

---

# **EvalML Documentation**

***Release 0.13.1***

**Alteryx Innovation Labs**

**Aug 25, 2020**



# CONTENTS

<b>1</b>	<b>Install</b>	<b>3</b>
<b>2</b>	<b>Start</b>	<b>5</b>
<b>3</b>	<b>Tutorials</b>	<b>9</b>
<b>4</b>	<b>User Guide</b>	<b>25</b>
<b>5</b>	<b>API Reference</b>	<b>61</b>
<b>6</b>	<b>Release Notes</b>	<b>287</b>
	<b>Index</b>	<b>305</b>



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
```

### **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

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.



## 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()
```

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.

```
[3]: automl = AutoMLSearch(problem_type="binary", objective="f1", max_pipelines=5)
```

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, test_
      ↪size=.2)
```

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(X_train, y_train)

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

Optimizing for F1.
Greater score is better.
```

(continues on next page)

(continued from previous page)

```
Searching up to 5 pipelines.
Allowed model families: extra_trees, xgboost, random_forest, catboost, linear_model
```

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

```
(1/5) Mode Baseline Binary Classification P... Elapsed:00:00
      Starting cross validation
      Finished cross validation - mean F1: 0.000
(2/5) Extra Trees Classifier w/ Imputer      Elapsed:00:00
      Starting cross validation
      Finished cross validation - mean F1: 0.936
(3/5) Elastic Net Classifier w/ Imputer + S... Elapsed:00:02
      Starting cross validation
      Finished cross validation - mean F1: 0.586
(4/5) CatBoost Classifier w/ Imputer        Elapsed:00:02
      Starting cross validation
      Finished cross validation - mean F1: 0.930
(5/5) XGBoost Classifier w/ Imputer        Elapsed:00:03
      Starting cross validation
      Finished cross validation - mean F1: 0.939
```

```
Search finished after 00:12
Best pipeline: XGBoost Classifier w/ Imputer
Best pipeline F1: 0.938817
```

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
```

```
[6]:   id                pipeline_name      score  \
0    4      XGBoost Classifier w/ Imputer  0.938817
1    1      Extra Trees Classifier w/ Imputer  0.936156
2    3      CatBoost Classifier w/ Imputer  0.929566
3    2  Elastic Net Classifier w/ Imputer + Standard S...  0.586028
4    0  Mode Baseline Binary Classification Pipeline  0.000000

      percent_better_than_baseline  high_variance_cv  \
0                                NaN                False
1                                NaN                False
2                                NaN                False
3                                NaN                False
4                                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  {'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)

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

Problem Type: Binary Classification
Model Family: CatBoost

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

Training
=====
Training for Binary Classification problems.
Total training time (including CV): 0.7 seconds

Cross Validation
-----
```

	F1	Accuracy	Binary	Balanced Accuracy	Binary	Precision	AUC	Log
→ Loss Binary	MCC	Binary #	Training #	Testing #				
0	0.991		0.993		0.991	1.000	0.999	
→ 0.366	0.986	303.000	152.000		0.902	0.877	0.971	
1	0.877		0.908		0.902	0.877	0.971	
→ 0.398	0.804	303.000	152.000		0.938	0.912	0.984	
2	0.920		0.940		0.938	0.912	0.984	
→ 0.388	0.873	304.000	151.000		0.944	0.930	0.985	
mean	0.930		0.947		0.944	0.930	0.985	
→ 0.384	0.887	-	-		0.045	0.063	0.014	
std	0.058		0.043		0.045	0.063	0.014	
→ 0.016	0.092	-	-		0.048	0.068	0.014	
coef of var	0.062		0.046		0.048	0.068	0.014	
→ 0.042	0.104	-	-					

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}, 'CatBoost Classifier': {'n_estimators': 10, 'eta': 0.
→ 03, 'max_depth': 6, 'bootstrap_type': None}}
```

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', 0.9367088607594937)])
```

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, you can set your 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 a 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
Labels
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]: X = X.drop(['datetime', 'expiration_date', 'country', 'region', 'provider'], axis=1)

X_train, X_holdout, y_train, y_holdout = evalml.preprocessing.split_data(X, y, test_
↪size=0.2, random_state=0)
```

```
print(X.dtypes)
```

```
card_id          int64
store_id         int64
amount          int64
currency         object
customer_present bool
lat             float64
lng             float64
dtype: object
```

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

```
[5]: automl = AutoMLSearch(problem_type='binary',
                           objective=fraud_objective,
                           additional_objectives=['auc', 'f1', 'precision'],
                           max_pipelines=5,
                           optimize_thresholds=True)
```

```
automl.search(X_train, y_train)
```

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

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

```
Optimizing for Fraud Cost.
Lower score is better.
```

```
Searching up to 5 pipelines.
```

```
Allowed model families: catboost, linear_model, random_forest, extra_trees, xgboost
```

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

```
(1/5) Mode Baseline Binary Classification P... Elapsed:00:00
      Starting cross validation
      Finished cross validation - mean Fraud Cost: 0.023
(2/5) Extra Trees Classifier w/ Imputer + O... Elapsed:00:01
      Starting cross validation
      Finished cross validation - mean Fraud Cost: 0.002
(3/5) Elastic Net Classifier w/ Imputer + O... Elapsed:00:04
      Starting cross validation
      Finished cross validation - mean Fraud Cost: 0.002
(4/5) CatBoost Classifier w/ Imputer          Elapsed:00:06
      Starting cross validation
      Finished cross validation - mean Fraud Cost: 0.002
(5/5) XGBoost Classifier w/ Imputer + One H... Elapsed:00:07
      Starting cross validation
      Finished cross validation - mean Fraud Cost: 0.007

Search finished after 00:24
Best pipeline: Extra Trees Classifier w/ Imputer + One Hot Encoder
Best pipeline Fraud Cost: 0.002316
```

## 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 fraud detection objective we defined

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

```
[6]:   id      pipeline_name      score \
0   1  Extra Trees Classifier w/ Imputer + One Hot En...  0.002316
1   2  Elastic Net Classifier w/ Imputer + One Hot En...  0.002316
2   3          CatBoost Classifier w/ Imputer  0.002316
3   4  XGBoost Classifier w/ Imputer + One Hot Encoder  0.007152
4   0      Mode Baseline Binary Classification Pipeline  0.022830

      percent_better_than_baseline  high_variance_cv \
0          89.855612          False
1          89.855612          False
2          89.855612          False
3          68.673546           True
4           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  {'Baseline Classifier': {'strategy': 'mode'}}
```

to select the best pipeline we can run

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

## Describe pipeline

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

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

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

Problem Type: Binary Classification
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
    * categories : None
    * drop : None
    * handle_unknown : ignore
    * handle_missing : error
3. Standard Scaler
4. Elastic Net Classifier
    * alpha : 0.5
    * l1_ratio : 0.5
    * n_jobs : -1
    * max_iter : 1000

Training
=====
Training for Binary Classification problems.
Objective to optimize binary classification pipeline thresholds for: <evalml.
->objectives.fraud_cost.FraudCost object at 0x7f66189e7128>
Total training time (including CV): 1.6 seconds

Cross Validation
-----
```

	Fraud Cost	AUC	F1	Precision	# Training	# Testing
0	0.002	0.500	0.247	0.141	1066.000	667.000
1	0.002	0.500	0.247	0.141	1066.000	667.000
2	0.002	0.500	0.247	0.141	1067.000	666.000
mean	0.002	0.500	0.247	0.141	-	-
std	0.000	0.000	0.000	0.000	-	-
coef of var	0.055	0.000	0.001	0.001	-	-

### 3.1.3 Evaluate on hold out

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

```
[9]: best_pipeline.fit(X_train, y_train)
[9]: <evalml.pipelines.utils.make_pipeline.<locals>.GeneratedPipeline at 0x7f65f173ac18>
```

Now, we can score the pipeline on the hold out data using both the fraud cost score and the AUC.



```
[10]: best_pipeline.score(X_holdout, y_holdout, objectives=["auc", fraud_objective])
[10]: OrderedDict([('AUC', 0.8036212624584718),
                  ('Fraud Cost', 0.013665469538525703)])
```

### 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(problem_type='binary',
                                objective='auc',
                                additional_objectives=['f1', 'precision'],
                                max_pipelines=5,
                                optimize_thresholds=True)

automl_auc.search(X_train, y_train)

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

Optimizing for AUC.
Greater score is better.

Searching up to 5 pipelines.
Allowed model families: catboost, linear_model, random_forest, extra_trees, xgboost

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

(1/5) Mode Baseline Binary Classification P... Elapsed:00:00
      Starting cross validation
      Finished cross validation - mean AUC: 0.500
(2/5) Extra Trees Classifier w/ Imputer + O... Elapsed:00:00
      Starting cross validation
      Finished cross validation - mean AUC: 0.826
(3/5) Elastic Net Classifier w/ Imputer + O... Elapsed:00:02
      Starting cross validation
      Finished cross validation - mean AUC: 0.500
(4/5) CatBoost Classifier w/ Imputer          Elapsed:00:02
      Starting cross validation
      Finished cross validation - mean AUC: 0.844
(5/5) XGBoost Classifier w/ Imputer + One H... Elapsed:00:03
      Starting cross validation
      Finished cross validation - mean AUC: 0.852

Search finished after 00:24
Best pipeline: XGBoost Classifier w/ Imputer + One Hot Encoder
Best pipeline AUC: 0.851945
```

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

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

```
[12]:      id      pipeline_name      score  \
0    4  XGBoost Classifier w/ Imputer + One Hot Encoder  0.851945
1    3      CatBoost Classifier w/ Imputer  0.843946
2    1  Extra Trees Classifier w/ Imputer + One Hot En...  0.825523
3    0      Mode Baseline Binary Classification Pipeline  0.500000
4    2  Elastic Net Classifier w/ Imputer + One Hot En...  0.500000

      percent_better_than_baseline  high_variance_cv  \
0                70.389091                False
1                68.789178                False
2                65.104501                False
3                 0.000000                False
4                 0.000000                False

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

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

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

```
[13]: <evalml.pipelines.utils.make_pipeline.<locals>.GeneratedPipeline at 0x7f65f004beb8>
```

```
[14]: # get the fraud score on holdout data
```

```
best_pipeline_auc.score(X_holdout, y_holdout, objectives=["auc", fraud_objective])
```

```
[14]: OrderedDict([('AUC', 0.8529235880398671),
                ('Fraud Cost', 0.004329350526560073)])
```

```
[15]: # fraud score on fraud optimized again
```

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

```
[15]: OrderedDict([('AUC', 0.8036212624584718),
                ('Fraud Cost', 0.013665469538525703)])
```

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

      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

### 3.2.3 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

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

```
[4]: X_train, X_holdout, y_train, y_holdout = evalml.preprocessing.split_data(X, y, test_
      ↪size=0.2, random_state=0)

print(X.dtypes)

job          object
state        object
zip          float64
action       object
amount       float64
dtype: object
```

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

```
[5]: automl = AutoMLSearch(problem_type='binary',
                           objective=lead_scoring_objective,
                           additional_objectives=['auc'],
                           max_pipelines=5,
                           optimize_thresholds=True)

automl.search(X_train, y_train)

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

Optimizing for Lead Scoring.
Greater score is better.

Searching up to 5 pipelines.
Allowed model families: linear_model, catboost, random_forest, xgboost, extra_trees

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

(1/5) Mode Baseline Binary Classification P... Elapsed:00:00
      Starting cross validation
      Finished cross validation - mean Lead Scoring: 0.000
(2/5) Extra Trees Classifier w/ Imputer + O... Elapsed:00:02
      Starting cross validation
      Finished cross validation - mean Lead Scoring: 34.437
(3/5) Elastic Net Classifier w/ Imputer + O... Elapsed:00:06
      Starting cross validation
      Finished cross validation - mean Lead Scoring: 42.140
(4/5) CatBoost Classifier w/ Imputer           Elapsed:00:08
      Starting cross validation
      Finished cross validation - mean Lead Scoring: 36.677
```

(continues on next page)

(continued from previous page)

```
(5/5) XGBoost Classifier w/ Imputer + One H... Elapsed:00:09
      Starting cross validation
      Finished cross validation - mean Lead Scoring: 41.646

Search finished after 00:27
Best pipeline: Elastic Net Classifier w/ Imputer + One Hot Encoder + Standard Scaler
Best pipeline Lead Scoring: 42.140250
```

## 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

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

	id	pipeline_name	score	\
0	2	Elastic Net Classifier w/ Imputer + One Hot En...	42.140250	
1	4	XGBoost Classifier w/ Imputer + One Hot Encoder	41.645611	
2	3	CatBoost Classifier w/ Imputer	36.676720	
3	1	Extra Trees Classifier w/ Imputer + One Hot En...	34.437241	
4	0	Mode Baseline Binary Classification Pipeline	0.000000	

	percent_better_than_baseline	high_variance_cv	\
0	NaN	False	
1	NaN	False	
2	NaN	False	
3	NaN	True	
4	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 {'Baseline Classifier': {'strategy': 'mode'}}

```

to select the best pipeline we can run

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

## Describe pipeline

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

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

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

Problem Type: Binary Classification
Model Family: Linear

Pipeline Steps
```

(continues on next page)

(continued from previous page)

```

=====
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
    * categories : None
    * drop : None
    * handle_unknown : ignore
    * handle_missing : error
3. Standard Scaler
4. Elastic Net Classifier
    * alpha : 0.5
    * l1_ratio : 0.5
    * n_jobs : -1
    * max_iter : 1000

Training
=====
Training for Binary Classification problems.
Objective to optimize binary classification pipeline thresholds for: <evalml.
↳objectives.lead_scoring.LeadScoring object at 0x7f335513e128>
Total training time (including CV): 1.9 seconds

Cross Validation
-----

```

	Lead Scoring	AUC	# Training	# Testing
0	42.129	0.500	2479.000	1550.000
1	42.129	0.500	2479.000	1550.000
2	42.163	0.500	2480.000	1549.000
mean	42.140	0.500	-	-
std	0.019	0.000	-	-
coef of var	0.000	0.000	-	-

### 3.2.4 Evaluate on hold out

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

```

[9]: best_pipeline.fit(X_train, y_train)
[9]: <evalml.pipelines.utils.make_pipeline.<locals>.GeneratedPipeline at 0x7f332c138b00>

```

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

```

[10]: best_pipeline.score(X_holdout, y_holdout, objectives=["auc", lead_scoring_objective])
[10]: OrderedDict([('AUC', 0.5), ('Lead Scoring', 0.0)])

```

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

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

```
[11]: automl_auc = evalml.AutoMLSearch(problem_type='binary',
                                     objective='auc',
                                     additional_objectives=[],
                                     max_pipelines=5,
                                     optimize_thresholds=True)

automl_auc.search(X_train, y_train)

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

Optimizing for AUC.
Greater score is better.

Searching up to 5 pipelines.
Allowed model families: linear_model, catboost, random_forest, xgboost, extra_trees

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

(1/5) Mode Baseline Binary Classification P... Elapsed:00:00
      Starting cross validation
      Finished cross validation - mean AUC: 0.500
(2/5) Extra Trees Classifier w/ Imputer + O... Elapsed:00:00
      Starting cross validation
      Finished cross validation - mean AUC: 0.716
(3/5) Elastic Net Classifier w/ Imputer + O... Elapsed:00:01
      Starting cross validation
      Finished cross validation - mean AUC: 0.500
(4/5) CatBoost Classifier w/ Imputer          Elapsed:00:02
      Starting cross validation
      Finished cross validation - mean AUC: 0.582
(5/5) XGBoost Classifier w/ Imputer + One H... Elapsed:00:02
      Starting cross validation
      Finished cross validation - mean AUC: 0.728

Search finished after 00:32
Best pipeline: XGBoost Classifier w/ Imputer + One Hot Encoder
Best pipeline AUC: 0.728289
```

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

```
[12]: automl_auc.rankings

[12]:   id      pipeline_name      score \
0    4  XGBoost Classifier w/ Imputer + One Hot Encoder  0.728289
1    1  Extra Trees Classifier w/ Imputer + One Hot En...  0.716259
2    3                CatBoost Classifier w/ Imputer  0.581861
3    0      Mode Baseline Binary Classification Pipeline  0.500000
4    2  Elastic Net Classifier w/ Imputer + One Hot En...  0.500000

      percent_better_than_baseline  high_variance_cv \
0                                45.657808      False
1                                43.251821      False
```

(continues on next page)

(continued from previous page)

```

2          16.372274          False
3          0.000000          False
4          0.000000          False

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

```

```

[13]: best_pipeline_auc = automl_auc.best_pipeline

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

[13]: <evalml.pipelines.utils.make_pipeline.<locals>.GeneratedPipeline at 0x7f332c0f4e10>

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

[14]: OrderedDict([('AUC', 0.6662964641885766),
                  ('Lead Scoring', -0.051590713671539126)])

```

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
X, y = load_churn()
```

```

          Number of Features
Categorical                16
Numeric                    3

Number of training examples: 7043
Labels
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(problem_type='binary', objective='log_loss_binary')
automl.search(X, y)

ll_pipeline = automl.best_pipeline
ll_pipeline.fit(X, y)
ll_pipeline.score(X, y, ['log_loss_binary'])
```

Using default limit of max\_pipelines=5.

