClassificationPipeline` method), 126
`get_component()` (`evalml.pipelines.PipelineBase` method), 113
`get_component()` (`evalml.pipelines.RegressionPipeline` method), 131
`get_component()` (`evalml.pipelines.TimeSeriesBaselineRegressionPipeline` method), 181
`get_component()` (`evalml.pipelines.TimeSeriesBinaryClassificationPipeline` method), 139
`get_component()` (`evalml.pipelines.TimeSeriesClassificationPipeline` method), 134
`get_component()` (`evalml.pipelines.TimeSeriesMulticlassClassificationPipeline` method), 144
`get_component()` (`evalml.pipelines.TimeSeriesRegressionPipeline` method), 148
`get_core_objective_names()` (in module `evalml.objectives`), 418

`get_core_objectives()` (in module `evalml.objectives`), 418
`get_default_primary_search_objective()` (in module `evalml.automl`), 105
`get_estimators()` (in module `evalml.pipelines.components.utils`), 193
`get_feature_names()` (`evalml.pipelines.components.DateTimeFeaturizer` method), 232
`get_feature_names()` (`evalml.pipelines.components.OneHotEncoder` method), 203
`get_feature_names()` (`evalml.pipelines.components.TargetEncoder` method), 207
`get_names()` (`evalml.pipelines.components.RFClassifier` method), 226
`get_names()` (`evalml.pipelines.components.RFRegressor` method), 223
`get_non_core_objectives()` (in module `evalml.objectives`), 418
`get_objective()` (in module `evalml.objectives`), 419
`get_pipeline()` (`evalml.automl.AutoMLSearch` method), 103
`get_prediction_vs_actual_data()` (in module `evalml.model_understanding`), 312
`get_prediction_vs_actual_over_time_data()` (in module `evalml.model_understanding`), 311
`get_random_seed()` (in module `evalml.utils`), 450
`get_random_state()` (in module `evalml.utils`), 449
`graph()` (`evalml.pipelines.BaselineBinaryPipeline` method), 153
`graph()` (`evalml.pipelines.BaselineMulticlassPipeline` method), 158
`graph()` (`evalml.pipelines.BaselineRegressionPipeline` method), 173
`graph()` (`evalml.pipelines.BinaryClassificationPipeline` method), 122
`graph()` (`evalml.pipelines.ClassificationPipeline` method), 117
`graph()` (`evalml.pipelines.MeanBaselineRegressionPipeline` method), 177
`graph()` (`evalml.pipelines.ModeBaselineBinaryPipeline` method), 163
`graph()` (`evalml.pipelines.ModeBaselineMulticlassPipeline` method), 168
`graph()` (`evalml.pipelines.MulticlassClassificationPipeline` method), 126
`graph()` (`evalml.pipelines.PipelineBase` method), 113
`graph()` (`evalml.pipelines.RegressionPipeline` method), 131
`graph()` (`evalml.pipelines.TimeSeriesBaselineRegressionPipeline` method), 182
`graph()` (`evalml.pipelines.TimeSeriesBinaryClassificationPipeline` method), 139
`graph()` (`evalml.pipelines.TimeSeriesClassificationPipeline` method), 135
`graph()` (`evalml.pipelines.TimeSeriesMulticlassClassificationPipeline` method), 144
`graph()` (`evalml.pipelines.TimeSeriesRegressionPipeline` method), 149
`graph_binary_objective_vs_threshold()` (in module `evalml.model_understanding`), 314
`graph_confusion_matrix()` (in module `evalml.model_understanding`), 314
`graph_feature_importance()` (`evalml.pipelines.BaselineBinaryPipeline` method), 154
`graph_feature_importance()` (`evalml.pipelines.BaselineMulticlassPipeline` method), 158
`graph_feature_importance()` (`evalml.pipelines.BaselineRegressionPipeline` method), 173
`graph_feature_importance()` (`evalml.pipelines.BinaryClassificationPipeline` method), 122
`graph_feature_importance()` (`evalml.pipelines.ClassificationPipeline` method), 117
`graph_feature_importance()` (`evalml.pipelines.MeanBaselineRegressionPipeline` method), 177
`graph_feature_importance()` (`evalml.pipelines.ModeBaselineBinaryPipeline` method), 163
`graph_feature_importance()` (`evalml.pipelines.ModeBaselineMulticlassPipeline` method), 168
`graph_feature_importance()` (`evalml.pipelines.MulticlassClassificationPipeline` method), 126
`graph_feature_importance()` (`evalml.pipelines.PipelineBase` method), 113
`graph_feature_importance()` (`evalml.pipelines.RegressionPipeline` method), 131
`graph_feature_importance()` (`evalml.pipelines.TimeSeriesBaselineRegressionPipeline` method), 182
`graph_feature_importance()` (`evalml.pipelines.TimeSeriesBinaryClassificationPipeline` method), 140
`graph_feature_importance()` (`evalml.pipelines.TimeSeriesClassificationPipeline` method), 135
`graph_feature_importance()` (`evalml.pipelines.TimeSeriesMulticlassClassificationPipeline` method), 144
`graph_feature_importance()` (`evalml.pipelines.TimeSeriesRegressionPipeline` method), 149

- (*evalml.pipelines.TimeSeriesMulticlassClassificationPipeline* method), 144
- graph_feature_importance()* (*evalml.pipelines.TimeSeriesRegressionPipeline* method), 149
- graph_partial_dependence()* (in module *evalml.model_understanding*), 315
- graph_permutation_importance()* (in module *evalml.model_understanding*), 314
- graph_precision_recall_curve()* (in module *evalml.model_understanding*), 313
- graph_prediction_vs_actual()* (in module *evalml.model_understanding*), 315
- graph_prediction_vs_actual_over_time()* (in module *evalml.model_understanding*), 315
- graph_roc_curve()* (in module *evalml.model_understanding*), 313
- GridSearchTuner* (class in *evalml.tuners*), 424
- ## H
- handle_model_family()* (in module *evalml.model_family*), 420
- handle_problem_types()* (in module *evalml.problem_types*), 419
- HighlyNullDataCheck* (class in *evalml.data_checks*), 431
- hyperparameter_ranges* (*evalml.pipelines.components.BaselineClassifier* attribute), 264
- hyperparameter_ranges* (*evalml.pipelines.components.BaselineRegressor* attribute), 292
- hyperparameter_ranges* (*evalml.pipelines.components.CatBoostClassifier* attribute), 243
- hyperparameter_ranges* (*evalml.pipelines.components.CatBoostRegressor* attribute), 274
- hyperparameter_ranges* (*evalml.pipelines.components.DateTimeFeaturizer* attribute), 230
- hyperparameter_ranges* (*evalml.pipelines.components.DecisionTreeClassifier* attribute), 271
- hyperparameter_ranges* (*evalml.pipelines.components.DecisionTreeRegressor* attribute), 302
- hyperparameter_ranges* (*evalml.pipelines.components.DelayedFeatureTransformer* attribute), 237
- hyperparameter_ranges* (*evalml.pipelines.components.DFSTransformer* attribute), 240
- hyperparameter_ranges* (*evalml.pipelines.components.DropColumns* attribute), 195
- hyperparameter_ranges* (*evalml.pipelines.components.DropNullColumns* attribute), 227
- hyperparameter_ranges* (*evalml.pipelines.components.ElasticNetClassifier* attribute), 246
- hyperparameter_ranges* (*evalml.pipelines.components.ElasticNetRegressor* attribute), 277
- hyperparameter_ranges* (*evalml.pipelines.components.ExtraTreesClassifier* attribute), 249
- hyperparameter_ranges* (*evalml.pipelines.components.ExtraTreesRegressor* attribute), 283
- hyperparameter_ranges* (*evalml.pipelines.components.Imputer* attribute), 211
- hyperparameter_ranges* (*evalml.pipelines.components.LightGBMClassifier* attribute), 255
- hyperparameter_ranges* (*evalml.pipelines.components.LightGBMRegressor* attribute), 305
- hyperparameter_ranges* (*evalml.pipelines.components.LinearRegressor* attribute), 280
- hyperparameter_ranges* (*evalml.pipelines.components.LogisticRegressionClassifier* attribute), 258
- hyperparameter_ranges* (*evalml.pipelines.components.OneHotEncoder* attribute), 201
- hyperparameter_ranges* (*evalml.pipelines.components.PerColumnImputer* attribute), 208
- hyperparameter_ranges* (*evalml.pipelines.components.RandomForestClassifier* attribute), 252
- hyperparameter_ranges* (*evalml.pipelines.components.RandomForestRegressor* attribute), 286
- hyperparameter_ranges* (*evalml.pipelines.components.RFClassifierSelectFromModel* attribute), 224
- hyperparameter_ranges* (*evalml.pipelines.components.RFRegressorSelectFromModel* attribute), 221
- hyperparameter_ranges* (*evalml.pipelines.components.SelectColumns* attribute), 198
- hyperparameter_ranges*