Generating pipelines to search over...

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

Optimizing for Log Loss Binary.  
Lower score is better.

Searching up to 5 pipelines.  
Allowed model families: random\_forest, xgboost, catboost, extra\_trees, linear\_model

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

(1/5) Mode Baseline Binary Classification P... Elapsed:00:00  
Starting cross validation  
Finished cross validation - mean Log Loss Binary: 9.166

(2/5) Extra Trees Classifier w/ Imputer + O... Elapsed:00:00  
Starting cross validation  
Finished cross validation - mean Log Loss Binary: 0.431

(3/5) Elastic Net Classifier w/ Imputer + O... Elapsed:00:04  
Starting cross validation  
Finished cross validation - mean Log Loss Binary: 0.579

(4/5) CatBoost Classifier w/ Imputer Elapsed:00:06  
Starting cross validation  
Finished cross validation - mean Log Loss Binary: 0.600

(5/5) XGBoost Classifier w/ Imputer + One H... Elapsed:00:07  
Starting cross validation  
Finished cross validation - mean Log Loss Binary: 0.433

Search finished after 01:31  
Best pipeline: Extra Trees Classifier w/ Imputer + One Hot Encoder  
Best pipeline Log Loss Binary: 0.431262

```
[3]: OrderedDict([('Log Loss Binary', 0.4204150403308955)])
```

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. When we 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, we get approximately 14.851, indicating \$14.85 in profit per customer. Across our 7043 customers, we get a total of approximately \$104,588.55 in profit.

```
[4]: ll_pipeline.score(X, y, [cost_benefit_matrix])
[4]: OrderedDict([('Cost Benefit Matrix', 15.973306829476066)])
```

## 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.

```
[5]: automl = AutoMLSearch(problem_type='binary', objective=cost_benefit_matrix)
      automl.search(X, y)

cbm_pipeline = automl.best_pipeline
cbm_pipeline.fit(X, y)

Using default limit of max_pipelines=5.

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

Optimizing for Cost Benefit Matrix.
Greater score is better.

Searching up to 5 pipelines.
Allowed model families: random_forest, xgboost, catboost, extra_trees, linear_model

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

(1/5) Mode Baseline Binary Classification P... Elapsed:00:00
      Starting cross validation
      Finished cross validation - mean Cost Benefit Matrix: -53.074
(2/5) Extra Trees Classifier w/ Imputer + O... Elapsed:00:00
      Starting cross validation
      Finished cross validation - mean Cost Benefit Matrix: 13.474
(3/5) Elastic Net Classifier w/ Imputer + O... Elapsed:00:03
      Starting cross validation
      Finished cross validation - mean Cost Benefit Matrix: -53.074
(4/5) CatBoost Classifier w/ Imputer          Elapsed:00:05
      Starting cross validation
      Finished cross validation - mean Cost Benefit Matrix: 16.087
(5/5) XGBoost Classifier w/ Imputer + One H... Elapsed:00:07
      Starting cross validation
      Finished cross validation - mean Cost Benefit Matrix: 23.129

Search finished after 01:28
Best pipeline: XGBoost Classifier w/ Imputer + One Hot Encoder
Best pipeline Cost Benefit Matrix: 23.129440

[5]: <evalml.pipelines.utils.make_pipeline.<locals>.GeneratedPipeline at 0x7f932e069320>
```

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 \$42.24 in profit per customer. Across our 7043 customers, we get a total of approximately \$297,496.32 in profit—much more than the profit returned from our previous 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.

```
[6]: cbm_pipeline.score(X, y, [cost_benefit_matrix])  
[6]: OrderedDict([('Cost Benefit Matrix', 42.2405225046145)])
```

Finally, we can graph the confusion matrices for both pipelines to show that our pipeline trained with the cost-benefit matrix is able to correctly classify more samples than the pipeline trained with log loss.

```
[7]: from evalml.model_understanding.graphs import graph_confusion_matrix  
  
# pipeline trained with log loss  
y_pred = ll_pipeline.predict(X)  
graph_confusion_matrix(y, y_pred)
```

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

```
[8]: # pipeline trained with cost-benefit matrix  
y_pred = cbm_pipeline.predict(X)  
graph_confusion_matrix(y, y_pred)
```

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

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.

```
[1]: import evalml

X, y = evalml.demos.load_breast_cancer()

automl = evalml.automl.AutoMLSearch(problem_type='binary')
automl.search(X, y)

Using default limit of max_pipelines=5.

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

Optimizing for Log Loss Binary.
Lower score is better.

Searching up to 5 pipelines.
Allowed model families: linear_model, catboost, random_forest, extra_trees, xgboost

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

(1/5) Mode Baseline Binary Classification P... Elapsed:00:00
      Starting cross validation
      Finished cross validation - mean Log Loss Binary: 12.868
(2/5) Extra Trees Classifier w/ Imputer      Elapsed:00:00
      Starting cross validation
      Finished cross validation - mean Log Loss Binary: 0.146
(3/5) Elastic Net Classifier w/ Imputer + S... Elapsed:00:02
      Starting cross validation
      Finished cross validation - mean Log Loss Binary: 0.504
(4/5) CatBoost Classifier w/ Imputer          Elapsed:00:02
      Starting cross validation
      Finished cross validation - mean Log Loss Binary: 0.390
(5/5) XGBoost Classifier w/ Imputer          Elapsed:00:03
      Starting cross validation
      Finished cross validation - mean Log Loss Binary: 0.101

Search finished after 00:11
Best pipeline: XGBoost Classifier w/ Imputer
Best pipeline Log Loss Binary: 0.100965
```

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

By default, AutoML will search a fixed number of pipeline and parameter pairs (5). 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](#).

## 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.

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

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

automl_custom = evalml.automl.AutoMLSearch(problem_type='multiclass', allowed_
↪pipelines=[CustomMulticlassClassificationPipeline])

Using default limit of max_pipelines=5.
```

## 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.

```
[3]: automl.rankings
```

	id	pipeline_name	score	\
0	4	XGBoost Classifier w/ Imputer	0.100965	
1	1	Extra Trees Classifier w/ Imputer	0.146066	
2	3	CatBoost Classifier w/ Imputer	0.390101	
3	2	Elastic Net Classifier w/ Imputer + Standard S...	0.503740	
4	0	Mode Baseline Binary Classification Pipeline	12.868443	

	percent_better_than_baseline	high_variance_cv	\
0	99.215404	True	
1	98.864930	False	
2	96.968547	False	
3	96.085460	False	
4	0.000000	False	

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

### 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`.

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

*****
* Extra Trees Classifier w/ Imputer *
*****

Problem Type: Binary Classification
Model Family: Extra Trees

Pipeline Steps
=====
1. Imputer
    * categorical_impute_strategy : most_frequent
    * numeric_impute_strategy : mean
    * categorical_fill_value : None
    * numeric_fill_value : None
2. 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

Training
=====
Training for Binary Classification problems.
Total training time (including CV): 1.6 seconds

Cross Validation
-----
```

	Precision	AUC	MCC	Binary # Training	Binary # Testing	Balanced Accuracy	Binary F1
0				0.157	0.942	0.925	0.917
↪984	0.990			379.000	190.000		0.
1				0.131	0.953	0.937	0.932
↪000	0.995			379.000	190.000		1.
2				0.150	0.958	0.955	0.943
↪943	0.988			380.000	189.000		0.
mean				0.146	0.951	0.939	0.931
↪976	0.991			-	-		0.
std				0.013	0.008	0.015	0.013
↪029	0.004			-	-		0.
coef of var				0.090	0.008	0.016	0.014
↪030	0.004			-	-		0.

## 4.1.5 Get Pipeline

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

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

Extra Trees Classifier w/ Imputer
{'Imputer': {'categorical_impute_strategy': 'most_frequent',
↪'numeric_impute_strategy': 'mean', 'categorical_fill_value': None,
↪'numeric_fill_value': None}, '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}}
```



(continued from previous page)

## Get best pipeline

If we specifically want to get the best pipeline, there is a convenient accessor for that.

```
[6]: best_pipeline = automl.best_pipeline
print(best_pipeline.name)
print(best_pipeline.parameters)

XGBoost Classifier w/ Imputer
{'Imputer': {'categorical_impute_strategy': 'most_frequent',
↳ 'numeric_impute_strategy': 'mean', 'categorical_fill_value': None,
↳ 'numeric_fill_value': None}, 'XGBoost Classifier': {'eta': 0.1, 'max_depth': 6,
↳ 'min_child_weight': 1, 'n_estimators': 100}}
```

## 4.1.6 Access raw results

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

```
[7]: automl.results

[7]: {'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'}},
  'score': 12.868443394958925,
  'high_variance_cv': False,
  'training_time': 0.09005975723266602,
  'cv_data': [{'all_objective_scores': OrderedDict([('Log Loss Binary',
    12.906595389677152),
    ('Accuracy Binary', 0.6263157894736842),
    ('Balanced Accuracy Binary', 0.5),
    ('F1', 0.0),
    ('Precision', 0.0),
    ('AUC', 0.5),
    ('MCC Binary', 0.0),
    ('# Training', 379),
    ('# Testing', 190)]),
    'score': 12.906595389677152,
    'binary_classification_threshold': 0.5},
  {'all_objective_scores': OrderedDict([('Log Loss Binary',
    12.906595389677149),
    ('Accuracy Binary', 0.6263157894736842),
    ('Balanced Accuracy Binary', 0.5),
    ('F1', 0.0),
    ('Precision', 0.0),
    ('AUC', 0.5),
    ('MCC Binary', 0.0),
    ('# Training', 379),
    ('# Testing', 190)]),
    'score': 12.906595389677149,
```

(continues on next page)

(continued from previous page)

```

    'binary_classification_threshold': 0.5},
    {'all_objective_scores': OrderedDict([('Log Loss Binary',
                                           12.792139405522475),
                                           ('Accuracy Binary', 0.6296296296296297),
                                           ('Balanced Accuracy Binary', 0.5),
                                           ('F1', 0.0),
                                           ('Precision', 0.0),
                                           ('AUC', 0.5),
                                           ('MCC Binary', 0.0),
                                           ('# Training', 380),
                                           ('# Testing', 189)]),
      'score': 12.792139405522475,
      'binary_classification_threshold': 0.5}},
    'percent_better_than_baseline': 0},
1: {'id': 1,
    'pipeline_name': 'Extra Trees Classifier w/ Imputer',
    'pipeline_class': evalml.pipelines.utils.make_pipeline.<locals>.GeneratedPipeline,
    'pipeline_summary': 'Extra Trees Classifier w/ Imputer',
    'parameters': {'Imputer': {'categorical_impute_strategy': 'most_frequent',
                                'numeric_impute_strategy': 'mean',
                                'categorical_fill_value': None,
                                'numeric_fill_value': None},
                    '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': 0.14606590090087035,
    'high_variance_cv': False,
    'training_time': 1.6354501247406006,
    'cv_data': [{'all_objective_scores': OrderedDict([('Log Loss Binary',
                                                         0.15683251263377887),
                                                         ('Accuracy Binary', 0.9421052631578948),
                                                         ('Balanced Accuracy Binary', 0.9253757841164635),
                                                         ('F1', 0.9172932330827067),
                                                         ('Precision', 0.9838709677419355),
                                                         ('AUC', 0.989821280624926),
                                                         ('MCC Binary', 0.8778182058245897),
                                                         ('# Training', 379),
                                                         ('# Testing', 190)]),
                  'score': 0.15683251263377887,
                  'binary_classification_threshold': 0.5},
                {'all_objective_scores': OrderedDict([('Log Loss Binary',
                                                         0.1314416667412096),
                                                         ('Accuracy Binary', 0.9526315789473684),
                                                         ('Balanced Accuracy Binary', 0.9366197183098591),
                                                         ('F1', 0.9323308270676691),
                                                         ('Precision', 1.0),
                                                         ('AUC', 0.995384069120606),
                                                         ('MCC Binary', 0.9010215251215669),
                                                         ('# Training', 379),
                                                         ('# Testing', 190)]),
                  'score': 0.1314416667412096,
                  'binary_classification_threshold': 0.5},
                {'all_objective_scores': OrderedDict([('Log Loss Binary',
                                                         0.14992352332762263),

```

(continues on next page)

(continued from previous page)

```

        ('Accuracy Binary', 0.9576719576719577),
        ('Balanced Accuracy Binary', 0.9546218487394957),
        ('F1', 0.9428571428571428),
        ('Precision', 0.9428571428571428),
        ('AUC', 0.9879951980792318),
        ('MCC Binary', 0.9092436974789916),
        ('# Training', 380),
        ('# Testing', 189)]),
    'score': 0.14992352332762263,
    'binary_classification_threshold': 0.5}},
'percent_better_than_baseline': 98.8649295301863},
2: {'id': 2,
    'pipeline_name': 'Elastic Net Classifier w/ Imputer + Standard Scaler',
    'pipeline_class': evalml.pipelines.utils.make_pipeline.<locals>.GeneratedPipeline,
    'pipeline_summary': 'Elastic Net Classifier w/ Imputer + Standard Scaler',
    'parameters': {'Imputer': {'categorical_impute_strategy': 'most_frequent',
                                'numeric_impute_strategy': 'mean',
                                'categorical_fill_value': None,
                                'numeric_fill_value': None},
                    'Elastic Net Classifier': {'alpha': 0.5,
                                                'l1_ratio': 0.5,
                                                'n_jobs': -1,
                                                'max_iter': 1000}},
    'score': 0.5037404207981783,
    'high_variance_cv': False,
    'training_time': 0.22106099128723145,
    'cv_data': [{'all_objective_scores': OrderedDict([('Log Loss Binary',
                                                        0.5106495939683122),
                                                        ('Accuracy Binary', 0.7684210526315789),
                                                        ('Balanced Accuracy Binary', 0.6901408450704225),
                                                        ('F1', 0.5510204081632654),
                                                        ('Precision', 1.0),
                                                        ('AUC', 0.9815362764824239),
                                                        ('MCC Binary', 0.5269050806054695),
                                                        ('# Training', 379),
                                                        ('# Testing', 190)]),
                'score': 0.5106495939683122,
                'binary_classification_threshold': 0.5},
                {'all_objective_scores': OrderedDict([('Log Loss Binary',
                                                        0.5088061082724736),
                                                        ('Accuracy Binary', 0.7684210526315789),
                                                        ('Balanced Accuracy Binary', 0.6901408450704225),
                                                        ('F1', 0.5510204081632654),
                                                        ('Precision', 1.0),
                                                        ('AUC', 0.9914782814534264),
                                                        ('MCC Binary', 0.5269050806054695),
                                                        ('# Training', 379),
                                                        ('# Testing', 190)]),
                'score': 0.5088061082724736,
                'binary_classification_threshold': 0.5},
                {'all_objective_scores': OrderedDict([('Log Loss Binary',
                                                        0.49176556015374906),
                                                        ('Accuracy Binary', 0.8201058201058201),
                                                        ('Balanced Accuracy Binary', 0.7571428571428571),
                                                        ('F1', 0.6792452830188679),
                                                        ('Precision', 1.0),
                                                        ('AUC', 0.9854741896758704),

```

(continues on next page)

(continued from previous page)

```

                ('MCC Binary', 0.6324555320336759),
                ('# Training', 380),
                ('# Testing', 189)]),
        'score': 0.49176556015374906,
        'binary_classification_threshold': 0.5}},
    'percent_better_than_baseline': 96.0854595591918},
3: {'id': 3,
    'pipeline_name': 'CatBoost Classifier w/ Imputer',
    'pipeline_class': evalml.pipelines.utils.make_pipeline.<locals>.GeneratedPipeline,
    'pipeline_summary': 'CatBoost Classifier w/ Imputer',
    'parameters': {'Imputer': {'categorical_impute_strategy': 'most_frequent',
                                'numeric_impute_strategy': 'mean',
                                'categorical_fill_value': None,
                                'numeric_fill_value': None},
                    'CatBoost Classifier': {'n_estimators': 10,
                                             'eta': 0.03,
                                             'max_depth': 6,
                                             'bootstrap_type': None}},
    'score': 0.3901007526814435,
    'high_variance_cv': False,
    'training_time': 0.7552793025970459,
    'cv_data': [{'all_objective_scores': OrderedDict([('Log Loss Binary',
                                                       0.3976961654004939),
                                                       ('Accuracy Binary', 0.9263157894736842),
                                                       ('Balanced Accuracy Binary', 0.9099301692507988),
                                                       ('F1', 0.8955223880597014),
                                                       ('Precision', 0.9523809523809523),
                                                       ('AUC', 0.9835483489170316),
                                                       ('MCC Binary', 0.8425009463611858),
                                                       ('# Training', 379),
                                                       ('# Testing', 190)]),
                'score': 0.3976961654004939,
                'binary_classification_threshold': 0.5},
                {'all_objective_scores': OrderedDict([('Log Loss Binary',
                                                       0.39058702666377215),
                                                       ('Accuracy Binary', 0.9631578947368421),
                                                       ('Balanced Accuracy Binary', 0.9535447982009706),
                                                       ('F1', 0.948905109489051),
                                                       ('Precision', 0.9848484848484849),
                                                       ('AUC', 0.9958574979287491),
                                                       ('MCC Binary', 0.9216584956231404),
                                                       ('# Training', 379),
                                                       ('# Testing', 190)]),
                'score': 0.39058702666377215,
                'binary_classification_threshold': 0.5},
                {'all_objective_scores': OrderedDict([('Log Loss Binary',
                                                       0.38201906598006447),
                                                       ('Accuracy Binary', 0.9629629629629629),
                                                       ('Balanced Accuracy Binary', 0.9647058823529412),
                                                       ('F1', 0.9510489510489512),
                                                       ('Precision', 0.9315068493150684),
                                                       ('AUC', 0.9885954381752702),
                                                       ('MCC Binary', 0.9218075091290715),
                                                       ('# Training', 380),
                                                       ('# Testing', 189)]),
                'score': 0.38201906598006447,
                'binary_classification_threshold': 0.5}}],

```

(continues on next page)

(continued from previous page)