<code>(evalml.pipelines.components.SimpleImputer attribute), 215</code>	<code>(evalml.objectives.AccuracyMulticlass class method), 343</code>
<code>hyperparameter_ranges</code> <code>(evalml.pipelines.components.StackedEnsembleClassifier attribute), 267</code>	<code>is_defined_for_problem_type()</code> <code>(evalml.objectives.AUC class method), 345</code>
<code>hyperparameter_ranges</code> <code>(evalml.pipelines.components.StackedEnsembleRegressor attribute), 298</code>	<code>is_defined_for_problem_type()</code> <code>(evalml.objectives.AUCMacro class method), 348</code>
<code>hyperparameter_ranges</code> <code>(evalml.pipelines.components.StandardScaler attribute), 218</code>	<code>is_defined_for_problem_type()</code> <code>(evalml.objectives.AUCMicro class method), 350</code>
<code>hyperparameter_ranges</code> <code>(evalml.pipelines.components.TargetEncoder attribute), 204</code>	<code>is_defined_for_problem_type()</code> <code>(evalml.objectives.AUCWeighted class method), 352</code>
<code>hyperparameter_ranges</code> <code>(evalml.pipelines.components.TextFeaturizer attribute), 234</code>	<code>is_defined_for_problem_type()</code> <code>(evalml.objectives.BalancedAccuracyBinary class method), 355</code>
<code>hyperparameter_ranges</code> <code>(evalml.pipelines.components.TimeSeriesBaselineRegressor class attribute), 295</code>	<code>is_defined_for_problem_type()</code> <code>(evalml.objectives.BalancedAccuracyMulticlass class method), 357</code>
<code>hyperparameter_ranges</code> <code>(evalml.pipelines.components.XGBoostClassifier attribute), 261</code>	<code>is_defined_for_problem_type()</code> <code>(evalml.objectives.BinaryClassificationObjective class method), 322</code>
<code>hyperparameter_ranges</code> <code>(evalml.pipelines.components.XGBoostRegressor attribute), 289</code>	<code>is_defined_for_problem_type()</code> <code>(evalml.objectives.CostBenefitMatrix class method), 336</code>
<code>hyperparameters</code> <code>(evalml.pipelines.BaselineBinaryPipeline attribute), 151</code>	<code>is_defined_for_problem_type()</code> <code>(evalml.objectives.ExpVariance class method), 412</code>
<code>hyperparameters</code> <code>(evalml.pipelines.BaselineMulticlassPipeline attribute), 156</code>	<code>is_defined_for_problem_type()</code> <code>(evalml.objectives.F1 class method), 360</code>
<code>hyperparameters</code> <code>(evalml.pipelines.BaselineRegressionPipeline attribute), 170</code>	<code>is_defined_for_problem_type()</code> <code>(evalml.objectives.F1Macro class method), 364</code>
<code>hyperparameters</code> <code>(evalml.pipelines.MeanBaselineRegressionPipeline attribute), 175</code>	<code>is_defined_for_problem_type()</code> <code>(evalml.objectives.F1Micro class method), 362</code>
<code>hyperparameters</code> <code>(evalml.pipelines.ModeBaselineBinaryPipeline attribute), 160</code>	<code>is_defined_for_problem_type()</code> <code>(evalml.objectives.F1Weighted class method), 366</code>
<code>hyperparameters</code> <code>(evalml.pipelines.ModeBaselineMulticlassPipeline attribute), 165</code>	<code>is_defined_for_problem_type()</code> <code>(evalml.objectives.FraudCost class method), 330</code>
<code>hyperparameters</code> <code>(evalml.pipelines.TimeSeriesBaselineRegressionPipeline attribute), 179</code>	<code>is_defined_for_problem_type()</code> <code>(evalml.objectives.FraudCost class method), 330</code>
<code> </code>	
<code>IDColumnsDataCheck (class in evalml.data_checks), 433</code>	<code>is_defined_for_problem_type()</code> <code>(evalml.objectives.LeadScoring class method), 333</code>
<code>import_or_raise() (in module evalml.utils), 449</code>	
<code>Imputer (class in evalml.pipelines.components), 211</code>	<code>is_defined_for_problem_type()</code> <code>(evalml.objectives.LogLossBinary class method), 369</code>
<code>infer_feature_types() (in module evalml.utils), 450</code>	
<code>InvalidTargetDataCheck (class in evalml.data_checks), 430</code>	<code>is_defined_for_problem_type()</code> <code>(evalml.objectives.LogLossMulticlass class method), 371</code>
<code>is_defined_for_problem_type()</code> <code>(evalml.objectives.AccuracyBinary class method), 340</code>	<code>is_defined_for_problem_type()</code> <code>(evalml.objectives.MAE class method), 399</code>
<code>is_defined_for_problem_type()</code>	<code>is_defined_for_problem_type()</code>

(*evalml.objectives.MAPE class method*), 401
is_defined_for_problem_type() (*evalml.objectives.MaxError class method*), 410
is_defined_for_problem_type() (*evalml.objectives.MCCBinary class method*), 374
is_defined_for_problem_type() (*evalml.objectives.MCCMulticlass class method*), 376
is_defined_for_problem_type() (*evalml.objectives.MeanSquaredLogError class method*), 406
is_defined_for_problem_type() (*evalml.objectives.MedianAE class method*), 408
is_defined_for_problem_type() (*evalml.objectives.MSE class method*), 403
is_defined_for_problem_type() (*evalml.objectives.MulticlassClassificationObjective class method*), 325
is_defined_for_problem_type() (*evalml.objectives.ObjectiveBase class method*), 320
is_defined_for_problem_type() (*evalml.objectives.Precision class method*), 379
is_defined_for_problem_type() (*evalml.objectives.PrecisionMacro class method*), 383
is_defined_for_problem_type() (*evalml.objectives.PrecisionMicro class method*), 381
is_defined_for_problem_type() (*evalml.objectives.PrecisionWeighted class method*), 385
is_defined_for_problem_type() (*evalml.objectives.R2 class method*), 396
is_defined_for_problem_type() (*evalml.objectives.Recall class method*), 388
is_defined_for_problem_type() (*evalml.objectives.RecallMacro class method*), 392
is_defined_for_problem_type() (*evalml.objectives.RecallMicro class method*), 390
is_defined_for_problem_type() (*evalml.objectives.RecallWeighted class method*), 394
is_defined_for_problem_type() (*evalml.objectives.RegressionObjective class method*), 327
is_defined_for_problem_type() (*evalml.objectives.RootMeanSquaredError class method*), 414
is_defined_for_problem_type() (*evalml.objectives.RootMeanSquaredLogError class method*), 416
is_search_space_exhausted() (*evalml.tuners.GridSearchTuner method*), 425
is_search_space_exhausted() (*evalml.tuners.RandomSearchTuner method*), 427
is_search_space_exhausted() (*evalml.tuners.SKOptTuner method*), 424
is_search_space_exhausted() (*evalml.tuners.Tuner method*), 422
 IterativeAlgorithm (*class in evalml.automl.automl_algorithm*), 108

L

LeadScoring (*class in evalml.objectives*), 332
 LightGBMClassifier (*class in evalml.pipelines.components*), 255
 LightGBMRegressor (*class in evalml.pipelines.components*), 305
 LinearRegressor (*class in evalml.pipelines.components*), 280
 load() (*evalml.automl.AutoMLSearch static method*), 103
 load() (*evalml.pipelines.BaselineBinaryPipeline static method*), 154
 load() (*evalml.pipelines.BaselineMulticlassPipeline static method*), 159
 load() (*evalml.pipelines.BaselineRegressionPipeline static method*), 173
 load() (*evalml.pipelines.BinaryClassificationPipeline static method*), 122
 load() (*evalml.pipelines.ClassificationPipeline static method*), 118
 load() (*evalml.pipelines.components.BaselineClassifier static method*), 266
 load() (*evalml.pipelines.components.BaselineRegressor static method*), 294
 load() (*evalml.pipelines.components.CatBoostClassifier static method*), 245
 load() (*evalml.pipelines.components.CatBoostRegressor static method*), 276
 load() (*evalml.pipelines.components.ComponentBase static method*), 187
 load() (*evalml.pipelines.components.DateFeaturizer static method*), 233
 load() (*evalml.pipelines.components.DecisionTreeClassifier static method*), 272