```

    'percent_better_than_baseline': 96.9685474714505}},
4: {'id': 4,
    'pipeline_name': 'XGBoost Classifier w/ Imputer',
    'pipeline_class': evalml.pipelines.utils.make_pipeline.<locals>.GeneratedPipeline,
    'pipeline_summary': 'XGBoost Classifier w/ Imputer',
    'parameters': {'Imputer': {'categorical_impute_strategy': 'most_frequent',
                                'numeric_impute_strategy': 'mean',
                                'categorical_fill_value': None,
                                'numeric_fill_value': None},
                    'XGBoost Classifier': {'eta': 0.1,
                                             'max_depth': 6,
                                             'min_child_weight': 1,
                                             'n_estimators': 100}},
    'score': 0.10096524696588778,
    'high_variance_cv': True,
    'training_time': 8.603399515151978,
    'cv_data': [{'all_objective_scores': OrderedDict([('Log Loss Binary',
                                                       0.11449877552314368),
                                                       ('Accuracy Binary', 0.9578947368421052),
                                                       ('Balanced Accuracy Binary', 0.9521836903775595),
                                                       ('F1', 0.9428571428571428),
                                                       ('Precision', 0.9565217391304348),
                                                       ('AUC', 0.9915966386554622),
                                                       ('MCC Binary', 0.9097672817424011),
                                                       ('# Training', 379),
                                                       ('# Testing', 190)]),
                  'score': 0.11449877552314368,
                  'binary_classification_threshold': 0.5},
                 {'all_objective_scores': OrderedDict([('Log Loss Binary',
                                                       0.07421586432028562),
                                                       ('Accuracy Binary', 0.9736842105263158),
                                                       ('Balanced Accuracy Binary', 0.9676293052432241),
                                                       ('F1', 0.9640287769784172),
                                                       ('Precision', 0.9852941176470589),
                                                       ('AUC', 0.9959758551307847),
                                                       ('MCC Binary', 0.943843520216036),
                                                       ('# Training', 379),
                                                       ('# Testing', 190)]),
                  'score': 0.07421586432028562,
                  'binary_classification_threshold': 0.5},
                 {'all_objective_scores': OrderedDict([('Log Loss Binary',
                                                       0.11418110105423404),
                                                       ('Accuracy Binary', 0.9576719576719577),
                                                       ('Balanced Accuracy Binary', 0.9605042016806722),
                                                       ('F1', 0.9444444444444445),
                                                       ('Precision', 0.918918918918919),
                                                       ('AUC', 0.9885954381752701),
                                                       ('MCC Binary', 0.9112159507396058),
                                                       ('# Training', 380),
                                                       ('# Testing', 189)]),
                  'score': 0.11418110105423404,
                  'binary_classification_threshold': 0.5}],
    'percent_better_than_baseline': 99.21540435103876}},
'search_order': [0, 1, 2, 3, 4]}

```

## 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_objectives` method to get a list of which objectives are included with EvalML for each problem type:

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

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

```
Accuracy Binary
Balanced Accuracy Binary
F1
Precision
AUC
Log Loss Binary
MCC 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', '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  
 Labels  
 False 91.00%  
 True 9.00%  
 Name: fraud, dtype: object  
 None  
 OrderedDict([('F1', 0.6153846153846153)])  
 0.21995559455279273  
 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]: imp = SimpleImputer(impute_strategy="mean")
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: Text Featurization Component
Transformer: LSA Transformer
Transformer: Drop Null Columns Transformer
Transformer: DateTime Featurization Component
Transformer: Select Columns Transformer
```

(continues on next page)

(continued from previous page)

```

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: 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: Baseline Regressor
Estimator: Extra Trees Regressor
Estimator: XGBoost Regressor
Estimator: CatBoost Regressor
Estimator: Random Forest Regressor
Estimator: Linear Regressor
Estimator: Elastic Net Regressor
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)

    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:
```

(continues on next page)

(continued from previous page)

```

        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]

```

(continues on next page)

(continued from previous page)

```

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

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.array (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', 'linear_model', 'catboost', 'extra_trees', 'baseline',
↪ 'none']
Problem Types:
['binary', 'multiclass', 'regression']
```

## 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 = {
```

(continues on next page)

(continued from previous page)

```

        '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.

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."""
    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

```

(continues on next page)



(continued from previous page)

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

## 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` list as a class variable containing the sequence of components to be fit and evaluated. Each component in the graph can be provided as either a string name or as a reference to the component class.

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

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

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

```
[2]: 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)
```

(continues on next page)

(continued from previous page)

```
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.

```
[3]: 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_multi']))

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:

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

class CustomPipeline(MulticlassClassificationPipeline):
    component_graph = ['Simple 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.

```
[5]: class CustomPipeline(MulticlassClassificationPipeline):
      component_graph = ['Simple Imputer', 'One Hot Encoder', 'Standard Scaler',
      ↪ 'Logistic Regression Classifier']

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

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

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

Without custom hyperparameters:

```
{'Simple Imputer': {'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')}}
```

With custom hyperparameters:

```
{'Simple Imputer': {'impute_strategy': ['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.

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

[6]: <__main__.CustomPipeline at 0x7f50af602ef0>
```

### 4.4.5 Pipeline Parameters

You can also pass in custom parameters. The `parameters` dictionary needs to be in the format of a two-layered dictionary where the first key-value pair is the component name and component parameters dictionary. The component parameters dictionary consists of a key value pair of parameter name and parameter values. An example will be shown below and component parameters can be found [here](#).

```
[7]: parameters = {
      'Simple Imputer': {
          'impute_strategy': 'mean'
      },
      'Logistic Regression Classifier': {
          'penalty': 'l2',
```

(continues on next page)

(continued from previous page)

```
        '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.

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

```
[8]:
```

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

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

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

Problem Type: Multiclass Classification
Model Family: Linear

Pipeline Steps
=====
1. Simple Imputer
   * impute_strategy : mean
   * fill_value : None
2. One Hot Encoder
   * top_n : 10
   * 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
```

## 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.

### 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
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
0          9.498540          0.371141
1          9.610488          0.371141
2          9.722436          0.371141
3          9.834384          0.371141
4          9.946332          0.371141
..          ...
95         20.133608          0.399560
96         20.245556          0.399560
97         20.357504          0.399560
98         20.469452          0.399560
99         20.581400          0.399560
```

[100 rows x 2 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

```
[8]: 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.

```
[9]: from evalml.model_understanding.graphs import graph_precision_recall_curve
      # get the predicted probabilities associated with the "true" label
      y_encoded = y.map({'malignant': 0, 'benign': 1})
      y_pred_proba = pipeline.predict_proba(X) ["benign"]
      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.

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

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

```
[11]:
```

	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]

```
[12]: 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

## 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%.

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

table = explain_prediction(pipeline=pipeline, input_features=X.iloc[3:4],
                           training_data=X, 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 area	567.70	-	-0.03
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, and error on that prediction. For a regression problem, we would see the predicted value instead of predicted probabilities.

```
[14]: 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, 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
```

(continues on next page)

(continued from previous page)

Cross Entropy: 0.0			
	Feature Name	Feature Value	Contribution to Prediction
↪ SHAP Value			
↪ =====			
↪ 0.10	worst perimeter	155.30	+
↪ 0.08	worst radius	23.14	+
↪ 0.08	worst concave points	0.17	+
↪ -0.00	worst fractal dimension	0.09	-
↪ -0.00	compactness error	0.04	-
↪ -0.00	worst symmetry	0.22	-
Best 2 of 2			
Predicted Probabilities: [benign: 0.0, malignant: 1.0]			
Predicted Value: malignant			
Target Value: malignant			
Cross Entropy: 0.0			
	Feature Name	Feature Value	Contribution to Prediction
↪ SHAP Value			
↪ =====			
↪ 0.10	worst perimeter	166.10	+
↪ 0.08	worst radius	25.45	+
↪ 0.08	worst concave points	0.22	+
↪ -0.00	compactness error	0.03	-
↪ -0.00	worst compactness	0.21	-
↪ -0.00	worst symmetry	0.21	-
Worst 1 of 2			
Predicted Probabilities: [benign: 0.552, malignant: 0.448]			
Predicted Value: benign			
Target Value: malignant			
Cross Entropy: 0.802			
	Feature Name	Feature Value	Contribution to Prediction
↪ SHAP Value			
↪ =====			

(continues on next page)

(continued from previous page)

↪ 0.04	smoothness error	0.00	+	↪
↪ 0.03	mean texture	21.58	+	↪
↪ 0.02	worst texture	30.25	+	↪
↪ -0.02	worst concave points	0.11	-	↪
↪ -0.03	worst radius	15.93	-	↪
↪ -0.03	mean concave points	0.02	-	↪
Worst 2 of 2				
Predicted Probabilities: [benign: 0.788, malignant: 0.212]				
Predicted Value: benign				
Target Value: malignant				
Cross Entropy: 1.55				
↪ SHAP Value	Feature Name	Feature Value	Contribution to Prediction	↪
↪ =====				
↪ 0.05	worst texture	33.37	+	↪
↪ 0.03	mean texture	22.47	+	↪
↪ 0.01	symmetry error	0.02	+	↪
↪ -0.04	worst concave points	0.09	-	↪
↪ -0.05	worst radius	14.49	-	↪
↪ -0.06	worst perimeter	92.04	-	↪

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)
```

(continues on next page)

(continued from previous page)

```
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.

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

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			
	=====		
	worst concave points	0.27	+
0.09	worst perimeter	184.60	+
0.09	worst radius	25.38	+
0.08	compactness error	0.05	-
-0.00	worst texture	17.33	-
-0.03	mean texture	10.38	-
-0.05			

2 of 3

	Feature Name	Feature Value	Contribution to Prediction
SHAP Value			
	=====		
	worst perimeter	152.20	+
0.11	worst radius	22.54	+
0.09	worst concave points	0.16	+
0.08	worst symmetry	0.24	-
-0.00	mean texture	14.34	-
-0.03	worst texture	16.67	-
-0.03			

3 of 3

(continues on next page)

(continued from previous page)

SHAP Value	Feature Name	Feature Value	Contribution to Prediction
=====			
0.20	worst concave points	0.22	++
0.11	mean concave points	0.09	+
0.08	mean concavity	0.23	+
-0.01	mean area	475.90	-
-0.03	worst radius	15.09	-
-0.05	worst perimeter	97.65	-

## 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 such as abnormal and missing data data checks.

### 4.6.1 Missing Data

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

for message in null_check.validate(X):
    print(message)
```

```
Column '2' is 80.0% or more null
```

## 4.6.2 Abnormal Data

EvalML provides two data checks to check for abnormal data: `OutliersDataCheck()` and `IDColumnsDataCheck()`.

### 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.

```
[2]: 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)

for message in id_col_check.validate(X):
    print(message.message)

Column 'id' is 90.0% or more likely to be an ID column
Column 'user_number' is 90.0% or more likely to be an ID column
```

## 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 Isolation Forests to notify you if a sample can be considered an outlier.

Below we generate a random dataset with some outliers.

```
[3]: data = np.random.randn(100, 100)
X = pd.DataFrame(data=data)

# generate some outliers in rows 3, 25, 55, and 72
X.iloc[3, :] = pd.Series(np.random.randn(100) * 10)
X.iloc[25, :] = pd.Series(np.random.randn(100) * 20)
X.iloc[55, :] = pd.Series(np.random.randn(100) * 100)
X.iloc[72, :] = pd.Series(np.random.randn(100) * 100)
```

We then utilize `OutliersDataCheck()` to rediscover these outliers.

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

outliers_check = OutliersDataCheck()

for message in outliers_check.validate(X):
    print(message.message)
```

```
Row '3' is likely to have outlier data
Row '25' is likely to have outlier data
Row '55' is likely to have outlier data
Row '72' is likely to have outlier data
```

## 4.6.4 Writing Your Own Data Check

If you would prefer to write your own data check, you can do so by extending the `DataCheck` class and implementing the `validate(self, X, y)` class method. Below, we've created a new `DataCheck`, `ZeroVarianceDataCheck`.

```
[5]: from evalml.data_checks import DataCheck
from evalml.data_checks.data_check_message import DataCheckError

class ZeroVarianceDataCheck(DataCheck):
    def validate(self, X, y):
        if not isinstance(X, pd.DataFrame):
            X = pd.DataFrame(X)
        warning_msg = "Column '{}' has zero variance"
        return [DataCheckError(warning_msg.format(column), self.name) for column in X.
            ↪columns if len(X[column].unique()) == 1]
```

## 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.

## 4.8 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.

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

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

Load credit card fraud dataset. The fraud dataset can be used for binary classification problems.

**Parameters**

- **n\_rows** (*int*) – number of rows from the dataset to return
- **verbose** (*bool*) – whether to print information about features and labels

**Returns** X, y

**Return type** pd.DataFrame, pd.Series

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

`evalml.demos.load_wine()`

Load wine dataset. Multiclass problem

**Returns** X, y

**Return type** pd.DataFrame, pd.Series

#### 5.1.3 `evalml.demos.load_breast_cancer`

`evalml.demos.load_breast_cancer()`

Load breast cancer dataset. Binary classification problem.

**Returns** X, y

**Return type** pd.DataFrame, pd.Series

### 5.1.4 evalml.demos.load\_diabetes

`evalml.demos.load_diabetes()`

Load diabetes dataset. Regression problem

**Returns** X, y

**Return type** pd.DataFrame, pd.Series

## 5.2 Preprocessing

Utilities to preprocess data before using evalml.

<code>drop_nan_target_rows</code>	Drops rows in X and y when row in the target y has a value of NaN.
<code>label_distribution</code>	Get the label distributions.
<code>load_data</code>	Load features and labels from file.
<code>number_of_features</code>	Get the number of features for specific dtypes.
<code>split_data</code>	Splits data into train and test sets.

### 5.2.1 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 values

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

**Return type** pd.DataFrame

### 5.2.2 evalml.preprocessing.label\_distribution

`evalml.preprocessing.label_distribution(labels)`

Get the label distributions.

**Parameters** **labels** (*pd.Series*) – Label values

**Returns** Label values and their frequency distribution as percentages.

**Return type** pd.Series

### 5.2.3 evalml.preprocessing.load\_data

`evalml.preprocessing.load_data(path, index, label, n_rows=None, drop=None, verbose=True, **kwargs)`

Load features and labels from file.

**Parameters**

- **path** (*str*) – Path to file or a http/ftp/s3 URL

- **index** (*str*) – Column for index
- **label** (*str*) – Column for labels
- **n\_rows** (*int*) – Number of rows to return
- **drop** (*list*) – List of columns to drop
- **verbose** (*bool*) – If True, prints information about features and labels

**Returns** features and labels

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

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

`evalml.preprocessing.number_of_features` (*dtypes*)

Get the number of features for specific dtypes.

**Parameters** **dtypes** (*pd.Series*) – 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*, *regression=False*, *test\_size=0.2*, *random\_state=None*)

Splits data into train and test sets.

**Parameters**

- **X** (*pd.DataFrame* or *np.array*) – data of shape [*n\_samples*, *n\_features*]
- **y** (*pd.Series*) – labels of length [*n\_samples*]
- **regression** (*bool*) – if true, do not use stratified split
- **test\_size** (*float*) – percent of train set to holdout for testing
- **random\_state** (*int*, *np.random.RandomState*) – seed for the random number generator

**Returns** features and labels each split into train and test sets

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

## 5.3 AutoML

### 5.3.1 AutoML Search Classes

---

*AutoMLSearch*

Automated Pipeline search.

---

## evalml automl AutoMLSearch

## evalml.automl.automl\_search.AutoMLSearch

```
class evalml.automl.AutoMLSearch(problem_type=None,                objective='auto',
                                max_pipelines=None,             max_time=None,        pa-
                                tience=None, tolerance=None,   data_split=None,    al-
                                lowed_pipelines=None,            allowed_model_families=None,
                                start_iteration_callback=None,   add_result_callback=None,
                                additional_objectives=None,      random_state=0,
                                n_jobs=-1, tuner_class=None,    verbose=True, opti-
                                mize_thresholds=False, _max_batches=None)
```

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__(problem_type=None,    objective='auto',    max_pipelines=None,
                       max_time=None, patience=None, tolerance=None, data_split=None,
                       allowed_pipelines=None,            allowed_model_families=None,
                       start_iteration_callback=None,      add_result_callback=None,
                       additional_objectives=None,         random_state=0,    n_jobs=-1,
                       tuner_class=None,    verbose=True,    optimize_thresholds=False,
                       _max_batches=None)
```

Automated pipeline search

## Parameters

- **problem\_type** (*str* or `ProblemTypes`) – Choice of ‘regression’, ‘binary’, or ‘multiclass’, depending on the desired problem type.
- **objective** (*str*, `ObjectiveBase`) – The objective to optimize for. When set to auto, chooses: `LogLossBinary` for binary classification problems, `LogLossMulticlass` for multiclass classification problems, and `R2` for regression problems.
- **max\_pipelines** (*int*) – Maximum number of pipelines to search. If `max_pipelines` and `max_time` is not set, then `max_pipelines` will default to `max_pipelines` 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\_split** (`sklearn.model_selection.BaseCrossValidator`) – data splitting method to use. Defaults to `StratifiedKFold`.
- **tuner\_class** – the tuner class to use. Defaults to `scikit-optimize` tuner
- **start\_iteration\_callback** (*callable*) – function called before each pipeline training iteration. Passed two parameters: `pipeline_class`, `parameters`.
- **add\_result\_callback** (*callable*) – function called after each pipeline training iteration. Passed two parameters: `results`, `trained_pipeline`.
- **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`
- **\_max\_batches** (*int*) – The maximum number of batches of pipelines to search. Parameters `max_time`, and `max_pipelines` have precedence over stopping the search.