<code>load()</code> (<code>evalml.pipelines.components.DecisionTreeRegressor</code> static method), 303	<code>load()</code> (<code>evalml.pipelines.components.TextFeaturizer</code> static method), 236
<code>load()</code> (<code>evalml.pipelines.components.DelayedFeatureTransformer</code> static method), 239	<code>load()</code> (<code>evalml.pipelines.components.TimeSeriesBaselineRegressor</code> static method), 297
<code>load()</code> (<code>evalml.pipelines.components.DFSTransformer</code> static method), 242	<code>load()</code> (<code>evalml.pipelines.components.Transformer</code> static method), 189
<code>load()</code> (<code>evalml.pipelines.components.DropColumns</code> static method), 197	<code>load()</code> (<code>evalml.pipelines.components.XGBoostClassifier</code> static method), 263
<code>load()</code> (<code>evalml.pipelines.components.DropNullColumns</code> static method), 229	<code>load()</code> (<code>evalml.pipelines.components.XGBoostRegressor</code> static method), 291
<code>load()</code> (<code>evalml.pipelines.components.ElasticNetClassifier</code> static method), 248	<code>load()</code> (<code>evalml.pipelines.MeanBaselineRegressionPipeline</code> static method), 178
<code>load()</code> (<code>evalml.pipelines.components.ElasticNetRegressor</code> static method), 279	<code>load()</code> (<code>evalml.pipelines.ModeBaselineBinaryPipeline</code> static method), 163
<code>load()</code> (<code>evalml.pipelines.components.Estimator</code> static method), 192	<code>load()</code> (<code>evalml.pipelines.ModeBaselineMulticlassPipeline</code> static method), 168
<code>load()</code> (<code>evalml.pipelines.components.ExtraTreesClassifier</code> static method), 251	<code>load()</code> (<code>evalml.pipelines.MulticlassClassificationPipeline</code> static method), 127
<code>load()</code> (<code>evalml.pipelines.components.ExtraTreesRegressor</code> static method), 285	<code>load()</code> (<code>evalml.pipelines.PipelineBase</code> static method), 114
<code>load()</code> (<code>evalml.pipelines.components.Imputer</code> static method), 214	<code>load()</code> (<code>evalml.pipelines.RegressionPipeline</code> static method), 131
<code>load()</code> (<code>evalml.pipelines.components.LightGBMClassifier</code> static method), 257	<code>load()</code> (<code>evalml.pipelines.TimeSeriesBaselineRegressionPipeline</code> static method), 182
<code>load()</code> (<code>evalml.pipelines.components.LightGBMRegressor</code> static method), 307	<code>load()</code> (<code>evalml.pipelines.TimeSeriesBinaryClassificationPipeline</code> static method), 140
<code>load()</code> (<code>evalml.pipelines.components.LinearRegressor</code> static method), 282	<code>load()</code> (<code>evalml.pipelines.TimeSeriesClassificationPipeline</code> static method), 135
<code>load()</code> (<code>evalml.pipelines.components.LogisticRegressionClassifier</code> static method), 260	<code>load()</code> (<code>evalml.pipelines.TimeSeriesMulticlassClassificationPipeline</code> static method), 145
<code>load()</code> (<code>evalml.pipelines.components.OneHotEncoder</code> static method), 203	<code>load()</code> (<code>evalml.pipelines.TimeSeriesRegressionPipeline</code> static method), 149
<code>load()</code> (<code>evalml.pipelines.components.PerColumnImputer</code> static method), 210	<code>load_breast_cancer()</code> (in module <code>evalml.demos</code>), 98
<code>load()</code> (<code>evalml.pipelines.components.RandomForestClassifier</code> static method), 254	<code>load_churn()</code> (in module <code>evalml.demos</code>), 98
<code>load()</code> (<code>evalml.pipelines.components.RandomForestRegressor</code> static method), 288	<code>load_data()</code> (in module <code>evalml.preprocessing</code>), 99
<code>load()</code> (<code>evalml.pipelines.components.RFClassifierSelectFromModel</code> static method), 226	<code>load_diabetes()</code> (in module <code>evalml.demos</code>), 98
<code>load()</code> (<code>evalml.pipelines.components.RFRegressorSelectFromModel</code> static method), 223	<code>load_fraud()</code> (in module <code>evalml.demos</code>), 97
<code>load()</code> (<code>evalml.pipelines.components.SelectColumns</code> static method), 200	<code>load_model()</code> (in module <code>evalml.demos</code>), 97
<code>load()</code> (<code>evalml.pipelines.components.SimpleImputer</code> static method), 217	<code>log_and_save_error_callback()</code> (in module <code>evalml.automl.callbacks</code>), 110
<code>load()</code> (<code>evalml.pipelines.components.StackedEnsembleClassifier</code> static method), 269	<code>log_error_callback()</code> (in module <code>evalml.automl.callbacks</code>), 110
<code>load()</code> (<code>evalml.pipelines.components.StackedEnsembleRegressor</code> static method), 300	<code>LogisticRegressionClassifier</code> (class in <code>evalml.pipelines.components</code>), 258
<code>load()</code> (<code>evalml.pipelines.components.StandardScaler</code> static method), 219	<code>LogLossBinary</code> (class in <code>evalml.objectives</code>), 367
<code>load()</code> (<code>evalml.pipelines.components.TargetEncoder</code> static method), 207	<code>LogLossMulticlass</code> (class in <code>evalml.objectives</code>), 370

M

`make_pipeline()` (in module `evalml.pipelines.utils`), 184
`make_pipeline_from_components()` (in module `evalml.pipelines.utils`), 184
MAPE (class in `evalml.objectives`), 400
MaxError (class in `evalml.objectives`), 409
MCCBinary (class in `evalml.objectives`), 372
MCCMulticlass (class in `evalml.objectives`), 375
MeanBaselineRegressionPipeline (class in `evalml.pipelines`), 175
MeanSquaredLogError (class in `evalml.objectives`), 405
MedianAE (class in `evalml.objectives`), 407
`message_type` (`evalml.data_checks.DataCheckError` attribute), 445
`message_type` (`evalml.data_checks.DataCheckMessage` attribute), 443
`message_type` (`evalml.data_checks.DataCheckWarning` attribute), 446
ModeBaselineBinaryPipeline (class in `evalml.pipelines`), 160
ModeBaselineMulticlassPipeline (class in `evalml.pipelines`), 165
`model_family` (`evalml.pipelines.BaselineBinaryPipeline` attribute), 151
`model_family` (`evalml.pipelines.BaselineMulticlassPipeline` attribute), 156
`model_family` (`evalml.pipelines.BaselineRegressionPipeline` attribute), 170
`model_family` (`evalml.pipelines.components.BaselineClassifier` attribute), 264
`model_family` (`evalml.pipelines.components.BaselineRegressor` attribute), 292
`model_family` (`evalml.pipelines.components.CatBoostClassifier` attribute), 243
`model_family` (`evalml.pipelines.components.CatBoostRegressor` attribute), 274
`model_family` (`evalml.pipelines.components.DateTimeFeaturizer` attribute), 230
`model_family` (`evalml.pipelines.components.DecisionTreeClassifier` attribute), 271
`model_family` (`evalml.pipelines.components.DecisionTreeRegressor` attribute), 302
`model_family` (`evalml.pipelines.components.DelayedFeatureTransformer` attribute), 237
`model_family` (`evalml.pipelines.components.DFSTransformer` attribute), 240
`model_family` (`evalml.pipelines.components.DropColumns` attribute), 195
`model_family` (`evalml.pipelines.components.DropNullColumns` attribute), 227
`model_family` (`evalml.pipelines.components.ElasticNetClassifier` attribute), 246
`model_family` (`evalml.pipelines.components.ElasticNetRegressor` attribute), 277
`model_family` (`evalml.pipelines.components.ExtraTreesClassifier` attribute), 249
`model_family` (`evalml.pipelines.components.ExtraTreesRegressor` attribute), 283
`model_family` (`evalml.pipelines.components.Imputer` attribute), 211
`model_family` (`evalml.pipelines.components.LightGBMClassifier` attribute), 255
`model_family` (`evalml.pipelines.components.LightGBMRegressor` attribute), 305
`model_family` (`evalml.pipelines.components.LinearRegressor` attribute), 280
`model_family` (`evalml.pipelines.components.LogisticRegressionClassifier` attribute), 258
`model_family` (`evalml.pipelines.components.OneHotEncoder` attribute), 201
`model_family` (`evalml.pipelines.components.PerColumnImputer` attribute), 208
`model_family` (`evalml.pipelines.components.RandomForestClassifier` attribute), 252
`model_family` (`evalml.pipelines.components.RandomForestRegressor` attribute), 286
`model_family` (`evalml.pipelines.components.RFClassifierSelectFromModel` attribute), 224
`model_family` (`evalml.pipelines.components.RFRegressorSelectFromModel` attribute), 221
`model_family` (`evalml.pipelines.components.SelectColumns` attribute), 198
`model_family` (`evalml.pipelines.components.SimpleImputer` attribute), 215
`model_family` (`evalml.pipelines.components.StackedEnsembleClassifier` attribute), 267
`model_family` (`evalml.pipelines.components.StackedEnsembleRegressor` attribute), 298
`model_family` (`evalml.pipelines.components.StandardScaler` attribute), 218
`model_family` (`evalml.pipelines.components.TargetEncoder` attribute), 204
`model_family` (`evalml.pipelines.components.TextFeaturizer` attribute), 234
`model_family` (`evalml.pipelines.components.TimeSeriesBaselineRegressor` attribute), 295
`model_family` (`evalml.pipelines.components.XGBoostClassifier` attribute), 261
`model_family` (`evalml.pipelines.components.XGBoostRegressor` attribute), 289
`model_family` (`evalml.pipelines.MeanBaselineRegressionPipeline` attribute), 175
`model_family` (`evalml.pipelines.ModeBaselineBinaryPipeline` attribute), 160
`model_family` (`evalml.pipelines.ModeBaselineMulticlassPipeline` attribute), 165
`model_family` (`evalml.pipelines.TimeSeriesBaselineRegressionPipeline` attribute), 295