### evalml automl.AutoMLSearch.add\_to\_rankings

`AutoMLSearch.add_to_rankings(pipeline, X, y)`

Fits and evaluates a given pipeline then adds the results to the automl rankings with the requirement that automl search has been run. Please use the same data as previous runs of automl search. If pipeline already exists in rankings this method will return *None*.

#### Parameters

- **pipeline** (`PipelineBase`) – pipeline to train and evaluate.
- **X** (`pd.DataFrame`) – the input training data of shape `[n_samples, n_features]`.
- **y** (`pd.Series`) – the target training labels of length `[n_samples]`.

### 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=4)`

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(X, y, data_checks='auto', feature_types=None, show_iteration_plot=True)`

Find the best pipeline for the data set.

**Parameters**

- **X** (*pd.DataFrame*) – the input training data of shape [n\_samples, n\_features]
- **y** (*pd.Series*) – the target training labels of length [n\_samples]
- **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.
- **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”.

**Returns** self

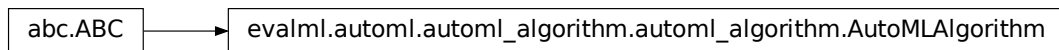
**Attributes**

<code>best_pipeline</code>	Returns an untrained instance of the best pipeline and parameters found during automl search.
<code>data_check_results</code>	
<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> .

**5.3.2 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_pipelines=None,
                                                    tuner_class=None,      ran-
                                                    dom_state=0)

```

Base class for the automl algorithms which power evalml.

### Methods

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

## evalml.automl.automl\_algorithm.AutoMLAlgorithm.\_\_init\_\_

```

AutoMLAlgorithm.__init__(allowed_pipelines=None, max_pipelines=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\_pipelines** (*int*) – The maximum number of pipelines 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

`AutoMLAlgorithm.next_batch()`

Get the next batch of pipelines to evaluate

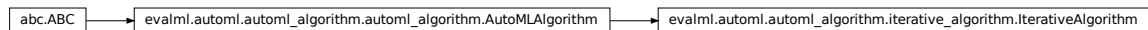
**Returns** a list of instances of PipelineBase subclasses, ready to be trained and evaluated.

**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.

### evalml.automl.automl\_algorithm.IterativeAlgorithm



```

class evalml.automl.automl_algorithm.IterativeAlgorithm(allowed_pipelines=None,
                                                         max_pipelines=None,
                                                         tuner_class=None,
                                                         random_state=0,
                                                         pipelines_per_batch=5,
                                                         n_jobs=-1,          number_features=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_pipelines=None, tuner_class=None, random_state=0, pipelines_per_batch=5, n_jobs=-1, number_features=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\_pipelines** (*int*) – The maximum number of pipelines 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.

### `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

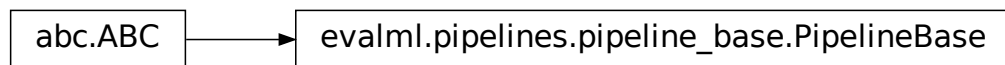
<code>batch_number</code>	Returns the number of batches which have been recommended so far.
<code>pipeline_number</code>	Returns the number of pipelines which have been recommended so far.

## 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.

#### evalml.pipelines.PipelineBase



**class** evalml.pipelines.**PipelineBase** (*parameters, random\_state=0*)  
Base class for all pipelines.

### 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>describe</code>	Outputs pipeline details including component parameters
<code>fit</code>	Build a model

Continued on next page

Table 12 – continued from previous page

<code>get_component</code>	Returns component by name
<code>graph</code>	Generate an image representing the pipeline graph
<code>graph_feature_importance</code>	Generate a bar graph of the pipeline’s feature importance
<code>load</code>	Loads pipeline at file path
<code>predict</code>	Make predictions using selected features.
<code>save</code>	Saves pipeline at file path
<code>score</code>	Evaluate model performance on current and additional objectives

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

`PipelineBase.__init__(parameters, random_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.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`

`PipelineBase.fit(X, y)`

Build a model

**Parameters**

- **X** (*pd.DataFrame* or *np.array*) – the input training data of shape [n\_samples, n\_features]
- **y** (*pd.Series*) – the target training labels 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** (*pd.DataFrame* or *np.array*) – 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.PipelineBase.save

PipelineBase.**save** (*file\_path*, *pickle\_protocol=4*)

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

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

Evaluate model performance on current and additional objectives

#### Parameters

- **X** (*pd.DataFrame* or *np.array*) – data of shape [n\_samples, n\_features]
- **y** (*pd.Series*) – true labels of length [n\_samples]
- **objectives** (*list*) – Non-empty list of objectives to score on

**Returns** ordered dictionary of objective scores

**Return type** *dict*

### evalml.pipelines.ClassificationPipeline



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

Pipeline subclass for all classification pipelines.

## 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>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.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.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** (*pd.DataFrame* or *np.array*) – the input training data of shape [`n_samples`, `n_features`]
- **y** (*pd.Series*) – 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` (*pd.DataFrame* or *np.array*) – 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` (*pd.DataFrame* or *np.array*) – data of shape [n\_samples, n\_features]**Returns** probability estimates**Return type** `pd.DataFrame`**evalml.pipelines.ClassificationPipeline.save**`ClassificationPipeline.save(file_path, pickle_protocol=4)`

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**

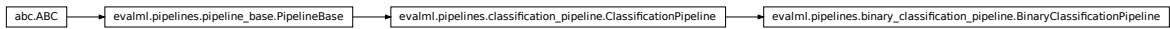
- `X` (*pd.DataFrame* or *np.array*) – data of shape [n\_samples, n\_features]
- `y` (*pd.Series*) – true labels of length [n\_samples]

- **objectives** (*list*) – list of objectives to score

**Returns** ordered dictionary of objective scores

**Return type** dict

## evalml.pipelines.BinaryClassificationPipeline



**class** evalml.pipelines.**BinaryClassificationPipeline** (*parameters*, *random\_state=0*)  
Pipeline subclass for all binary classification pipelines.

### 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>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.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** (*pd.DataFrame* or *np.array*) – the input training data of shape [n\_samples, n\_features]
- **y** (*pd.Series*) – 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** (*pd.DataFrame or np.array*) – 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.array*) – data of shape [n\_samples, n\_features]

**Returns** probability estimates

**Return type** pd.DataFrame

**evalml.pipelines.BinaryClassificationPipeline.save**

`BinaryClassificationPipeline.save` (*file\_path*, *pickle\_protocol=4*)

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** (*pd.DataFrame* or *np.array*) – data of shape [n\_samples, n\_features]
- **y** (*pd.Series*) – true labels of length [n\_samples]
- **objectives** (*list*) – list of objectives to score

**Returns** ordered dictionary of objective scores

**Return type** dict

**evalml.pipelines.MulticlassClassificationPipeline**

**class** evalml.pipelines.**MulticlassClassificationPipeline** (*parameters*, *random\_state=0*)

Pipeline subclass for all multiclass classification pipelines.

**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>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

Continued on next page

Table 15 – 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.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.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** (*pd.DataFrame* or *np.array*) – the input training data of shape [n\_samples, n\_features]
- **y** (*pd.Series*) – 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** (*pd.DataFrame* or *np.array*) – 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** (*pd.DataFrame* or *np.array*) – data of shape [n\_samples, n\_features]

**Returns** probability estimates

**Return type** `pd.DataFrame`

**evalml.pipelines.MulticlassClassificationPipeline.save**

`MulticlassClassificationPipeline.save` (*file\_path*, *pickle\_protocol=4*)

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**

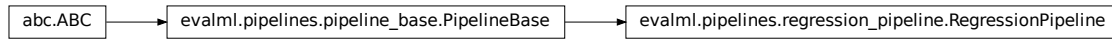
- **X** (*pd.DataFrame* or *np.array*) – data of shape [n\_samples, n\_features]
- **y** (*pd.Series*) – true labels of length [n\_samples]
- **objectives** (*list*) – list of objectives to score

**Returns** ordered dictionary of objective scores

**Return type** `dict`



## evalml.pipelines.RegressionPipeline



**class** evalml.pipelines.**RegressionPipeline** (*parameters*, *random\_state=0*)  
 Pipeline subclass for all regression pipelines.

### 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>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.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` (*pd.DataFrame* or *np.array*) – the input training data of shape `[n_samples, n_features]`
- `y` (*pd.Series*) – the target training labels 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` (*pd.DataFrame or np.array*) – 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.RegressionPipeline.save**

`RegressionPipeline.save` (*file\_path, pickle\_protocol=4*)

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` (*pd.DataFrame or np.array*) – data of shape `[n_samples, n_features]`
- `y` (*pd.Series*) – true labels of length `[n_samples]`

- **objectives** (*list*) – Non-empty list of objectives to score on

**Returns** ordered dictionary of objective scores

**Return type** dict

### 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

feature_importance	Return an attribute of instance, which is of type owner.
parameters	Returns parameter dictionary for this pipeline
threshold	

#### 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.

Continued on next page

Table 19 – continued from previous page

<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
<i>graph_feature_importance</i>	Generate a bar graph of the pipeline’s feature importance
<i>load</i>	Loads pipeline at file path
<i>predict</i>	Make predictions using selected features.
<i>predict_proba</i>	Make probability estimates for labels.
<i>save</i>	Saves pipeline at file path
<i>score</i>	Evaluate model performance on objectives

**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.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** (*pd.DataFrame* or *np.array*) – the input training data of shape `[n_samples, n_features]`
- **y** (*pd.Series*) – 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** (*pd.DataFrame* or *np.array*) – 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.array*) – data of shape [n\_samples, n\_features]

**Returns** probability estimates

**Return type** pd.DataFrame

### evalml.pipelines.BaselineBinaryPipeline.save

BaselineBinaryPipeline.**save** (*file\_path*, *pickle\_protocol=4*)

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** (*pd.DataFrame* or *np.array*) – data of shape [n\_samples, n\_features]
- **y** (*pd.Series*) – true labels of length [n\_samples]
- **objectives** (*list*) – list of objectives to score

**Returns** ordered dictionary of objective scores

**Return type** dict

## 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>feature_importance</code>	Return an attribute of instance, which is of type owner.
<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>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.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** (*pd.DataFrame* or *np.array*) – the input training data of shape [n\_samples, n\_features]
- **y** (*pd.Series*) – 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` (*pd.DataFrame or np.array*) – 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* (*pd.DataFrame* or *np.array*) – data of shape [n\_samples, n\_features]

**Returns** probability estimates

**Return type** *pd.DataFrame*

**evalml.pipelines.BaselineMulticlassPipeline.save**

BaselineMulticlassPipeline.**save**(*file\_path*, *pickle\_protocol=4*)

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** (*pd.DataFrame* or *np.array*) – data of shape [n\_samples, n\_features]
- **y** (*pd.Series*) – true labels of length [n\_samples]
- **objectives** (*list*) – list of objectives to score

**Returns** ordered dictionary of objective scores

**Return type** dict

**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>feature_importance</code>	Return an attribute of instance, which is of type owner.
<code>parameters</code>	Returns parameter dictionary for this pipeline
<code>threshold</code>	

### 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>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.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` (*pd.DataFrame* or *np.array*) – the input training data of shape `[n_samples, n_features]`
- `y` (*pd.Series*) – 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** (*pd.DataFrame or np.array*) – 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.array*) – data of shape `[n_samples, n_features]`

**Returns** probability estimates

**Return type** `pd.DataFrame`

**evalml.pipelines.ModeBaselineBinaryPipeline.save**

ModeBaselineBinaryPipeline.**save** (*file\_path*, *pickle\_protocol=4*)

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** (*pd.DataFrame* or *np.array*) – data of shape [n\_samples, n\_features]
- **y** (*pd.Series*) – true labels of length [n\_samples]
- **objectives** (*list*) – list of objectives to score

**Returns** ordered dictionary of objective scores

**Return type** dict

**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>feature_importance</code>	Return an attribute of instance, which is of type owner.
<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>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.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** (*pd.DataFrame* or *np.array*) – the input training data of shape `[n_samples, n_features]`
- **y** (*pd.Series*) – 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** (*pd.DataFrame or np.array*) – 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** (*pd.DataFrame or np.array*) – data of shape [n\_samples, n\_features]

**Returns** probability estimates

**Return type** `pd.DataFrame`

### `evalml.pipelines.ModeBaselineMulticlassPipeline.save`

`ModeBaselineMulticlassPipeline.save` (*file\_path, pickle\_protocol=4*)

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**

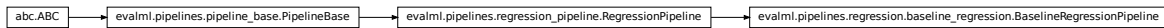
- **X** (*pd.DataFrame* or *np.array*) – data of shape [n\_samples, n\_features]
- **y** (*pd.Series*) – true labels of length [n\_samples]
- **objectives** (*list*) – list of objectives to score

**Returns** ordered dictionary of objective scores

**Return type** dict

**5.4.3 Regression Pipelines**

<i>BaselineRegressionPipeline</i>	Baseline Pipeline for regression problems.
<i>MeanBaselineRegressionPipeline</i>	Baseline Pipeline for 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**

<i>feature_importance</i>	Return an attribute of instance, which is of type owner.
<i>parameters</i>	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>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.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** (`pd.DataFrame` or `np.array`) – the input training data of shape `[n_samples, n_features]`
- **y** (`pd.Series`) – the target training labels 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` (*pd.DataFrame* or *np.array*) – 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.BaselineRegressionPipeline.save**`BaselineRegressionPipeline.save(file_path, pickle_protocol=4)`

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` (*pd.DataFrame* or *np.array*) – data of shape [n\_samples, n\_features]
- `y` (*pd.Series*) – true labels of length [n\_samples]
- `objectives` (*list*) – Non-empty list of objectives to score on

**Returns** ordered dictionary of objective scores**Return type** `dict`

## 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 an attribute of instance, which is of type owner.
<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>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.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** (*pd.DataFrame* or *np.array*) – the input training data of shape [n\_samples, n\_features]
- **y** (*pd.Series*) – the target training labels 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** (*pd.DataFrame or np.array*) – 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.MeanBaselineRegressionPipeline.save**

`MeanBaselineRegressionPipeline.save` (*file\_path*, *pickle\_protocol=4*)

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** (*pd.DataFrame* or *np.array*) – data of shape [n\_samples, n\_features]
- **y** (*pd.Series*) – true labels of length [n\_samples]
- **objectives** (*list*) – Non-empty list of objectives to score on

**Returns** ordered dictionary of objective scores

**Return type** dict

## 5.4.4 Pipeline Utils

---

*make\_pipeline*

Given input data, target data, an estimator class and the problem type,

---

**evalml.pipelines.utils.make\_pipeline**

`evalml.pipelines.utils.make_pipeline` (*X*, *y*, *estimator*, *problem\_type*)

**Given input data, target data, an estimator class and the problem type**, generates a pipeline class with a preprocessing chain which was recommended based on the inputs. The pipeline will be a subclass of the appropriate pipeline base class for the specified *problem\_type*.

**Parameters**

- **X** (*pd.DataFrame*) – the input data of shape [n\_samples, n\_features]
- **y** (*pd.Series*) – the target labels of length [n\_samples]
- **estimator** (*Estimator*) – estimator for pipeline
- **problem\_type** – problem type for pipeline to generate

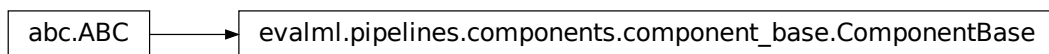
## 5.5 Components

### 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*) – 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** (*pd.DataFrame or np.array*) – 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.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=4*)

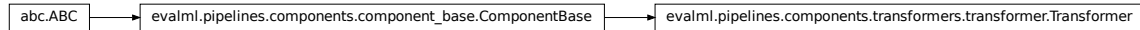
Saves component at file path

#### **Parameters**

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

**Returns** None

## evalml.pipelines.components.Transformer



```
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`) – 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** (*pd.DataFrame or np.array*) – 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.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*) – Labels to fit and transform

**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=4)`

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*) – Input Labels

**Returns** Transformed X

**Return type** `pd.DataFrame`

**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

Continued on next page

Table 35 – continued from previous page

<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.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*) – 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** (*pd.DataFrame or np.array*) – 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.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` (*pd.DataFrame*) – features

**Returns** estimated labels

**Return type** *pd.Series*

### `evalml.pipelines.components.Estimator.predict_proba`

`Estimator.predict_proba` (*X*)

Make probability estimates for labels.