attribute), 179
 ModelFamily (class in *evalml.model_family*), 421
 MSE (class in *evalml.objectives*), 402
 MulticlassClassificationObjective (class in *evalml.objectives*), 324
 MulticlassClassificationPipeline (class in *evalml.pipelines*), 124

N

name (*evalml.data_checks.ClassImbalanceDataCheck* attribute), 439
 name (*evalml.data_checks.DataCheck* attribute), 428
 name (*evalml.data_checks.HighlyNullDataCheck* attribute), 431
 name (*evalml.data_checks.IDColumnsDataCheck* attribute), 433
 name (*evalml.data_checks.InvalidTargetDataCheck* attribute), 430
 name (*evalml.data_checks.NoVarianceDataCheck* attribute), 438
 name (*evalml.data_checks.OutliersDataCheck* attribute), 436
 name (*evalml.data_checks.TargetLeakageDataCheck* attribute), 434
 name (*evalml.pipelines.BaselineBinaryPipeline* attribute), 151
 name (*evalml.pipelines.BaselineMulticlassPipeline* attribute), 156
 name (*evalml.pipelines.BaselineRegressionPipeline* attribute), 170
 name (*evalml.pipelines.BinaryClassificationPipeline* attribute), 119
 name (*evalml.pipelines.ClassificationPipeline* attribute), 115
 name (*evalml.pipelines.components.BaselineClassifier* attribute), 264
 name (*evalml.pipelines.components.BaselineRegressor* attribute), 292
 name (*evalml.pipelines.components.CatBoostClassifier* attribute), 243
 name (*evalml.pipelines.components.CatBoostRegressor* attribute), 274
 name (*evalml.pipelines.components.DateTimeFeaturizer* attribute), 230
 name (*evalml.pipelines.components.DecisionTreeClassifier* attribute), 271
 name (*evalml.pipelines.components.DecisionTreeRegressor* attribute), 302
 name (*evalml.pipelines.components.DelayedFeatureTransformer* attribute), 237
 name (*evalml.pipelines.components.DFSTransformer* attribute), 240
 name (*evalml.pipelines.components.DropColumns* attribute), 195
 name (*evalml.pipelines.components.DropNullColumns* attribute), 227
 name (*evalml.pipelines.components.ElasticNetClassifier* attribute), 246
 name (*evalml.pipelines.components.ElasticNetRegressor* attribute), 277
 name (*evalml.pipelines.components.ExtraTreesClassifier* attribute), 249
 name (*evalml.pipelines.components.ExtraTreesRegressor* attribute), 283
 name (*evalml.pipelines.components.Imputer* attribute), 211
 name (*evalml.pipelines.components.LightGBMClassifier* attribute), 255
 name (*evalml.pipelines.components.LightGBMRegressor* attribute), 305
 name (*evalml.pipelines.components.LinearRegressor* attribute), 280
 name (*evalml.pipelines.components.LogisticRegressionClassifier* attribute), 258
 name (*evalml.pipelines.components.OneHotEncoder* attribute), 201
 name (*evalml.pipelines.components.PerColumnImputer* attribute), 208
 name (*evalml.pipelines.components.RandomForestClassifier* attribute), 252
 name (*evalml.pipelines.components.RandomForestRegressor* attribute), 286
 name (*evalml.pipelines.components.RFClassifierSelectFromModel* attribute), 224
 name (*evalml.pipelines.components.RFRegressorSelectFromModel* attribute), 221
 name (*evalml.pipelines.components.SelectColumns* attribute), 198
 name (*evalml.pipelines.components.SimpleImputer* attribute), 215
 name (*evalml.pipelines.components.StackedEnsembleClassifier* attribute), 267
 name (*evalml.pipelines.components.StackedEnsembleRegressor* attribute), 298
 name (*evalml.pipelines.components.StandardScaler* attribute), 218
 name (*evalml.pipelines.components.TargetEncoder* attribute), 204
 name (*evalml.pipelines.components.TextFeaturizer* attribute), 234
 name (*evalml.pipelines.components.TimeSeriesBaselineRegressor* attribute), 295
 name (*evalml.pipelines.components.XGBoostClassifier* attribute), 261
 name (*evalml.pipelines.components.XGBoostRegressor* attribute), 289
 name (*evalml.pipelines.MeanBaselineRegressionPipeline* attribute), 175

name (*evalml.pipelines.ModeBaselineBinaryPipeline* attribute), 160
 name (*evalml.pipelines.ModeBaselineMulticlassPipeline* attribute), 165
 name (*evalml.pipelines.MulticlassClassificationPipeline* attribute), 124
 name (*evalml.pipelines.PipelineBase* attribute), 111
 name (*evalml.pipelines.RegressionPipeline* attribute), 128
 name (*evalml.pipelines.TimeSeriesBaselineRegressionPipeline* attribute), 179
 name (*evalml.pipelines.TimeSeriesBinaryClassificationPipeline* attribute), 137
 name (*evalml.pipelines.TimeSeriesClassificationPipeline* attribute), 133
 name (*evalml.pipelines.TimeSeriesMulticlassClassificationPipeline* attribute), 142
 name (*evalml.pipelines.TimeSeriesRegressionPipeline* attribute), 146
 next_batch() (*evalml.automl.automl_algorithm.AutoMLAlgorithm* method), 107
 next_batch() (*evalml.automl.automl_algorithm.IterativeAlgorithm* method), 109
 normalize_confusion_matrix() (in module *evalml.model_understanding*), 309
 NoVarianceDataCheck (class in *evalml.data_checks*), 438
 number_of_features() (in module *evalml.preprocessing*), 99

O

objective_function() (*evalml.objectives.AccuracyBinary* method), 340
 objective_function() (*evalml.objectives.AccuracyMulticlass* method), 343
 objective_function() (*evalml.objectives.AUC* method), 345
 objective_function() (*evalml.objectives.AUCMacro* method), 348
 objective_function() (*evalml.objectives.AUCMicro* method), 350
 objective_function() (*evalml.objectives.AUCWeighted* method), 352
 objective_function() (*evalml.objectives.BalancedAccuracyBinary* method), 355
 objective_function() (*evalml.objectives.BalancedAccuracyMulticlass* method), 357
 objective_function() (*evalml.objectives.BinaryClassificationObjective* class method), 322
 objective_function() (*evalml.objectives.CostBenefitMatrix* method), 336
 objective_function() (*evalml.objectives.ExpVariance* method), 412
 objective_function() (*evalml.objectives.F1* method), 360
 objective_function() (*evalml.objectives.F1Macro* method), 364
 objective_function() (*evalml.objectives.F1Micro* method), 362
 objective_function() (*evalml.objectives.F1Weighted* method), 366
 objective_function() (*evalml.objectives.FraudCost* method), 330
 objective_function() (*evalml.objectives.LeadScoring* method), 333
 objective_function() (*evalml.objectives.LogLossBinary* method), 369
 objective_function() (*evalml.objectives.LogLossMulticlass* method), 371
 objective_function() (*evalml.objectives.MAE* method), 399
 objective_function() (*evalml.objectives.MAPE* method), 401
 objective_function() (*evalml.objectives.MaxError* method), 410
 objective_function() (*evalml.objectives.MCCBinary* method), 374
 objective_function() (*evalml.objectives.MCCMulticlass* method), 376
 objective_function() (*evalml.objectives.MeanSquaredLogError* method), 406
 objective_function() (*evalml.objectives.MedianAE* method), 408
 objective_function() (*evalml.objectives.MSE* method), 403
 objective_function() (*evalml.objectives.MulticlassClassificationObjective* class method), 325
 objective_function() (*evalml.objectives.ObjectiveBase* class method), 320
 objective_function() (*evalml.objectives.Precision* method), 379