**Parameters** `X` (*pd.DataFrame*) – features

**Returns** probability estimates

**Return type** *pd.DataFrame*

### `evalml.pipelines.components.Estimator.save`

`Estimator.save` (*file\_path*, *pickle\_protocol=4*)

Saves component at file path

**Parameters**

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

**Returns** None

## 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.

### `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]

## 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>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.

## 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*) – 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=4)`

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`

**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*) – 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=4)`

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`

### `evalml.pipelines.components.OneHotEncoder`



```

class evalml.pipelines.components.OneHotEncoder(top_n=10, categories=None,
                                                drop=None, handle_
                                                dle_unknown='ignore', han-
                                                dle_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, 'handle_missing': 'error',
```

### 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>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>	Returns names of transformed and added columns
<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, 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.
- **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*) – 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.clone**`OneHotEncoder.clone (random_state=0)`

Constructs a new component with the same parameters

**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 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` (*pd.DataFrame* or *np.array*) – 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.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*) – Labels to fit and transform

**Returns** Transformed X

**Return type** `pd.DataFrame`

**evalml.pipelines.components.OneHotEncoder.get\_feature\_names**

`OneHotEncoder.get_feature_names()`

Returns names of transformed and added columns

**Returns** list of feature names not including dropped features

**Return type** list

**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=4)`

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**

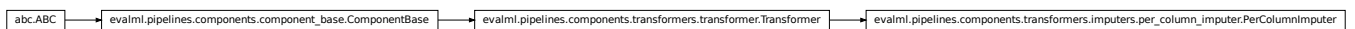
`OneHotEncoder.transform(X, y=None)`

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.

**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 data X
<code>fit_transform</code>	Fits imputer on data X then imputes missing values in 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

### 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*) – 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 data X

**Parameters**

- **X** (*pd.DataFrame*) – Data to fit
- **y** (*pd.Series, optional*) – Input Labels

**Returns** self

### `evalml.pipelines.components.PerColumnImputer.fit_transform`

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

Fits imputer on data X then imputes missing values in X

**Parameters**

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

**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=4)`

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)`

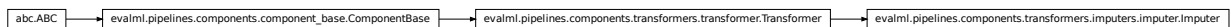
Transforms data X by imputing missing values

**Parameters**

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

**Returns** Transformed X

**Return type** `pd.DataFrame`

**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
<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

### `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 dtypes. 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*) – 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

**Parameters**

- **X** (*pd.DataFrame or np.array*) – 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.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*) – Labels to fit and transform

**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=4)`

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

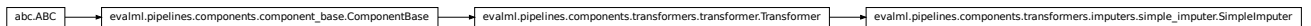
#### Parameters

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

**Returns** Transformed X

**Return type** `pd.DataFrame`

### 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
<code>fit_transform</code>	Fits on <i>X</i> and transforms <i>X</i> :param <i>X</i> : Data to fit and transform :type <i>X</i> : <code>pd.DataFrame</code> :param <i>y</i> : Labels to fit and transform :type <i>y</i> : <code>pd</code> .
<code>load</code>	Loads component at file path
<code>save</code>	Saves component at file path
<code>transform</code>	Transforms data <i>X</i> by imputing missing values

### `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*) – 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

#### Parameters

- **X** (*pd.DataFrame* or *np.array*) – 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.SimpleImputer.fit\_transform

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

Fits on X and transforms X :param X: Data to fit and transform :type X: pd.DataFrame :param y: Labels to fit and transform :type y: pd. DataFrame

**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=4)`

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

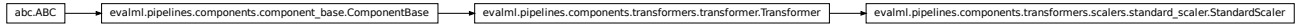
#### Parameters

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

**Returns** Transformed X

**Return type** `pd.DataFrame`

## 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*) – 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** (*pd.DataFrame or np.array*) – 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.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*) – Labels to fit and transform

**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=4)`

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)`

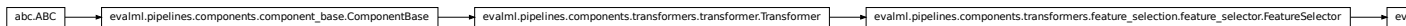
Transforms data X

**Parameters**

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

**Returns** Transformed X

**Return type** `pd.DataFrame`

**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',

**default\_parameters** = {'max\_depth': None, 'n\_estimators': 10, 'n\_jobs': -1, 'number\_

**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*) – 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** (`pd.DataFrame` or `np.array`) – 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.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`) – Labels to fit and transform

**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=4)`

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*) – Input Labels

**Returns** Transformed X

**Return type** *pd.DataFrame*

**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',

**default\_parameters** = {'max\_depth': None, 'n\_estimators': 10, 'n\_jobs': -1, 'number\_

**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

Continued on next page



Table 55 – continued from previous page

<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*) – 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** (*pd.DataFrame* or *np.array*) – 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.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`) – Labels to fit and transform

**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=4)`

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

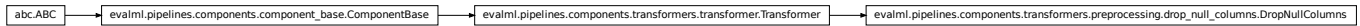
**Parameters**

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

**Returns** Transformed X

**Return type** `pd.DataFrame`

## 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*) – 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** (*pd.DataFrame or np.array*) – 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.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*) – Labels to fit and transform

**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=4)`

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. :param X: Data to transform :type X: pd.DataFrame :param y: Targets :type y: pd.Series, optional

**Returns** Transformed X

**Return type** pd.DataFrame

**evalml.pipelines.components.DateTimeFeaturizer**

**class** evalml.pipelines.components.**DateTimeFeaturizer** (*features\_to\_extract=None, random\_state=0, \*\*kwargs*)

Transformer that can automatically featurize DateTime columns.

**name** = 'DateTime Featurization Component'

**model\_family** = 'none'

**hyperparameter\_ranges** = {}

**default\_parameters** = {'features\_to\_extract': ['year', 'month', 'day\_of\_week', 'hour']}

**Instance attributes**

<code>needs_fitting</code>	
<code>parameters</code>	Returns the parameters which were used to initialize the component

**Methods:**

<code>__init__</code>	Extracts features from DateTime columns
<code>clone</code>	Constructs a new component with the same parameters

Continued on next page

Table 59 – continued from previous page

<i>describe</i>	Describe a component and its parameters
<i>fit</i>	Fits component to data
<i>fit_transform</i>	Fits on X and transforms X
<i>load</i>	Loads component at file path
<i>save</i>	Saves component at file path
<i>transform</i>	Transforms data X by creating new features using existing DateTime columns, and then dropping those DateTime columns

### `evalml.pipelines.components.DateTimeFeaturizer.__init__`

`DateTimeFeaturizer.__init__` (*features\_to\_extract=None, 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”.
- **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*) – 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** (*pd.DataFrame* or *np.array*) – 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.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*) – Labels to fit and transform

**Returns** Transformed X

**Return type** *pd.DataFrame*

### 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=4)`

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*) – Input Labels

**Returns** Transformed X

**Return type** *pd.DataFrame*

## evalml.pipelines.components.TextFeaturizer



```

class evalml.pipelines.components.TextFeaturizer(text_columns=None,          ran-
                                                  dom_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*) – 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.array*) – 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*) – Labels to fit and transform

**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=4*)

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*) – Input Labels

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

## 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>LogisticRegressionClassifier</i>	Logistic Regression Classifier.
<i>XGBoostClassifier</i>	XGBoost Classifier.
<i>BaselineClassifier</i>	Classifier that predicts using the specified strategy.

**evalml.pipelines.components.CatBoostClassifier**

```

class evalml.pipelines.components.CatBoostClassifier(n_estimators=10, eta=0.03,
                                                    max_depth=6,          boot-
                                                    strap_type=None,        ran-
                                                    dom_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: 'multiclass'>]
```

```
hyperparameter_ranges = {'eta': Real(low=1e-06, high=1, prior='uniform', transform='log')}
```

```
default_parameters = {'bootstrap_type': None, 'eta': 0.03, 'max_depth': 6, 'n_estimators': 10}
```

**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, boot-
                           strap_type=None, 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*) – 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** (*pd.DataFrame or np.array*) – 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.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` (*pd.DataFrame*) – features

**Returns** estimated labels

**Return type** `pd.Series`

### `evalml.pipelines.components.CatBoostClassifier.predict_proba`

`CatBoostClassifier.predict_proba(X)`

Make probability estimates for labels.

**Parameters** `X` (`pd.DataFrame`) – features

**Returns** probability estimates

**Return type** `pd.DataFrame`

### `evalml.pipelines.components.CatBoostClassifier.save`

`CatBoostClassifier.save(file_path, pickle_protocol=4)`

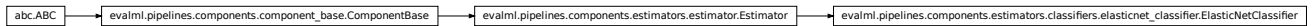
Saves component at file path

**Parameters**

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

**Returns** `None`

### `evalml.pipelines.components.ElasticNetClassifier`



```

class evalml.pipelines.components.ElasticNetClassifier(alpha=0.5, l1_ratio=0.5,
                                                       n_jobs=-1, max_iter=1000,
                                                       random_state=0,
                                                       **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='identity',`

`default_parameters = {'alpha': 0.5, 'l1_ratio': 0.5, 'max_iter': 1000, 'n_jobs': -1,`

#### Instance attributes

<code>feature_importance</code>	Return an attribute of instance, which is of type owner.
<code>needs_fitting</code>	

Continued on next page

Table 65 – continued from previous page

<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, \*\*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*) – 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** (*pd.DataFrame* or *np.array*) – 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.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** (*pd.DataFrame*) – features**Returns** estimated labels**Return type** pd.Series**evalml.pipelines.components.ElasticNetClassifier.predict\_proba**ElasticNetClassifier.**predict\_proba**(*X*)

Make probability estimates for labels.

**Parameters** **X** (*pd.DataFrame*) – features**Returns** probability estimates**Return type** pd.DataFrame**evalml.pipelines.components.ElasticNetClassifier.save**ElasticNetClassifier.**save**(*file\_path*, *pickle\_protocol=4*)

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.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:
hyperparameter_ranges = {'max_depth': Integer(low=4, high=10, prior='uniform', transf
default_parameters = {'max_depth': 6, 'max_features': 'auto', 'min_samples_split':

```

## 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.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*) – 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** (*pd.DataFrame* or *np.array*) – 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.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` (*pd.DataFrame*) – features

**Returns** estimated labels

**Return type** `pd.Series`

### `evalml.pipelines.components.ExtraTreesClassifier.predict_proba`

`ExtraTreesClassifier.predict_proba(X)`

Make probability estimates for labels.

**Parameters** `X` (`pd.DataFrame`) – features

**Returns** probability estimates

**Return type** `pd.DataFrame`

### `evalml.pipelines.components.ExtraTreesClassifier.save`

`ExtraTreesClassifier.save(file_path, pickle_protocol=4)`

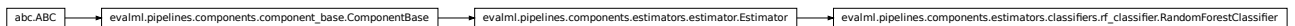
Saves component at file path

**Parameters**

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

**Returns** `None`

### `evalml.pipelines.components.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', transf
default_parameters = {'max_depth': 6, 'n_estimators': 100, 'n_jobs': -1}
  
```

#### Instance attributes

<code>feature_importance</code>	Return an attribute of instance, which is of type owner.
<code>needs_fitting</code>	

Continued on next page

Table 69 – continued from previous page

<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*) – 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** (*pd.DataFrame* or *np.array*) – 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.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** (*pd.DataFrame*) – features

**Returns** estimated labels

**Return type** *pd.Series*

**evalml.pipelines.components.RandomForestClassifier.predict\_proba**

`RandomForestClassifier.predict_proba(X)`

Make probability estimates for labels.

**Parameters** **X** (*pd.DataFrame*) – features

**Returns** probability estimates

**Return type** *pd.DataFrame*

**evalml.pipelines.components.RandomForestClassifier.save**

`RandomForestClassifier.save(file_path, pickle_protocol=4)`

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.LogisticRegressionClassifier



```

class evalml.pipelines.components.LogisticRegressionClassifier(penalty='l2',
                                                             C=1.0,
                                                             n_jobs=-1, 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, '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, 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*) – 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` (*pd.DataFrame or np.array*) – 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.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` (*pd.DataFrame*) – features

**Returns** estimated labels

**Return type** `pd.Series`

### `evalml.pipelines.components.LogisticRegressionClassifier.predict_proba`

`LogisticRegressionClassifier.predict_proba(X)`

Make probability estimates for labels.

**Parameters** `X` (`pd.DataFrame`) – features

**Returns** probability estimates

**Return type** `pd.DataFrame`

### `evalml.pipelines.components.LogisticRegressionClassifier.save`

`LogisticRegressionClassifier.save(file_path, pickle_protocol=4)`

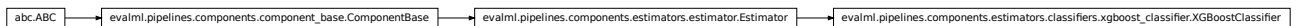
Saves component at file path

**Parameters**

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

**Returns** `None`

### `evalml.pipelines.components.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='i
    default_parameters = {'eta':  0.1, 'max_depth':  6, 'min_child_weight':  1, 'n_estimat
  
```

#### Instance attributes

SEED_MAX	
SEED_MIN	
feature_importance	Return an attribute of instance, which is of type owner.

Continued on next page

Table 73 – continued from previous page

<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*) – 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** (*pd.DataFrame* or *np.array*) – 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.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** (*pd.DataFrame*) – features**Returns** estimated labels**Return type** *pd.Series***evalml.pipelines.components.XGBoostClassifier.predict\_proba**XGBoostClassifier.**predict\_proba** (*X*)

Make probability estimates for labels.

**Parameters** **X** (*pd.DataFrame*) – features**Returns** probability estimates**Return type** *pd.DataFrame***evalml.pipelines.components.XGBoostClassifier.save**XGBoostClassifier.**save** (*file\_path*, *pickle\_protocol=4*)

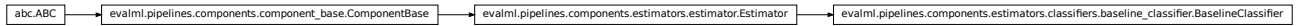
Saves component at file path

**Parameters**

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

**Returns** None

## evalml.pipelines.components.BaselineClassifier



```

class evalml.pipelines.components.BaselineClassifier(strategy='mode',          ran-
                                                    dom_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>	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>	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*) – 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** (*pd.DataFrame* or *np.array*) – 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.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` (`pd.DataFrame`) – features

**Returns** estimated labels

**Return type** `pd.Series`

### evalml.pipelines.components.BaselineClassifier.predict\_proba

`BaselineClassifier.predict_proba(X)`

Make probability estimates for labels.

**Parameters** `X` (`pd.DataFrame`) – features

**Returns** probability estimates

**Return type** `pd.DataFrame`

### evalml.pipelines.components.BaselineClassifier.save

`BaselineClassifier.save(file_path, pickle_protocol=4)`

Saves component at file path

**Parameters**

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

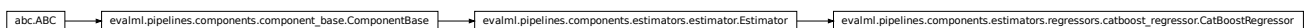
**Returns** `None`

## Regressors

Regressors are components that output a predicted target value.

<code>CatBoostRegressor</code>	CatBoost Regressor, a regressor that uses gradient-boosting on decision trees.
<code>ElasticNetRegressor</code>	Elastic Net Regressor.
<code>LinearRegressor</code>	Linear Regressor.
<code>ExtraTreesRegressor</code>	Extra Trees Regressor.
<code>RandomForestRegressor</code>	Random Forest Regressor.
<code>XGBoostRegressor</code>	XGBoost Regressor.
<code>BaselineRegressor</code>	Regressor that predicts using the specified strategy.

### evalml.pipelines.components.CatBoostRegressor



```
class evalml.pipelines.components.CatBoostRegressor (n_estimators=10, eta=0.03,  
                                                    max_depth=6, boot-  
                                                    strap_type=None, ran-  
                                                    dom_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'>]
```

```
hyperparameter_ranges = {'eta': Real(low=1e-06, high=1, prior='uniform', transform='i
```

```
default_parameters = {'bootstrap_type': None, 'eta': 0.03, 'max_depth': 6, 'n_estim
```

### 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>	Build a model
<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, boot-  
                             strap_type=None, 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*) – 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*)

Build a model

#### **Parameters**

- **X** (*pd.DataFrame or np.array*) – the input training data of shape [n\_samples, n\_features]
- **y** (*pd.Series*) – the target training labels 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** (*pd.DataFrame*) – features

**Returns** estimated labels

**Return type** pd.Series

**evalml.pipelines.components.CatBoostRegressor.predict\_proba**`CatBoostRegressor.predict_proba(X)`

Make probability estimates for labels.

**Parameters** *X* (`pd.DataFrame`) – features**Returns** probability estimates**Return type** `pd.DataFrame`**evalml.pipelines.components.CatBoostRegressor.save**`CatBoostRegressor.save(file_path, pickle_protocol=4)`

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.ElasticNetRegressor**

```

abc.ABC → evalml.pipelines.components.component_base.ComponentBase → evalml.pipelines.components.estimators.estimator.Estimator → evalml.pipelines.components.estimators.regressors.elasticnet_regressor.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'>`]**hyperparameter\_ranges** = {'alpha': `Real(low=0, high=1, prior='uniform', transform='identity')`,**default\_parameters** = {'alpha': 0.5, 'l1\_ratio': 0.5, 'max\_iter': 1000, 'normalize':**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*) – 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** (*pd.DataFrame* or *np.array*) – 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.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** (*pd.DataFrame*) – features

**Returns** estimated labels

**Return type** pd.Series

### evalml.pipelines.components.ElasticNetRegressor.predict\_proba

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

Make probability estimates for labels.