[objective_function\(\)](#)
 ([evalml.objectives.PrecisionMacro](#) *method*), [334](#)
[objective_function\(\)](#)
 ([evalml.objectives.PrecisionMicro](#) *method*), [381](#)
[objective_function\(\)](#)
 ([evalml.objectives.PrecisionWeighted](#) *method*), [385](#)
[objective_function\(\)](#) ([evalml.objectives.R2](#) *method*), [397](#)
[objective_function\(\)](#) ([evalml.objectives.Recall](#) *method*), [388](#)
[objective_function\(\)](#)
 ([evalml.objectives.RecallMacro](#) *method*), [392](#)
[objective_function\(\)](#)
 ([evalml.objectives.RecallMicro](#) *method*), [390](#)
[objective_function\(\)](#)
 ([evalml.objectives.RecallWeighted](#) *method*), [394](#)
[objective_function\(\)](#)
 ([evalml.objectives.RegressionObjective](#) *class method*), [327](#)
[objective_function\(\)](#)
 ([evalml.objectives.RootMeanSquaredError](#) *method*), [414](#)
[objective_function\(\)](#)
 ([evalml.objectives.RootMeanSquaredLogError](#) *method*), [416](#)
[ObjectiveBase](#) (*class in evalml.objectives*), [319](#)
[OneHotEncoder](#) (*class in evalml.pipelines.components*), [201](#)
[optimize_threshold\(\)](#)
 ([evalml.objectives.AccuracyBinary](#) *method*), [340](#)
[optimize_threshold\(\)](#) ([evalml.objectives.AUC](#) *method*), [346](#)
[optimize_threshold\(\)](#)
 ([evalml.objectives.BalancedAccuracyBinary](#) *method*), [355](#)
[optimize_threshold\(\)](#)
 ([evalml.objectives.BinaryClassificationObjective](#) *method*), [323](#)
[optimize_threshold\(\)](#)
 ([evalml.objectives.CostBenefitMatrix](#) *method*), [337](#)
[optimize_threshold\(\)](#) ([evalml.objectives.F1](#) *method*), [360](#)
[optimize_threshold\(\)](#)
 ([evalml.objectives.FraudCost](#) *method*), [331](#)
[optimize_threshold\(\)](#)
 ([evalml.objectives.LeadScoring](#) *method*), [334](#)
[optimize_threshold\(\)](#)
 ([evalml.objectives.LogLossBinary](#) *method*), [369](#)
[optimize_threshold\(\)](#)
 ([evalml.objectives.MCCBinary](#) *method*), [374](#)
[optimize_threshold\(\)](#)
 ([evalml.objectives.Precision](#) *method*), [379](#)
[optimize_threshold\(\)](#) ([evalml.objectives.Recall](#) *method*), [388](#)
[OutliersDataCheck](#) (*class in evalml.data_checks*), [436](#)

P

[pad_with_nans\(\)](#) (*in module evalml.utils*), [450](#)
[partial_dependence\(\)](#) (*in module evalml.model_understanding*), [311](#)
[PerColumnImputer](#) (*class in evalml.pipelines.components*), [208](#)
[PipelineBase](#) (*class in evalml.pipelines*), [111](#)
[Precision](#) (*class in evalml.objectives*), [377](#)
[precision_recall_curve\(\)](#) (*in module evalml.model_understanding*), [309](#)
[PrecisionMacro](#) (*class in evalml.objectives*), [382](#)
[PrecisionMicro](#) (*class in evalml.objectives*), [380](#)
[PrecisionWeighted](#) (*class in evalml.objectives*), [384](#)
[predict\(\)](#) ([evalml.pipelines.BaselineBinaryPipeline](#) *method*), [154](#)
[predict\(\)](#) ([evalml.pipelines.BaselineMulticlassPipeline](#) *method*), [159](#)
[predict\(\)](#) ([evalml.pipelines.BaselineRegressionPipeline](#) *method*), [173](#)
[predict\(\)](#) ([evalml.pipelines.BinaryClassificationPipeline](#) *method*), [122](#)
[predict\(\)](#) ([evalml.pipelines.ClassificationPipeline](#) *method*), [118](#)
[predict\(\)](#) ([evalml.pipelines.components.BaselineClassifier](#) *method*), [266](#)
[predict\(\)](#) ([evalml.pipelines.components.BaselineRegressor](#) *method*), [294](#)
[predict\(\)](#) ([evalml.pipelines.components.CatBoostClassifier](#) *method*), [245](#)
[predict\(\)](#) ([evalml.pipelines.components.CatBoostRegressor](#) *method*), [276](#)
[predict\(\)](#) ([evalml.pipelines.components.DecisionTreeClassifier](#) *method*), [273](#)
[predict\(\)](#) ([evalml.pipelines.components.DecisionTreeRegressor](#) *method*), [304](#)
[predict\(\)](#) ([evalml.pipelines.components.ElasticNetClassifier](#) *method*), [248](#)
[predict\(\)](#) ([evalml.pipelines.components.ElasticNetRegressor](#) *method*), [279](#)

`predict()` (`evalml.pipelines.components.Estimator` `method`), 192
`predict_proba()` (`evalml.pipelines.BinaryClassificationPipeline` `method`), 123
`predict()` (`evalml.pipelines.components.ExtraTreesClassifier` `method`), 251
`predict_proba()` (`evalml.pipelines.ClassificationPipeline` `method`), 118
`predict()` (`evalml.pipelines.components.ExtraTreesRegressor` `method`), 285
`predict_proba()` (`evalml.pipelines.components.BaselineClassifier` `method`), 267
`predict()` (`evalml.pipelines.components.LightGBMClassifier` `method`), 257
`predict_proba()` (`evalml.pipelines.components.BaselineRegressor` `method`), 295
`predict()` (`evalml.pipelines.components.LightGBMRegressor` `method`), 307
`predict_proba()` (`evalml.pipelines.components.CatBoostClassifier` `method`), 246
`predict()` (`evalml.pipelines.components.LinearRegressor` `method`), 282
`predict_proba()` (`evalml.pipelines.components.CatBoostRegressor` `method`), 276
`predict()` (`evalml.pipelines.components.LogisticRegressionClassifier` `method`), 260
`predict_proba()` (`evalml.pipelines.components.DecisionTreeClassifier` `method`), 273
`predict()` (`evalml.pipelines.components.RandomForestClassifier` `method`), 254
`predict_proba()` (`evalml.pipelines.components.DecisionTreeRegressor` `method`), 304
`predict()` (`evalml.pipelines.components.RandomForestRegressor` `method`), 288
`predict_proba()` (`evalml.pipelines.components.ElasticNetClassifier` `method`), 248
`predict()` (`evalml.pipelines.components.StackedEnsembleClassifier` `method`), 270
`predict_proba()` (`evalml.pipelines.components.ElasticNetRegressor` `method`), 279
`predict()` (`evalml.pipelines.components.StackedEnsembleRegressor` `method`), 301
`predict_proba()` (`evalml.pipelines.components.Estimator` `method`), 192
`predict()` (`evalml.pipelines.components.TimeSeriesBaselineRegressor` `method`), 297
`predict_proba()` (`evalml.pipelines.components.ExtraTreesClassifier` `method`), 251
`predict()` (`evalml.pipelines.components.XGBoostClassifier` `method`), 263
`predict_proba()` (`evalml.pipelines.components.ExtraTreesRegressor` `method`), 285
`predict()` (`evalml.pipelines.components.XGBoostRegressor` `method`), 291
`predict_proba()` (`evalml.pipelines.components.LightGBMClassifier` `method`), 257
`predict()` (`evalml.pipelines.MeanBaselineRegressionPipeline` `method`), 178
`predict_proba()` (`evalml.pipelines.components.LightGBMRegressor` `method`), 307
`predict()` (`evalml.pipelines.ModeBaselineBinaryPipeline` `method`), 163
`predict_proba()` (`evalml.pipelines.components.LinearRegressor` `method`), 282
`predict()` (`evalml.pipelines.ModeBaselineMulticlassPipeline` `method`), 168
`predict_proba()` (`evalml.pipelines.components.LogisticRegressionClassifier` `method`), 260
`predict()` (`evalml.pipelines.MulticlassClassificationPipeline` `method`), 127
`predict_proba()` (`evalml.pipelines.components.RandomForestClassifier` `method`), 254
`predict()` (`evalml.pipelines.PipelineBase` `method`), 114
`predict_proba()` (`evalml.pipelines.components.RandomForestRegressor` `method`), 288
`predict()` (`evalml.pipelines.RegressionPipeline` `method`), 131
`predict_proba()` (`evalml.pipelines.components.StackedEnsembleClassifier` `method`), 270
`predict()` (`evalml.pipelines.TimeSeriesBaselineRegressionPipeline` `method`), 182
`predict_proba()` (`evalml.pipelines.components.StackedEnsembleRegressor` `method`), 301
`predict()` (`evalml.pipelines.TimeSeriesBinaryClassificationPipeline` `method`), 140
`predict_proba()` (`evalml.pipelines.components.TimeSeriesBaselineRegressor` `method`), 298
`predict()` (`evalml.pipelines.TimeSeriesClassificationPipeline` `method`), 135
`predict_proba()` (`evalml.pipelines.components.XGBoostClassifier` `method`), 263
`predict()` (`evalml.pipelines.TimeSeriesMulticlassClassificationPipeline` `method`), 145
`predict_proba()` (`evalml.pipelines.components.XGBoostRegressor` `method`), 291
`predict()` (`evalml.pipelines.TimeSeriesRegressionPipeline` `method`), 149
`predict_proba()` (`evalml.pipelines.ModeBaselineBinaryPipeline` `method`), 164
`predict_proba()` (`evalml.pipelines.BaselineBinaryPipeline` `method`), 154
`predict_proba()` (`evalml.pipelines.ModeBaselineMulticlassPipeline` `method`), 169
`predict_proba()` (`evalml.pipelines.BaselineMulticlassPipeline` `method`), 159
`predict_proba()` (`evalml.pipelines.MulticlassClassificationPipeline` `method`), 127