**Parameters** **X** (*pd.DataFrame*) – features

**Returns** probability estimates

**Return type** pd.DataFrame

### evalml.pipelines.components.ElasticNetRegressor.save

ElasticNetRegressor.**save** (*file\_path*, *pickle\_protocol=4*)

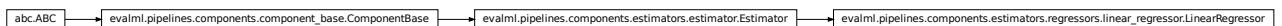
Saves component at file path

**Parameters**

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

**Returns** None

### evalml.pipelines.components.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'>]
    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*) – 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** (*pd.DataFrame or np.array*) – 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.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** (*pd.DataFrame*) – features**Returns** estimated labels**Return type** pd.Series**evalml.pipelines.components.LinearRegressor.predict\_proba**`LinearRegressor.predict_proba` (*X*)

Make probability estimates for labels.

**Parameters** **X** (*pd.DataFrame*) – features**Returns** probability estimates

**Return type** `pd.DataFrame`

### `evalml.pipelines.components.LinearRegressor.save`

`LinearRegressor.save(file_path, pickle_protocol=4)`

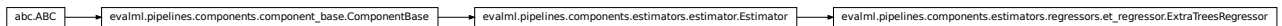
Saves component at file path

#### Parameters

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

**Returns** `None`

### `evalml.pipelines.components.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'>`]

**hyperparameter\_ranges** = {'max\_depth': `Integer(low=4, high=10, prior='uniform', transfo`

**default\_parameters** = {'max\_depth': 6, 'max\_features': 'auto', 'min\_samples\_split':

#### 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.
-----------------------	------------------

Continued on next page

Table 85 – continued from previous page

<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>predict</i>	Make predictions using selected features.
<i>predict_proba</i>	Make probability estimates for labels.
<i>save</i>	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*) – 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** (*pd.DataFrame* or *np.array*) – 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.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** (*pd.DataFrame*) – features**Returns** estimated labels**Return type** pd.Series**evalml.pipelines.components.ExtraTreesRegressor.predict\_proba**ExtraTreesRegressor.**predict\_proba** (*X*)

Make probability estimates for labels.

**Parameters** **X** (*pd.DataFrame*) – features**Returns** probability estimates**Return type** pd.DataFrame**evalml.pipelines.components.ExtraTreesRegressor.save**ExtraTreesRegressor.**save** (*file\_path*, *pickle\_protocol=4*)

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.RandomForestRegressor**

```
abc.ABC → evalml.pipelines.components.component_base.ComponentBase → evalml.pipelines.components.estimators.estimator.Estimator → evalml.pipelines.components.estimators.regressors.rf_regressor.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'>]
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>	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.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*) – 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** (*pd.DataFrame or np.array*) – 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.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** (*pd.DataFrame*) – features

**Returns** estimated labels

**Return type** pd.Series

### **evalml.pipelines.components.RandomForestRegressor.predict\_proba**

`RandomForestRegressor.predict_proba(X)`

Make probability estimates for labels.

**Parameters** **X** (*pd.DataFrame*) – features

**Returns** probability estimates

**Return type** pd.DataFrame



**evalml.pipelines.components.RandomForestRegressor.save**

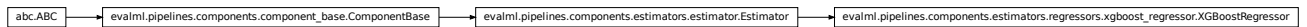
`RandomForestRegressor.save(file_path, pickle_protocol=4)`

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.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'>]
```

```
hyperparameter_ranges = {'eta': Real(low=1e-06, high=1, prior='uniform', transform='i
```

```
default_parameters = {'eta': 0.1, 'max_depth': 6, 'min_child_weight': 1, 'n_estimat
```

**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

Continued on next page

Table 89 – continued from previous page

<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*) – 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** (*pd.DataFrame* or *np.array*) – 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.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** (*pd.DataFrame*) – features

**Returns** estimated labels

**Return type** pd.Series

**evalml.pipelines.components.XGBoostRegressor.predict\_proba**

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

Make probability estimates for labels.

**Parameters** **X** (*pd.DataFrame*) – features

**Returns** probability estimates

**Return type** pd.DataFrame

**evalml.pipelines.components.XGBoostRegressor.save**

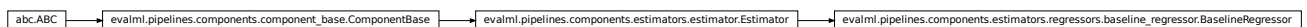
XGBoostRegressor.**save** (*file\_path*, *pickle\_protocol=4*)

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.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'>]
hyperparameter_ranges = {}
default_parameters = {'strategy': 'mean'}
```

### 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>	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*) – 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** (*pd.DataFrame or np.array*) – 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.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** (*pd.DataFrame*) – features**Returns** estimated labels**Return type** pd.Series**evalml.pipelines.components.BaselineRegressor.predict\_proba**`BaselineRegressor.predict_proba` (*X*)

Make probability estimates for labels.

**Parameters** **X** (*pd.DataFrame*) – features**Returns** probability estimates

**Return type** `pd.DataFrame`

### `evalml.pipelines.components.BaselineRegressor.save`

`BaselineRegressor.save(file_path, pickle_protocol=4)`

Saves component at file path

#### Parameters

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

**Returns** `None`

## 5.6 Model Understanding

### 5.6.1 Graph Utils

<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>graph_precision_recall_curve</code>	Generate and display a precision-recall plot.
<code>roc_curve</code>	Given labels and classifier predicted probabilities, compute and return the data representing a Receiver Operating Characteristic (ROC) curve.
<code>graph_roc_curve</code>	Generate and display a Receiver Operating Characteristic (ROC) plot.
<code>graph_confusion_matrix</code>	Generate and display a confusion matrix plot.
<code>calculate_permutation_importance</code>	Calculates permutation importance for features.
<code>graph_permutation_importance</code>	Generate a bar graph of the pipeline's permutation importance.
<code>binary_objective_vs_threshold</code>	Computes objective score as a function of potential binary classification
<code>graph_binary_objective_vs_threshold</code>	Generates a plot graphing objective score vs.

### `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** (*pd.Series* or *np.array*) – true binary labels.
- **y\_pred** (*pd.Series* or *np.array*) – predictions from a binary classifier.
- **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** confusion matrix

**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** (*pd.DataFrame* or *np.array*) – 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.

**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** (*pd.Series* or *np.array*) – true binary labels.
- **y\_pred\_proba** (*pd.Series* or *np.array*) – 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.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** (*pd.Series* or *np.array*) – true binary labels.

- **y\_pred\_proba** (*pd.Series* or *np.array*) – 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.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.

#### **Parameters**

- **y\_true** (*pd.Series* or *np.array*) – true labels.
- **y\_pred\_proba** (*pd.Series* or *np.array*) – predictions from a classifier, before thresholding has been applied. Note that 1 dimensional input is expected.

#### **Returns**

Dictionary containing 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** *dict*

### **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.

#### **Parameters**

- **y\_true** (*pd.Series* or *np.array*) – true labels.
- **y\_pred\_proba** (*pd.Series* or *np.array*) – 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** (*pd.Series* or *np.array*) – true binary labels.
- **y\_pred** (*pd.Series* or *np.array*) – predictions from a binary classifier.
- **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'.
- **title\_addition** (*str* or *None*) – if not None, append to plot title. Default None.

**Returns** *plotly.Figure* representing the confusion matrix plot generated

## 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** (*pd.DataFrame*) – the input data used to score and compute permutation importance
- **y** (*pd.Series*) – the target labels
- **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.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** (*pd.DataFrame*) – The input data used to score and compute permutation importance
- **y** (*pd.Series*) – The target labels
- **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.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** (*pd.DataFrame*) – the input data used to compute objective score
- **y** (*pd.Series*) – the target labels
- **objective** (*ObjectiveBase obj, str*) – objective used to score
- **steps** (*int*) – Number of intervals to divide and calculate objective score at

**Returns** *DataFrame* with thresholds and the corresponding objective score calculated at each threshold

**Return type** *pd.DataFrame*

### **evalml.model\_understanding.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** (*pd.DataFrame*) – the input data used to score and compute scores
- **y** (*pd.Series*) – the target labels
- **objective** (*ObjectiveBase obj, str*) – objective used to score, shown on the y-axis of the graph
- **steps** (*int*) – Number of intervals to divide and calculate objective score at

**Returns** *plotly.Figure* representing the objective score vs. threshold graph generated

## 5.6.2 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*,  
*input\_features*,  
*top\_k=3*,  
*training\_data=None*,  
*include\_shap\_values=False*)

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** (`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.

**Returns** Table

**Return type** str

### 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)
```

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** (`pd.DataFrame`) – Dataframe of input data to evaluate the pipeline on.
- **training\_data** (`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.

#### Returns

**str** - A report with the pipeline name and parameters and a table for each row of input\_features.

The table will have the following columns: Feature Name, Contribution to Prediction, SHAP Value (optional), and each row of the table will be a feature.

### evalml.model\_understanding.prediction\_explanations.explain\_predictions\_best\_worst

```
evalml.model_understanding.prediction_explanations.explain_predictions_best_worst(pipeline,  
                                                                           in-  
                                                                           put_features,  
                                                                           y_true,  
                                                                           num_to_expl  
                                                                           top_k_featur  
                                                                           in-  
                                                                           clude_shap_  
                                                                           met-  
                                                                           ric=None)
```

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** (*pd.DataFrame*) – Dataframe of input data to evaluate the pipeline on.
- **y\_true** (*pd.Series*) – True labels for the input data.
- **num\_to\_explain** (*int*) – How many of the best, worst, random data points to explain.
- **top\_k\_features** (*int*) – How many of the highest/lowest contributing feature to include in the table for each data point.
- **include\_shap\_values** (*bool*) – Whether SHAP values should be included in the table. Default is False.
- **metric** (*callable*) – The metric used to identify the best and worst points in the dataset. Function must accept the true labels and predicted value or probabilities as the only arguments and lower values must be better. By default, this will be the absolute error for regression problems and cross entropy loss for classification problems.

#### Returns

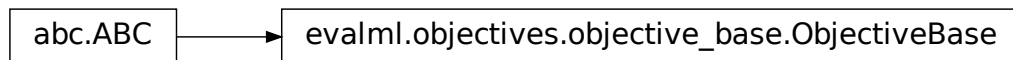
**str** - A report with the pipeline name and parameters. For each of the best/worst rows of **input\_features**, the predicted values, true labels, and metric value will be listed along with a table. The table will have the following columns: Feature Name, Contribution to Prediction, SHAP Value (optional), and each row of the table will correspond to a feature.

## 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

<code>calculate_percent_difference</code>	Calculate the percent difference between scores.
<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.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.objective_function`

**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.array*): 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.array*) – extra data of shape [*n\_samples*, *n\_features*] necessary to calculate score

**Returns** score

**evalml.objectives.ObjectiveBase.validate\_inputs**

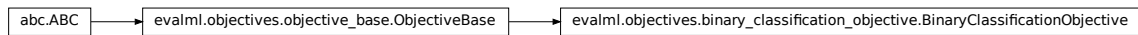
`ObjectiveBase.validate_inputs(y_true, y_predicted)`

Validates the input based on a few simple checks.

**Parameters**

- **y\_predicted** (*pd.Series*) – predicted values of length [n\_samples]
- **y\_true** (*pd.Series*) – actual class labels of length [n\_samples]

**Returns** None

**evalml.objectives.BinaryClassificationObjective****class evalml.objectives.BinaryClassificationObjective**

Base class for all binary classification objectives.

**problem\_type** (*ProblemTypes*): Type of problem this objective is. Set to *ProblemTypes.BINARY*.

**can\_optimize\_threshold** (*bool*): Determines if threshold used by objective can be optimized or not.

**Methods**

<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>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.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** (*list*) – The classifier’s predicted probabilities
- **threshold** (*float, optional*) – Threshold used to make a prediction. Defaults to 0.5.
- **X** (*pd.DataFrame, optional*) – Any extra columns that are needed from training data.

**Returns** predictions

**evalml.objectives.BinaryClassificationObjective.objective\_function**

**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.array*): extra data of shape `[n_samples, n_features]` necessary to calculate score

**Returns** numerical value used to calculate score

**evalml.objectives.BinaryClassificationObjective.optimize\_threshold**

`BinaryClassificationObjective.optimize_threshold(ypred_proba, y_true, X=None)`

Learn a binary classification threshold which optimizes the current objective.

**Parameters**

- **ypred\_proba** (*list*) – The classifier’s predicted probabilities
- **y\_true** (*list*) – The ground truth for the predictions.
- **X** (*pd.DataFrame, optional*) – Any extra columns that are needed from training data.

**Returns** Optimal threshold for this objective



**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.array`) – extra data of shape [n\_samples, n\_features] necessary to calculate score

**Returns** score

**evalml.objectives.BinaryClassificationObjective.validate\_inputs**

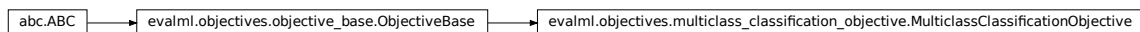
`BinaryClassificationObjective.validate_inputs(y_true, y_predicted)`

Validates the input based on a few simple checks.

**Parameters**

- **y\_predicted** (`pd.Series`) – predicted values of length [n\_samples]
- **y\_true** (`pd.Series`) – actual class labels of length [n\_samples]

**Returns** None

**evalml.objectives.MulticlassClassificationObjective****class evalml.objectives.MulticlassClassificationObjective**

Base class for all multiclass classification objectives.

**problem\_type** (`ProblemTypes`): Type of problem this objective is. Set to `ProblemTypes.MULTICLASS`.

**Methods**

<code>calculate_percent_difference</code>	Calculate the percent difference between scores.
<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.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.objective_function`

**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.array*): 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.array*) – extra data of shape [*n\_samples*, *n\_features*] necessary to calculate score

**Returns** score

### `evalml.objectives.MulticlassClassificationObjective.validate_inputs`

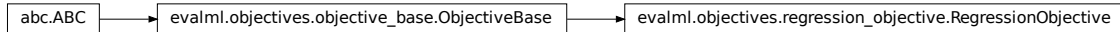
`MulticlassClassificationObjective.validate_inputs` (*y\_true*, *y\_predicted*)

Validates the input based on a few simple checks.

**Parameters**

- **y\_predicted** (*pd.Series*) – predicted values of length [n\_samples]
- **y\_true** (*pd.Series*) – actual class labels of length [n\_samples]

**Returns** None

**evalml.objectives.RegressionObjective****class evalml.objectives.RegressionObjective**

Base class for all regression objectives.

**problem\_type** (ProblemTypes): Type of problem this objective is. Set to ProblemTypes.REGRESSION.

**Methods**

<i>calculate_percent_difference</i>	Calculate the percent difference between scores.
<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.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.objective\_function**

**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.array`): 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.array`) – 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**

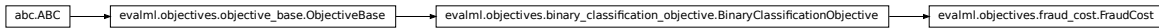
- **y\_predicted** (`pd.Series`) – predicted values of length [*n\_samples*]
- **y\_true** (`pd.Series`) – actual class labels of length [*n\_samples*]

**Returns** None

## 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>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(y_pred_proba, threshold=0.0, X=None)`

Determine if a transaction is fraud given predicted probabilities, threshold, and dataframe with transaction amount.