[predict_proba\(\) \(evalml.pipelines.TimeSeriesBinaryClassificationPipelineRegressor \(class in evalml.pipelines.components\), 286](#)
[predict_proba\(\) \(evalml.pipelines.TimeSeriesClassificationPipelineSearchTuner \(class in evalml.tuners\), 426](#)
[predict_proba\(\) \(evalml.pipelines.TimeSeriesClassificationPipelineRecall \(class in evalml.objectives\), 386](#)
[predict_proba\(\) \(evalml.pipelines.TimeSeriesMulticlassClassificationPipelineRecall \(class in evalml.objectives\), 391](#)
[predict_proba\(\) \(evalml.pipelines.TimeSeriesMulticlassClassificationPipelineRecallMicro \(class in evalml.objectives\), 389](#)
[problem_type \(evalml.pipelines.BaselineBinaryPipelineRecallWeighted \(class in evalml.objectives\), 393](#)
[problem_type \(evalml.pipelines.BaselineMulticlassPipelineRegressionObjective \(class in evalml.objectives\), 326](#)
[problem_type \(evalml.pipelines.BaselineRegressionPipelineRegressionPipeline \(class in evalml.pipelines\), 128](#)
[problem_type \(evalml.pipelines.BinaryClassificationPipelineRFCClassifierSelectFromModel \(class in evalml.pipelines.components\), 224](#)
[problem_type \(evalml.pipelines.BinaryClassificationPipelineRFRegressorSelectFromModel \(class in evalml.pipelines.components\), 221](#)
[problem_type \(evalml.pipelines.ClassificationPipelineroc_curve\(\) \(in module evalml.model_understanding\), 310](#)
[problem_type \(evalml.pipelines.MeanBaselineRegressionPipelineRootMeanSquaredError \(class in evalml.objectives\), 413](#)
[problem_type \(evalml.pipelines.ModeBaselineBinaryPipelineRootMeanSquaredLogError \(class in evalml.objectives\), 415](#)
[problem_type \(evalml.pipelines.ModeBaselineMulticlassPipelinePipeline attribute\), 124](#)
[problem_type \(evalml.pipelines.MulticlassClassificationPipeline attribute\), 111](#)
[problem_type \(evalml.pipelines.PipelineBase attribute\), 128](#)
[problem_type \(evalml.pipelines.RegressionPipeline attribute\), 179](#)
[problem_type \(evalml.pipelines.TimeSeriesBaselineRegressionPipeline attribute\), 174](#)
[problem_type \(evalml.pipelines.TimeSeriesBinaryClassificationPipeline attribute\), 123](#)
[problem_type \(evalml.pipelines.TimeSeriesClassificationPipeline attribute\), 118](#)
[problem_type \(evalml.pipelines.TimeSeriesMulticlassClassificationPipeline attribute\), 267](#)
[problem_type \(evalml.pipelines.TimeSeriesRegressionPipeline attribute\), 295](#)
[ProblemTypes \(class in evalml.problem_types\), 420](#)
[propose\(\) \(evalml.tuners.GridSearchTuner method\), 426](#)
[propose\(\) \(evalml.tuners.RandomSearchTuner method\), 427](#)
[propose\(\) \(evalml.tuners.SKOptTuner method\), 424](#)
[propose\(\) \(evalml.tuners.Tuner method\), 422](#)
R
[R2 \(class in evalml.objectives\), 396](#)
[raise_and_save_error_callback\(\) \(in module evalml.automl.callbacks\), 110](#)
[raise_error_callback\(\) \(in module evalml.automl.callbacks\), 110](#)
[RandomForestClassifier \(class in evalml.pipelines.components\), 252](#)
[save\(\) \(evalml.automl.AutoMLSearch method\), 104](#)
[save\(\) \(evalml.pipelines.BaselineBinaryPipeline method\), 155](#)
[save\(\) \(evalml.pipelines.BaselineMulticlassPipeline method\), 159](#)
[save\(\) \(evalml.pipelines.BaselineRegressionPipeline method\), 174](#)
[save\(\) \(evalml.pipelines.BinaryClassificationPipeline method\), 123](#)
[save\(\) \(evalml.pipelines.ClassificationPipeline method\), 118](#)
[save\(\) \(evalml.pipelines.components.BaselineClassifier method\), 267](#)
[save\(\) \(evalml.pipelines.components.BaselineRegressor method\), 295](#)
[save\(\) \(evalml.pipelines.components.CatBoostClassifier method\), 246](#)
[save\(\) \(evalml.pipelines.components.CatBoostRegressor method\), 277](#)
[save\(\) \(evalml.pipelines.components.ComponentBase method\), 187](#)
[save\(\) \(evalml.pipelines.components.DateFeatureFeaturizer method\), 233](#)
[save\(\) \(evalml.pipelines.components.DecisionTreeClassifier method\), 273](#)
[save\(\) \(evalml.pipelines.components.DecisionTreeRegressor method\), 304](#)
[save\(\) \(evalml.pipelines.components.DelayedFeatureTransformer method\), 239](#)
[save\(\) \(evalml.pipelines.components.DFSSTransformer method\), 242](#)

<code>save()</code> (<code>evalml.pipelines.components.DropColumns</code> method), 197	<code>save()</code> (<code>evalml.pipelines.components.XGBoostClassifier</code> method), 264
<code>save()</code> (<code>evalml.pipelines.components.DropNullColumns</code> method), 229	<code>save()</code> (<code>evalml.pipelines.components.XGBoostRegressor</code> method), 292
<code>save()</code> (<code>evalml.pipelines.components.ElasticNetClassifier</code> method), 249	<code>save()</code> (<code>evalml.pipelines.MeanBaselineRegressionPipeline</code> method), 178
<code>save()</code> (<code>evalml.pipelines.components.ElasticNetRegressor</code> method), 280	<code>save()</code> (<code>evalml.pipelines.ModeBaselineBinaryPipeline</code> method), 164
<code>save()</code> (<code>evalml.pipelines.components.Estimator</code> method), 192	<code>save()</code> (<code>evalml.pipelines.ModeBaselineMulticlassPipeline</code> method), 169
<code>save()</code> (<code>evalml.pipelines.components.ExtraTreesClassifiers</code> method), 252	<code>save()</code> (<code>evalml.pipelines.MulticlassClassificationPipeline</code> method), 127
<code>save()</code> (<code>evalml.pipelines.components.ExtraTreesRegressor</code> method), 286	<code>save()</code> (<code>evalml.pipelines.PipelineBase</code> method), 114
<code>save()</code> (<code>evalml.pipelines.components.Imputer</code> method), 214	<code>save()</code> (<code>evalml.pipelines.RegressionPipeline</code> method), 132
<code>save()</code> (<code>evalml.pipelines.components.LightGBMClassifier</code> method), 258	<code>save()</code> (<code>evalml.pipelines.TimeSeriesBaselineRegressionPipeline</code> method), 183
<code>save()</code> (<code>evalml.pipelines.components.LightGBMRegressor</code> method), 307	<code>save()</code> (<code>evalml.pipelines.TimeSeriesBinaryClassificationPipeline</code> method), 141
<code>save()</code> (<code>evalml.pipelines.components.LinearRegressor</code> method), 283	<code>save()</code> (<code>evalml.pipelines.TimeSeriesClassificationPipeline</code> method), 136
<code>save()</code> (<code>evalml.pipelines.components.LogisticRegressionClassifier</code> method), 261	<code>save()</code> (<code>evalml.pipelines.TimeSeriesMulticlassClassificationPipeline</code> method), 145
<code>save()</code> (<code>evalml.pipelines.components.OneHotEncoder</code> method), 204	<code>save()</code> (<code>evalml.pipelines.TimeSeriesRegressionPipeline</code> method), 150
<code>save()</code> (<code>evalml.pipelines.components.PerColumnImputer</code> method), 210	<code>score()</code> (<code>evalml.objectives.AccuracyBinary</code> method), 341
<code>save()</code> (<code>evalml.pipelines.components.RandomForestClassifier</code> method), 255	<code>score()</code> (<code>evalml.objectives.AccuracyMulticlass</code> method), 343
<code>save()</code> (<code>evalml.pipelines.components.RandomForestRegressor</code> method), 289	<code>score()</code> (<code>evalml.objectives.AUC</code> method), 346
<code>save()</code> (<code>evalml.pipelines.components.RFClassifierSelectFromModel</code> method), 226	<code>score()</code> (<code>evalml.objectives.AUCMacro</code> method), 348
<code>save()</code> (<code>evalml.pipelines.components.RFRegressorSelectFromModel</code> method), 223	<code>score()</code> (<code>evalml.objectives.AUCMicro</code> method), 350
<code>save()</code> (<code>evalml.pipelines.components.SelectColumns</code> method), 200	<code>score()</code> (<code>evalml.objectives.AUCWeighted</code> method), 352
<code>save()</code> (<code>evalml.pipelines.components.SimpleImputer</code> method), 217	<code>score()</code> (<code>evalml.objectives.BalancedAccuracyBinary</code> method), 355
<code>save()</code> (<code>evalml.pipelines.components.StackedEnsembleClassifier</code> method), 270	<code>score()</code> (<code>evalml.objectives.BalancedAccuracyMulticlass</code> method), 357
<code>save()</code> (<code>evalml.pipelines.components.StackedEnsembleRegressor</code> method), 301	<code>score()</code> (<code>evalml.objectives.BinaryClassificationObjective</code> method), 323
<code>save()</code> (<code>evalml.pipelines.components.StandardScaler</code> method), 220	<code>score()</code> (<code>evalml.objectives.CostBenefitMatrix</code> method), 337
<code>save()</code> (<code>evalml.pipelines.components.TargetEncoder</code> method), 207	<code>score()</code> (<code>evalml.objectives.ExpVariance</code> method), 412
<code>save()</code> (<code>evalml.pipelines.components.TextFeaturizer</code> method), 236	<code>score()</code> (<code>evalml.objectives.F1</code> method), 360
<code>save()</code> (<code>evalml.pipelines.components.TimeSeriesBaselineRegressor</code> method), 298	<code>score()</code> (<code>evalml.objectives.F1Macro</code> method), 364
<code>save()</code> (<code>evalml.pipelines.components.Transformer</code> method), 189	<code>score()</code> (<code>evalml.objectives.F1Micro</code> method), 362
	<code>score()</code> (<code>evalml.objectives.F1Weighted</code> method), 366
	<code>score()</code> (<code>evalml.objectives.FraudCost</code> method), 331
	<code>score()</code> (<code>evalml.objectives.LeadScoring</code> method), 334
	<code>score()</code> (<code>evalml.objectives.LogLossBinary</code> method), 369
	<code>score()</code> (<code>evalml.objectives.LogLossMulticlass</code> method), 371
	<code>score()</code> (<code>evalml.objectives.MAE</code> method), 399
	<code>score()</code> (<code>evalml.objectives.MAPE</code> method), 401

- `score()` (*evalml.objectives.MaxError method*), 410
- `score()` (*evalml.objectives.MCCBinary method*), 374
- `score()` (*evalml.objectives.MCCMulticlass method*), 376
- `score()` (*evalml.objectives.MeanSquaredLogError method*), 406
- `score()` (*evalml.objectives.MedianAE method*), 408
- `score()` (*evalml.objectives.MSE method*), 404
- `score()` (*evalml.objectives.MulticlassClassificationObjective method*), 325
- `score()` (*evalml.objectives.ObjectiveBase method*), 320
- `score()` (*evalml.objectives.Precision method*), 379
- `score()` (*evalml.objectives.PrecisionMacro method*), 383
- `score()` (*evalml.objectives.PrecisionMicro method*), 381
- `score()` (*evalml.objectives.PrecisionWeighted method*), 385
- `score()` (*evalml.objectives.R2 method*), 397
- `score()` (*evalml.objectives.Recall method*), 388
- `score()` (*evalml.objectives.RecallMacro method*), 392
- `score()` (*evalml.objectives.RecallMicro method*), 390
- `score()` (*evalml.objectives.RecallWeighted method*), 394
- `score()` (*evalml.objectives.RegressionObjective method*), 327
- `score()` (*evalml.objectives.RootMeanSquaredError method*), 414
- `score()` (*evalml.objectives.RootMeanSquaredLogError method*), 417
- `score()` (*evalml.pipelines.BaselineBinaryPipeline method*), 155
- `score()` (*evalml.pipelines.BaselineMulticlassPipeline method*), 160
- `score()` (*evalml.pipelines.BaselineRegressionPipeline method*), 174
- `score()` (*evalml.pipelines.BinaryClassificationPipeline method*), 123
- `score()` (*evalml.pipelines.ClassificationPipeline method*), 119
- `score()` (*evalml.pipelines.MeanBaselineRegressionPipeline method*), 178
- `score()` (*evalml.pipelines.ModeBaselineBinaryPipeline method*), 164
- `score()` (*evalml.pipelines.ModeBaselineMulticlassPipeline method*), 169
- `score()` (*evalml.pipelines.MulticlassClassificationPipeline method*), 128
- `score()` (*evalml.pipelines.PipelineBase method*), 114
- `score()` (*evalml.pipelines.RegressionPipeline method*), 132
- `score()` (*evalml.pipelines.TimeSeriesBaselineRegressionPipeline method*), 183
- `score()` (*evalml.pipelines.TimeSeriesBinaryClassificationPipeline method*), 141
- `score()` (*evalml.pipelines.TimeSeriesClassificationPipeline method*), 136
- `score()` (*evalml.pipelines.TimeSeriesMulticlassClassificationPipeline method*), 146
- `score()` (*evalml.pipelines.TimeSeriesRegressionPipeline method*), 150
- `search()` (*evalml.automl.AutoMLSearch method*), 104
- `SelectColumns` (class in *evalml.pipelines.components*), 198
- `silent_error_callback()` (in module *evalml.automl.callbacks*), 110
- `SimpleImputer` (class in *evalml.pipelines.components*), 215
- `SKOptTuner` (class in *evalml.tuners*), 423
- `split_data()` (in module *evalml.preprocessing*), 100
- `StackedEnsembleClassifier` (class in *evalml.pipelines.components*), 267
- `StackedEnsembleRegressor` (class in *evalml.pipelines.components*), 298
- `StandardScaler` (class in *evalml.pipelines.components*), 218
- `summary` (*evalml.pipelines.BaselineBinaryPipeline attribute*), 151
- `summary` (*evalml.pipelines.BaselineMulticlassPipeline attribute*), 156
- `summary` (*evalml.pipelines.BaselineRegressionPipeline attribute*), 170
- `summary` (*evalml.pipelines.MeanBaselineRegressionPipeline attribute*), 175
- `summary` (*evalml.pipelines.ModeBaselineBinaryPipeline attribute*), 160
- `summary` (*evalml.pipelines.ModeBaselineMulticlassPipeline attribute*), 165
- `summary` (*evalml.pipelines.TimeSeriesBaselineRegressionPipeline attribute*), 179
- `supported_problem_types` (*evalml.pipelines.components.BaselineClassifier attribute*), 264
- `supported_problem_types` (*evalml.pipelines.components.BaselineRegressor attribute*), 292
- `supported_problem_types` (*evalml.pipelines.components.CatBoostClassifier attribute*), 243
- `supported_problem_types` (*evalml.pipelines.components.CatBoostRegressor attribute*), 274
- `supported_problem_types` (*evalml.pipelines.components.DecisionTreeClassifier attribute*), 271
- `supported_problem_types` (*evalml.pipelines.components.DecisionTreeRegressor attribute*), 271