**Parameters**

- **y\_pred\_proba** (*pd.Series*) – Predicted probabilities
- **X** (*pd.DataFrame*) – Dataframe containing transaction amount
- **threshold** (*float*) – Dollar threshold to determine if transaction is fraud

**Returns** Series of predicted fraud labels using X and threshold

**Return type** pd.Series

**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** (*pd.Series*) – predicted fraud labels y\_true (*pd.Series*): true fraud labels
- **X** (*pd.DataFrame*) – dataframe with transaction amounts

**Returns** amount lost to fraud per transaction

**Return type** float

**evalml.objectives.FraudCost.optimize\_threshold**

`FraudCost.optimize_threshold(y_pred_proba, y_true, X=None)`

Learn a binary classification threshold which optimizes the current objective.

**Parameters**

- **y\_pred\_proba** (*list*) – The classifier’s predicted probabilities
- **y\_true** (*list*) – The ground truth for the predictions.
- **X** (*pd.DataFrame, optional*) – Any extra columns that are needed from training data.

**Returns** Optimal threshold for this objective

### 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.array*) – 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)`

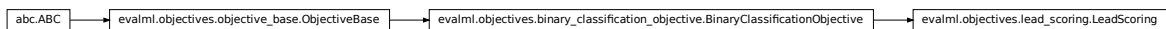
Validates the input based on a few simple checks.

#### Parameters

- **y\_predicted** (*pd.Series*) – predicted values of length [n\_samples]
- **y\_true** (*pd.Series*) – actual class labels of length [n\_samples]

**Returns** None

### 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>objective_function</code>	Calculate the profit per lead.
<code>optimize_threshold</code>	Learn a binary classification threshold which optimizes the current objective.

Continued on next page

Table 101 – 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.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** (*list*) – The classifier’s predicted probabilities
- **threshold** (*float, optional*) – Threshold used to make a prediction. Defaults to 0.5.
- **X** (*pd.DataFrame, optional*) – Any extra columns that are needed from training data.

**Returns** predictions

### `evalml.objectives.LeadScoring.objective_function`

`LeadScoring.objective_function(y_true, y_predicted, X=None)`  
Calculate the profit per lead.



**Parameters**

- **y\_predicted** (*pd.Series*) – predicted labels
- **y\_true** (*pd.Series*) – true labels
- **X** (*pd.DataFrame*) – None, not used.

**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** (*list*) – The classifier’s predicted probabilities
- **y\_true** (*list*) – The ground truth for the predictions.
- **X** (*pd.DataFrame, optional*) – Any extra columns that are needed from training data.

**Returns** Optimal threshold for this objective

**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.array*) – 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)`

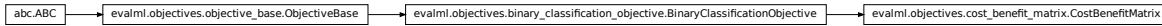
Validates the input based on a few simple checks.

**Parameters**

- **y\_predicted** (*pd.Series*) – predicted values of length [n\_samples]
- **y\_true** (*pd.Series*) – actual class labels of length [n\_samples]

**Returns** None

## 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>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** (*list*) – The classifier’s predicted probabilities
- **threshold** (*float, optional*) – Threshold used to make a prediction. Defaults to 0.5.
- **X** (*pd.DataFrame, optional*) – Any extra columns that are needed from training data.

**Returns** predictions

**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*) – Predicted labels
- **y\_true** (*pd.Series*) – True labels
- **X** (*pd.DataFrame*) – Ignored.

**Returns** score

**Return type** float

**evalml.objectives.CostBenefitMatrix.optimize\_threshold**

`CostBenefitMatrix.optimize_threshold(ypred_proba, y_true, X=None)`

Learn a binary classification threshold which optimizes the current objective.

**Parameters**

- **ypred\_proba** (*list*) – The classifier’s predicted probabilities
- **y\_true** (*list*) – The ground truth for the predictions.
- **X** (*pd.DataFrame, optional*) – Any extra columns that are needed from training data.

**Returns** Optimal threshold for this objective

**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.array*) – 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** (*pd.Series*) – predicted values of length [n\_samples]
- **y\_true** (*pd.Series*) – actual class labels of length [n\_samples]

**Returns** None

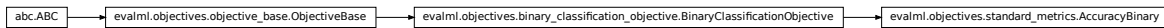
## 5.7.3 Classification Objectives

<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.

Continued on next page

Table 103 – continued from previous page

<i>MCCMulticlass</i>	Matthews correlation coefficient for multiclass classification.
<i>Precision</i>	Precision score for binary classification.
<i>PrecisionMicro</i>	Precision score for multiclass classification using micro averaging.
<i>PrecisionMacro</i>	Precision score for multiclass classification using macro averaging.
<i>PrecisionWeighted</i>	Precision score for multiclass classification using weighted averaging.
<i>Recall</i>	Recall score for binary classification.
<i>RecallMicro</i>	Recall score for multiclass classification using micro averaging.
<i>RecallMacro</i>	Recall score for multiclass classification using macro averaging.
<i>RecallWeighted</i>	Recall score for multiclass classification using weighted averaging.

**evalml.objectives.AccuracyBinary**

**class** evalml.objectives.**AccuracyBinary**

Accuracy score for binary classification.

**Methods**

<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>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.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** (*list*) – The classifier’s predicted probabilities
- **threshold** (*float, optional*) – Threshold used to make a prediction. Defaults to 0.5.
- **X** (*pd.DataFrame, optional*) – Any extra columns that are needed from training data.

**Returns** predictions

**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.array*): 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** (*list*) – The classifier’s predicted probabilities
- **y\_true** (*list*) – The ground truth for the predictions.
- **X** (*pd.DataFrame, optional*) – Any extra columns that are needed from training data.

**Returns** Optimal threshold for this objective

## 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.array*) – 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*)

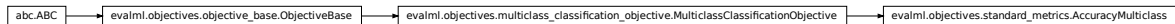
Validates the input based on a few simple checks.

### Parameters

- **y\_predicted** (*pd.Series*) – predicted values of length [n\_samples]
- **y\_true** (*pd.Series*) – actual class labels of length [n\_samples]

**Returns** None

## evalml.objectives.AccuracyMulticlass



**class** evalml.objectives.**AccuracyMulticlass**

Accuracy score for multiclass classification.

### Methods

<i>calculate_percent_difference</i>	Calculate the percent difference between scores.
<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.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.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.array*): 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.array*) – 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**

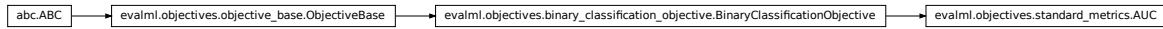
- **y\_predicted** (*pd.Series*) – predicted values of length [*n\_samples*]



- **y\_true** (*pd.Series*) – actual class labels of length [n\_samples]

**Returns** None

## evalml.objectives.AUC



**class** evalml.objectives.AUC  
AUC score for binary classification.

### Methods

<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>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.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** (*list*) – The classifier’s predicted probabilities
- **threshold** (*float, optional*) – Threshold used to make a prediction. Defaults to 0.5.
- **X** (*pd.DataFrame, optional*) – Any extra columns that are needed from training data.

**Returns** predictions

### **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.array*): 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** (*list*) – The classifier’s predicted probabilities
- **y\_true** (*list*) – The ground truth for the predictions.
- **X** (*pd.DataFrame, optional*) – Any extra columns that are needed from training data.

**Returns** Optimal threshold for this objective

### **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.array*) – extra data of shape [*n\_samples*, *n\_features*] necessary to calculate score

**Returns** score

**evalml.objectives.AUC.validate\_inputs**

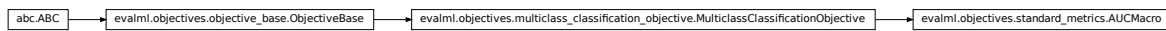
**AUC.validate\_inputs** (*y\_true*, *y\_predicted*)

Validates the input based on a few simple checks.

**Parameters**

- **y\_predicted** (*pd.Series*) – predicted values of length [n\_samples]
- **y\_true** (*pd.Series*) – actual class labels of length [n\_samples]

**Returns** None

**evalml.objectives.AUCMacro**

**class** evalml.objectives.**AUCMacro**

AUC score for multiclass classification using macro averaging.

**Methods**

<i>calculate_percent_difference</i>	Calculate the percent difference between scores.
<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.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.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 to a specified metric

**Arguments:** `y_predicted` (`pd.Series`): predicted values of length `[n_samples]` `y_true` (`pd.Series`): actual class labels of length `[n_samples]` `X` (`pd.DataFrame` or `np.array`): 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.array`) – 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)`

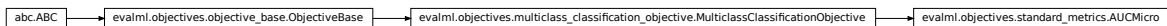
Validates the input based on a few simple checks.

#### Parameters

- **y\_predicted** (`pd.Series`) – predicted values of length `[n_samples]`
- **y\_true** (`pd.Series`) – actual class labels of length `[n_samples]`

**Returns** None

### evalml.objectives.AUCMicro



**class** `evalml.objectives.AUCMicro`

AUC score for multiclass classification using micro averaging.

#### Methods

<code>calculate_percent_difference</code>	Calculate the percent difference between scores.
<code>objective_function</code>	Computes the relative value of the provided predictions compared to the actual labels, according a specified metric
<code>score</code>	Returns a numerical score indicating performance based on the differences between the predicted and actual values.
<code>validate_inputs</code>	Validates the input based on a few simple checks.

### `evalml.objectives.AUCMicro.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.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.array*): 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.array*) – 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** (*pd.Series*) – predicted values of length [n\_samples]
- **y\_true** (*pd.Series*) – actual class labels of length [n\_samples]

**Returns** None

### evalml.objectives.AUCWeighted



**class** evalml.objectives.**AUCWeighted**

AUC Score for multiclass classification using weighted averaging.

#### Methods

<i>calculate_percent_difference</i>	Calculate the percent difference between scores.
<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.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.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.array): 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.array) – 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*)

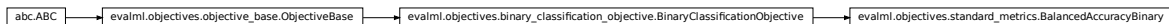
Validates the input based on a few simple checks.

#### Parameters

- **y\_predicted** (pd.Series) – predicted values of length [n\_samples]
- **y\_true** (pd.Series) – actual class labels of length [n\_samples]

**Returns** None

### evalml.objectives.BalancedAccuracyBinary



**class** evalml.objectives.BalancedAccuracyBinary

Balanced accuracy score for binary classification.

#### Methods

<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>objective_function</code>	Computes the relative value of the provided predictions compared to the actual labels, according a specified metric
<code>optimize_threshold</code>	Learn a binary classification threshold which optimizes the current objective.
<code>score</code>	Returns a numerical score indicating performance based on the differences between the predicted and actual values.
<code>validate_inputs</code>	Validates the input based on a few simple checks.

### `evalml.objectives.BalancedAccuracyBinary.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** (*list*) – The classifier’s predicted probabilities
- **threshold** (*float*, *optional*) – Threshold used to make a prediction. Defaults to 0.5.
- **X** (*pd.DataFrame*, *optional*) – Any extra columns that are needed from training data.

**Returns** `predictions`

### `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.array`): 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(y_pred_proba, y_true, X=None)`

Learn a binary classification threshold which optimizes the current objective.

#### Parameters

- **`y_pred_proba`** (`list`) – The classifier’s predicted probabilities
- **`y_true`** (`list`) – The ground truth for the predictions.
- **`X`** (`pd.DataFrame`, *optional*) – Any extra columns that are needed from training data.

**Returns** Optimal threshold for this objective

### `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.array`) – 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)`

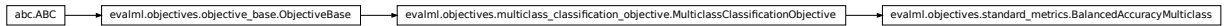
Validates the input based on a few simple checks.

#### Parameters

- **`y_predicted`** (`pd.Series`) – predicted values of length `[n_samples]`
- **`y_true`** (`pd.Series`) – actual class labels of length `[n_samples]`

**Returns** None

## evalml.objectives.BalancedAccuracyMulticlass



**class** evalml.objectives.BalancedAccuracyMulticlass

Balanced accuracy score for multiclass classification.

### Methods

<code>calculate_percent_difference</code>	Calculate the percent difference between scores.
<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.calculate\_percent\_difference

**classmethod** BalancedAccuracyMulticlass.`calculate_percent_difference` (*score*, *base-line\_score*)

Calculate the percent difference between scores.

#### Parameters

- **score** (*float*) – A score. Output of the score method of this objective.
- **baseline\_score** (*float*) – A score. Output of the score method of this objective. In practice, this is the score achieved on this objective with a baseline estimator.

#### Returns

**The percent difference between the scores. This will be the difference normalized by the baseline score.**

**Return type** float

### 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.array): 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.array*) – extra data of shape [n\_samples, n\_features] necessary to calculate score

**Returns** score

**evalml.objectives.BalancedAccuracyMulticlass.validate\_inputs**

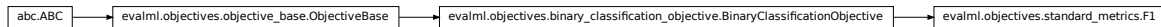
BalancedAccuracyMulticlass.**validate\_inputs** (*y\_true*, *y\_predicted*)

Validates the input based on a few simple checks.

**Parameters**

- **y\_predicted** (*pd.Series*) – predicted values of length [n\_samples]
- **y\_true** (*pd.Series*) – actual class labels of length [n\_samples]

**Returns** None

**evalml.objectives.F1**

**class** evalml.objectives.**F1**  
F1 score for binary classification.

**Methods**

<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>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.

Continued on next page

Table 112 – continued from previous page

---

<code>validate_inputs</code>	Validates the input based on a few simple checks.
------------------------------	---

---

### `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** (*list*) – The classifier’s predicted probabilities
- **threshold** (*float*, *optional*) – Threshold used to make a prediction. Defaults to 0.5.
- **X** (*pd.DataFrame*, *optional*) – Any extra columns that are needed from training data.

**Returns** predictions

### `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.array*): 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**

- **y\_pred\_proba** (*list*) – The classifier’s predicted probabilities
- **y\_true** (*list*) – The ground truth for the predictions.
- **X** (*pd.DataFrame, optional*) – Any extra columns that are needed from training data.

**Returns** Optimal threshold for this objective

**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.array*) – extra data of shape [n\_samples, n\_features] necessary to calculate score

**Returns** score

**evalml.objectives.F1.validate\_inputs**

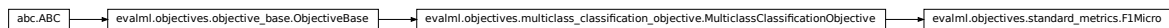
**F1.validate\_inputs** (*y\_true, y\_predicted*)

Validates the input based on a few simple checks.

**Parameters**

- **y\_predicted** (*pd.Series*) – predicted values of length [n\_samples]
- **y\_true** (*pd.Series*) – actual class labels of length [n\_samples]

**Returns** None

**evalml.objectives.F1Micro**

**class** evalml.objectives.F1Micro

F1 score for multiclass classification using micro averaging.

**Methods**


---

*calculate\_percent\_difference*

Calculate the percent difference between scores.

Continued on next page

Table 113 – continued from previous page

<i>objective_function</i>	Computes the relative value of the provided predictions compared to the actual labels, according a specified metric
<i>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.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.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.array*): 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.array*) – extra data of shape [*n\_samples*, *n\_features*] necessary to calculate score

**Returns** score

**evalml.objectives.F1Micro.validate\_inputs**

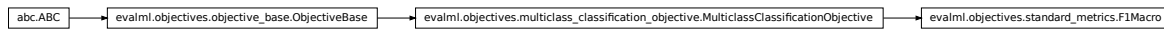
**F1Micro.validate\_inputs** (*y\_true*, *y\_predicted*)

Validates the input based on a few simple checks.

**Parameters**

- **y\_predicted** (*pd.Series*) – predicted values of length [n\_samples]
- **y\_true** (*pd.Series*) – actual class labels of length [n\_samples]

**Returns** None

**evalml.objectives.F1Macro**

**class** evalml.objectives.**F1Macro**

F1 score for multiclass classification using macro averaging.

**Methods**

<i>calculate_percent_difference</i>	Calculate the percent difference between scores.
<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.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.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.array`): 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.array`) – 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)`

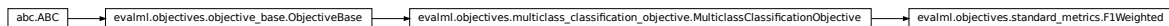
Validates the input based on a few simple checks.

#### Parameters

- **y\_predicted** (`pd.Series`) – predicted values of length `[n_samples]`
- **y\_true** (`pd.Series`) – actual class labels of length `[n_samples]`

**Returns** None

### evalml.objectives.F1Weighted



**class** `evalml.objectives.F1Weighted`

F1 score for multiclass classification using weighted averaging.

#### Methods



<code>calculate_percent_difference</code>	Calculate the percent difference between scores.
<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.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.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.array*): 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.array*) – 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** (*pd.Series*) – predicted values of length [n\_samples]
- **y\_true** (*pd.Series*) – actual class labels of length [n\_samples]

**Returns** None

### evalml.objectives.LogLossBinary



**class** evalml.objectives.**LogLossBinary**

Log Loss for binary classification.

#### Methods

<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>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.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** (*list*) – The classifier’s predicted probabilities
- **threshold** (*float*, *optional*) – Threshold used to make a prediction. Defaults to 0.5.
- **X** (*pd.DataFrame*, *optional*) – Any extra columns that are needed from training data.