attribute), 302
 supported_problem_types (evalml.pipelines.components.ElasticNetClassifier attribute), 246
 supported_problem_types (evalml.pipelines.components.ElasticNetRegressor attribute), 277
 supported_problem_types (evalml.pipelines.components.ExtraTreesClassifier attribute), 249
 supported_problem_types (evalml.pipelines.components.ExtraTreesRegressor attribute), 283
 supported_problem_types (evalml.pipelines.components.LightGBMClassifier attribute), 255
 supported_problem_types (evalml.pipelines.components.LightGBMRegressor attribute), 305
 supported_problem_types (evalml.pipelines.components.LinearRegressor attribute), 280
 supported_problem_types (evalml.pipelines.components.LogisticRegressionClassifier attribute), 258
 supported_problem_types (evalml.pipelines.components.RandomForestClassifier attribute), 252
 supported_problem_types (evalml.pipelines.components.RandomForestRegressor attribute), 286
 supported_problem_types (evalml.pipelines.components.StackedEnsembleClassifier attribute), 267
 supported_problem_types (evalml.pipelines.components.StackedEnsembleRegressor attribute), 298
 supported_problem_types (evalml.pipelines.components.TimeSeriesBaselineRegressor attribute), 295
 supported_problem_types (evalml.pipelines.components.XGBoostClassifier attribute), 261
 supported_problem_types (evalml.pipelines.components.XGBoostRegressor attribute), 289

TextFeaturizer (class in evalml.pipelines.components), 234
 TimeSeriesBaselineRegressionPipeline (class in evalml.pipelines), 179
 TimeSeriesBaselineRegressor (class in evalml.pipelines.components), 295
 TimeSeriesBinaryClassificationPipeline (class in evalml.pipelines), 137
 TimeSeriesClassificationPipeline (class in evalml.pipelines), 133
 TimeSeriesMulticlassClassificationPipeline (class in evalml.pipelines), 142
 TimeSeriesRegressionPipeline (class in evalml.pipelines), 146
 to_dict() (evalml.data_checks.DataCheckError method), 445
 to_dict() (evalml.data_checks.DataCheckMessage method), 444
 to_dict() (evalml.data_checks.DataCheckWarning method), 447
 transform() (evalml.pipelines.components.DateTimeFeaturizer method), 233
 transform() (evalml.pipelines.components.DelayedFeatureTransformer method), 239
 transform() (evalml.pipelines.components.DFSTransformer method), 242
 transform() (evalml.pipelines.components.DropColumns method), 197
 transform() (evalml.pipelines.components.DropNullColumns method), 230
 transform() (evalml.pipelines.components.Imputer method), 214
 transform() (evalml.pipelines.components.OneHotEncoder method), 204
 transform() (evalml.pipelines.components.PerColumnImputer method), 211
 transform() (evalml.pipelines.components.RFClassifierSelectFromModel method), 227
 transform() (evalml.pipelines.components.RFRegressorSelectFromModel method), 223
 transform() (evalml.pipelines.components.SelectColumns method), 200
 transform() (evalml.pipelines.components.SimpleImputer method), 217
 transform() (evalml.pipelines.components.StandardScaler method), 220
 transform() (evalml.pipelines.components.TargetEncoder method), 207
 transform() (evalml.pipelines.components.TextFeaturizer method), 236
 transform() (evalml.pipelines.components.Transformer method), 190
 Transformer (class in evalml.pipelines.components), 187

T
 target_distribution() (in module evalml.preprocessing), 99
 TargetEncoder (class in evalml.pipelines.components), 204
 TargetLeakageDataCheck (class in evalml.data_checks), 434

Tuner (class in `evalml.tuners`), 421

V

`validate()` (`evalml.data_checks.ClassImbalanceDataCheck` method), 440

`validate()` (`evalml.data_checks.DataCheck` method), 429

`validate()` (`evalml.data_checks.DataChecks` method), 441

`validate()` (`evalml.data_checks.DefaultDataChecks` method), 443

`validate()` (`evalml.data_checks.HighlyNullDataCheck` method), 432

`validate()` (`evalml.data_checks.IDColumnsDataCheck` method), 433

`validate()` (`evalml.data_checks.InvalidTargetDataCheck` method), 430

`validate()` (`evalml.data_checks.NoVarianceDataCheck` method), 438

`validate()` (`evalml.data_checks.OutliersDataCheck` method), 437

`validate()` (`evalml.data_checks.TargetLeakageDataCheck` method), 435

`validate_inputs()` (`evalml.objectives.AccuracyBinary` method), 341

`validate_inputs()` (`evalml.objectives.AccuracyMulticlass` method), 343

`validate_inputs()` (`evalml.objectives.AUC` method), 346

`validate_inputs()` (`evalml.objectives.AUCMacro` method), 349

`validate_inputs()` (`evalml.objectives.AUCMicro` method), 351

`validate_inputs()` (`evalml.objectives.AUCWeighted` method), 353

`validate_inputs()` (`evalml.objectives.BalancedAccuracyBinary` method), 356

`validate_inputs()` (`evalml.objectives.BalancedAccuracyMulticlass` method), 358

`validate_inputs()` (`evalml.objectives.BinaryClassificationObjective` method), 323

`validate_inputs()` (`evalml.objectives.CostBenefitMatrix` method), 337

`validate_inputs()` (`evalml.objectives.ExpVariance` method), 413

`validate_inputs()` (`evalml.objectives.F1` method), 361

`validate_inputs()` (`evalml.objectives.F1Macro` method), 365

`validate_inputs()` (`evalml.objectives.F1Micro` method), 363

`validate_inputs()` (`evalml.objectives.F1Weighted` method), 367

`validate_inputs()` (`evalml.objectives.FraudCost` method), 331

`validate_inputs()` (`evalml.objectives.LeadScoring` method), 334

`validate_inputs()` (`evalml.objectives.LogLossBinary` method), 370

`validate_inputs()` (`evalml.objectives.LogLossMulticlass` method), 372

`validate_inputs()` (`evalml.objectives.MAE` method), 400

`validate_inputs()` (`evalml.objectives.MAPE` method), 402

`validate_inputs()` (`evalml.objectives.MaxError` method), 411

`validate_inputs()` (`evalml.objectives.MCCBinary` method), 375

`validate_inputs()` (`evalml.objectives.MCCMulticlass` method), 377

`validate_inputs()` (`evalml.objectives.MeanSquaredLogError` method), 406

`validate_inputs()` (`evalml.objectives.MedianAE` method), 409

`validate_inputs()` (`evalml.objectives.MSE` method), 404

`validate_inputs()` (`evalml.objectives.MulticlassClassificationObjective` method), 326

`validate_inputs()` (`evalml.objectives.ObjectiveBase` method), 320

`validate_inputs()` (`evalml.objectives.Precision` method), 380

`validate_inputs()` (`evalml.objectives.PrecisionMacro` method), 384

`validate_inputs()` (`evalml.objectives.PrecisionMicro` method), 382

`validate_inputs()` (`evalml.objectives.PrecisionWeighted` method), 386

`validate_inputs()` (`evalml.objectives.R2` method), 397

`validate_inputs()` (`evalml.objectives.Recall` method), 397

[method](#)), [389](#)
`validate_inputs()`
 ([evalml.objectives.RecallMacro](#) [method](#)),
 [393](#)
`validate_inputs()` ([evalml.objectives.RecallMicro](#)
 [method](#)), [391](#)
`validate_inputs()`
 ([evalml.objectives.RecallWeighted](#) [method](#)),
 [395](#)
`validate_inputs()`
 ([evalml.objectives.RegressionObjective](#)
 [method](#)), [328](#)
`validate_inputs()`
 ([evalml.objectives.RootMeanSquaredError](#)
 [method](#)), [415](#)
`validate_inputs()`
 ([evalml.objectives.RootMeanSquaredLogError](#)
 [method](#)), [417](#)

X

XGBoostClassifier (class in
 [evalml.pipelines.components](#)), [261](#)
XGBoostRegressor (class in
 [evalml.pipelines.components](#)), [289](#)