**Returns** predictions

### 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.array*): 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** (*list*) – The classifier’s predicted probabilities
- **y\_true** (*list*) – The ground truth for the predictions.
- **X** (*pd.DataFrame*, *optional*) – Any extra columns that are needed from training data.

**Returns** Optimal threshold for this objective

### 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.array*) – 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)`

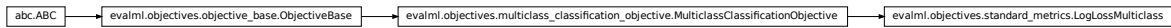
Validates the input based on a few simple checks.

#### Parameters

- **y\_predicted** (*pd.Series*) – predicted values of length [n\_samples]
- **y\_true** (*pd.Series*) – actual class labels of length [n\_samples]

**Returns** None

### `evalml.objectives.LogLossMulticlass`



**class** `evalml.objectives.LogLossMulticlass`

Log Loss for multiclass classification.

#### Methods

<code>calculate_percent_difference</code>	Calculate the percent difference between scores.
<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.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.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.array`): 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.array`) – 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** (`pd.Series`) – predicted values of length `[n_samples]`
- **y\_true** (`pd.Series`) – actual class labels of length `[n_samples]`

**Returns** None

### evalml.objectives.MCCBinary



**class** evalml.objectives.MCCBinary  
Matthews correlation coefficient for binary classification.

### Methods

<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>objective_function</code>	Computes the relative value of the provided predictions compared to the actual labels, according a specified metric
<code>optimize_threshold</code>	Learn a binary classification threshold which optimizes the current objective.
<code>score</code>	Returns a numerical score indicating performance based on the differences between the predicted and actual values.
<code>validate_inputs</code>	Validates the input based on a few simple checks.

### evalml.objectives.MCCBinary.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** (*list*) – The classifier’s predicted probabilities
- **threshold** (*float*, *optional*) – Threshold used to make a prediction. Defaults to 0.5.
- **X** (*pd.DataFrame*, *optional*) – Any extra columns that are needed from training data.

**Returns** predictions

**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.array`): 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** (`list`) – The classifier’s predicted probabilities
- **y\_true** (`list`) – The ground truth for the predictions.
- **X** (`pd.DataFrame`, *optional*) – Any extra columns that are needed from training data.

**Returns** Optimal threshold for this objective

**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.array`) – 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** (`pd.Series`) – predicted values of length `[n_samples]`
- **y\_true** (`pd.Series`) – actual class labels of length `[n_samples]`

**Returns** None

## evalml.objectives.MCCMulticlass



**class** evalml.objectives.**MCCMulticlass**  
Matthews correlation coefficient for multiclass classification.

### Methods

<code>calculate_percent_difference</code>	Calculate the percent difference between scores.
<code>objective_function</code>	Computes the relative value of the provided predictions compared to the actual labels, according a specified metric
<code>score</code>	Returns a numerical score indicating performance based on the differences between the predicted and actual values.
<code>validate_inputs</code>	Validates the input based on a few simple checks.

### evalml.objectives.MCCMulticlass.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.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.array): 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.array*) – extra data of shape [n\_samples, n\_features] necessary to calculate score

**Returns** score

**evalml.objectives.MCCMulticlass.validate\_inputs**

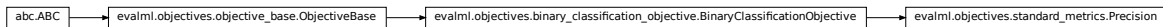
MCCMulticlass.**validate\_inputs**(*y\_true*, *y\_predicted*)

Validates the input based on a few simple checks.

**Parameters**

- **y\_predicted** (*pd.Series*) – predicted values of length [n\_samples]
- **y\_true** (*pd.Series*) – actual class labels of length [n\_samples]

**Returns** None

**evalml.objectives.Precision**

**class** evalml.objectives.**Precision**

Precision score for binary classification.

**Methods**

<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>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.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** (*list*) – The classifier’s predicted probabilities
- **threshold** (*float*, *optional*) – Threshold used to make a prediction. Defaults to 0.5.
- **X** (*pd.DataFrame*, *optional*) – Any extra columns that are needed from training data.

**Returns** predictions

### 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.array*): 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** (*list*) – The classifier’s predicted probabilities
- **y\_true** (*list*) – The ground truth for the predictions.

- **X** (*pd.DataFrame*, *optional*) – Any extra columns that are needed from training data.

**Returns** Optimal threshold for this objective

### 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.array*) – 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)`

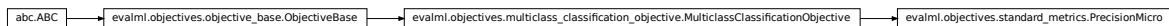
Validates the input based on a few simple checks.

#### Parameters

- **y\_predicted** (*pd.Series*) – predicted values of length [n\_samples]
- **y\_true** (*pd.Series*) – actual class labels of length [n\_samples]

**Returns** None

### evalml.objectives.PrecisionMicro



**class** evalml.objectives.PrecisionMicro

Precision score for multiclass classification using micro averaging.

#### Methods

<code>calculate_percent_difference</code>	Calculate the percent difference between scores.
<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.

Continued on next page

Table 121 – continued from previous page

---

<code>validate_inputs</code>	Validates the input based on a few simple checks.
------------------------------	---

---

### `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.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.array*): 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.array*) – extra data of shape [*n\_samples*, *n\_features*] necessary to calculate score

**Returns** score

### `evalml.objectives.PrecisionMicro.validate_inputs`

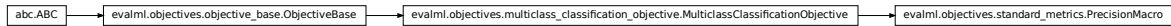
`PrecisionMicro.validate_inputs` (*y\_true*, *y\_predicted*)

Validates the input based on a few simple checks.

**Parameters**

- **y\_predicted** (*pd.Series*) – predicted values of length [n\_samples]
- **y\_true** (*pd.Series*) – actual class labels of length [n\_samples]

**Returns** None

**evalml.objectives.PrecisionMacro**

**class** evalml.objectives.PrecisionMacro

Precision score for multiclass classification using macro averaging.

**Methods**

<i>calculate_percent_difference</i>	Calculate the percent difference between scores.
<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.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.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.array`): 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.array`) – 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)`

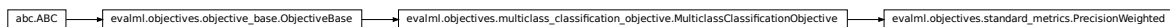
Validates the input based on a few simple checks.

#### Parameters

- **`y_predicted`** (`pd.Series`) – predicted values of length `[n_samples]`
- **`y_true`** (`pd.Series`) – actual class labels of length `[n_samples]`

**Returns** None

### `evalml.objectives.PrecisionWeighted`



**class** `evalml.objectives.PrecisionWeighted`

Precision score for multiclass classification using weighted averaging.

#### Methods

<code>calculate_percent_difference</code>	Calculate the percent difference between scores.
<code>objective_function</code>	Computes the relative value of the provided predictions compared to the actual labels, according a specified metric

Continued on next page

Table 123 – 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.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.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.array`): 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.array`) – 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*)

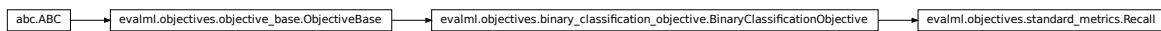
Validates the input based on a few simple checks.

#### Parameters

- **y\_predicted** (*pd.Series*) – predicted values of length [n\_samples]
- **y\_true** (*pd.Series*) – actual class labels of length [n\_samples]

**Returns** None

### evalml.objectives.Recall



**class** evalml.objectives.Recall

Recall score for binary classification.

#### Methods

<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>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.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** (*list*) – The classifier’s predicted probabilities
- **threshold** (*float, optional*) – Threshold used to make a prediction. Defaults to 0.5.
- **X** (*pd.DataFrame, optional*) – Any extra columns that are needed from training data.

**Returns** predictions

### 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.array*): 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** (*list*) – The classifier’s predicted probabilities
- **y\_true** (*list*) – The ground truth for the predictions.
- **X** (*pd.DataFrame, optional*) – Any extra columns that are needed from training data.

**Returns** Optimal threshold for this objective

### 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.array*) – 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*)

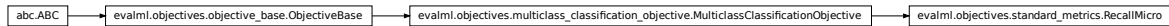
Validates the input based on a few simple checks.

#### **Parameters**

- **y\_predicted** (*pd.Series*) – predicted values of length [n\_samples]
- **y\_true** (*pd.Series*) – actual class labels of length [n\_samples]

**Returns** None

### **evalml.objectives.RecallMicro**



**class** evalml.objectives.**RecallMicro**

Recall score for multiclass classification using micro averaging.

#### **Methods**

<i>calculate_percent_difference</i>	Calculate the percent difference between scores.
<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.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.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.array`): 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.array`) – 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)`

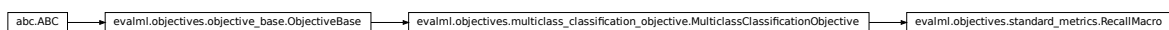
Validates the input based on a few simple checks.

#### Parameters

- **`y_predicted`** (`pd.Series`) – predicted values of length `[n_samples]`
- **`y_true`** (`pd.Series`) – actual class labels of length `[n_samples]`

**Returns** None

### `evalml.objectives.RecallMacro`



**class** `evalml.objectives.RecallMacro`  
Recall score for multiclass classification using macro averaging.

### Methods

<code>calculate_percent_difference</code>	Calculate the percent difference between scores.
<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.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.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.array*): 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.array*) – 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)`

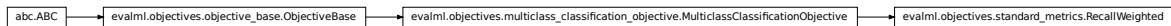
Validates the input based on a few simple checks.

#### Parameters

- **y\_predicted** (*pd.Series*) – predicted values of length [n\_samples]
- **y\_true** (*pd.Series*) – actual class labels of length [n\_samples]

**Returns** None

### evalml.objectives.RecallWeighted



**class** evalml.objectives.RecallWeighted

Recall score for multiclass classification using weighted averaging.

#### Methods

<code>calculate_percent_difference</code>	Calculate the percent difference between scores.
<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.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.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.array`): 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.array`) – 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`** (`pd.Series`) – predicted values of length `[n_samples]`
- **`y_true`** (`pd.Series`) – actual class labels of length `[n_samples]`

**Returns** None

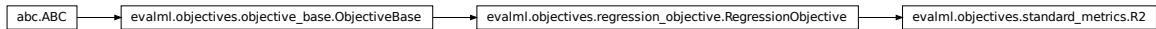
## 5.7.4 Regression Objectives

<i>R2</i>	Coefficient of determination for regression.
<i>MAE</i>	Mean absolute error for regression.
<i>MSE</i>	Mean squared error for regression.
<i>MeanSquaredLogError</i>	Mean squared log error for regression.

Continued on next page

Table 128 – continued from previous page

<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**

<i>calculate_percent_difference</i>	Calculate the percent difference between scores.
<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.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.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.array`): 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.array`) – 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`** (`pd.Series`) – predicted values of length `[n_samples]`
- **`y_true`** (`pd.Series`) – actual class labels of length `[n_samples]`

**Returns** None

### `evalml.objectives.MAE`



**class** `evalml.objectives.MAE`  
Mean absolute error for regression.

#### Methods

<code>calculate_percent_difference</code>	Calculate the percent difference between scores.
<code>objective_function</code>	Computes the relative value of the provided predictions compared to the actual labels, according a specified metric

Continued on next page



Table 130 – 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.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.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.array*): 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.array*) – 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** (*pd.Series*) – predicted values of length [n\_samples]
- **y\_true** (*pd.Series*) – actual class labels of length [n\_samples]

**Returns** None

### evalml.objectives.MSE



**class** evalml.objectives.MSE  
Mean squared error for regression.

#### Methods

<i>calculate_percent_difference</i>	Calculate the percent difference between scores.
<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.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.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.array): 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.array) – 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** (pd.Series) – predicted values of length [n\_samples]
- **y\_true** (pd.Series) – actual class labels of length [n\_samples]

**Returns** None

## 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>calculate_percent_difference</code>	Calculate the percent difference between scores.
<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.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.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.array*): 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.array*) – 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** (*pd.Series*) – predicted values of length [n\_samples]
- **y\_true** (*pd.Series*) – actual class labels of length [n\_samples]

**Returns** None

### evalml.objectives.MedianAE



**class** evalml.objectives.MedianAE

Median absolute error for regression.

#### Methods

<i>calculate_percent_difference</i>	Calculate the percent difference between scores.
<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.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.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.array`): extra data of shape `[n_samples, n_features]` necessary to calculate score

**Returns** numerical value used to calculate score

### `evalml.objectives.MedianAE.score`

`MedianAE.score(y_true, y_predicted, X=None)`

Returns a numerical score indicating performance based on the differences between the predicted and actual values.

#### **Parameters**

- **`y_predicted`** (`pd.Series`) – predicted values of length `[n_samples]`
- **`y_true`** (`pd.Series`) – actual class labels of length `[n_samples]`
- **`X`** (`pd.DataFrame` or `np.array`) – 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`** (`pd.Series`) – predicted values of length `[n_samples]`
- **`y_true`** (`pd.Series`) – actual class labels of length `[n_samples]`

**Returns** None

### `evalml.objectives.MaxError`



**class** `evalml.objectives.MaxError`

Maximum residual error for regression.

## Methods

<code>calculate_percent_difference</code>	Calculate the percent difference between scores.
<code>objective_function</code>	Computes the relative value of the provided predictions compared to the actual labels, according a specified metric
<code>score</code>	Returns a numerical score indicating performance based on the differences between the predicted and actual values.
<code>validate_inputs</code>	Validates the input based on a few simple checks.

### `evalml.objectives.MaxError.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.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.array*): 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.array*) – 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** (*pd.Series*) – predicted values of length [n\_samples]
- **y\_true** (*pd.Series*) – actual class labels of length [n\_samples]

**Returns** None

### `evalml.objectives.ExpVariance`



**class** `evalml.objectives.ExpVariance`

Explained variance score for regression.

#### Methods

<code>calculate_percent_difference</code>	Calculate the percent difference between scores.
<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.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.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.array`): 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.array`) – 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)`

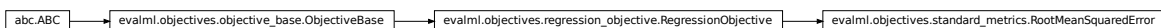
Validates the input based on a few simple checks.

#### Parameters

- **y\_predicted** (`pd.Series`) – predicted values of length `[n_samples]`
- **y\_true** (`pd.Series`) – actual class labels of length `[n_samples]`

**Returns** None

### evalml.objectives.RootMeanSquaredError



**class** evalml.objectives.RootMeanSquaredError

Root mean squared error for regression.

## Methods

<code>calculate_percent_difference</code>	Calculate the percent difference between scores.
<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.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.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.array*): 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.array*) – 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** (*pd.Series*) – predicted values of length [n\_samples]
- **y\_true** (*pd.Series*) – actual class labels of length [n\_samples]

**Returns** None

### 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

<i>calculate_percent_difference</i>	Calculate the percent difference between scores.
<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.RootMeanSquaredLogError.calculate\_percent\_difference

**classmethod** RootMeanSquaredLogError.**calculate\_percent\_difference** (*score*, *base-line\_score*)

Calculate the percent difference between scores.

#### Parameters

- **score** (*float*) – A score. Output of the score method of this objective.
- **baseline\_score** (*float*) – A score. Output of the score method of this objective. In practice, this is the score achieved on this objective with a baseline estimator.

**Returns**

The percent difference between the scores. This will be the difference normalized by the baseline score.

**Return type** float

**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.array`): 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.array`) – 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** (`pd.Series`) – predicted values of length `[n_samples]`
- **y\_true** (`pd.Series`) – actual class labels of length `[n_samples]`

**Returns** None

## 5.8 Problem Types

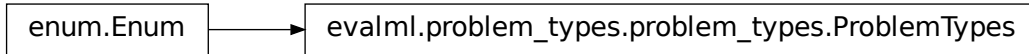
---

*ProblemTypes*

Enum for type of machine learning problem: BINARY, MULTICLASS, or REGRESSION.

---

### 5.8.1 evalml.problem\_types.ProblemTypes



**class** evalml.problem\_types.**ProblemTypes**

Enum for type of machine learning problem: BINARY, MULTICLASS, or REGRESSION.

---

*handle\_problem\_types*

Handles problem\_type by either returning the ProblemTypes or converting from a str.

---

### 5.8.2 evalml.problem\_types.handle\_problem\_types

evalml.problem\_types.**handle\_problem\_types** (*problem\_type*)

Handles problem\_type by either returning the ProblemTypes or converting from a str.

**Parameters** *problem\_type* (*str* or *ProblemTypes*) – problem type that needs to be handled

**Returns** ProblemTypes

## 5.9 Model Family

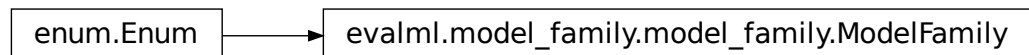
---

*ModelFamily*

Enum for family of machine learning models.

---

### 5.9.1 evalml.model\_family.ModelFamily



**class** evalml.model\_family.**ModelFamily**

Enum for family of machine learning models.

---

<code>handle_model_family</code>	Handles <code>model_family</code> by either returning the <code>ModelFamily</code> or converting from a <code>str</code> :param <code>model_family</code> : model type that needs to be handled :type <code>model_family</code> : <code>str</code> or <code>ModelFamily</code>
----------------------------------	--

---

5.9.2 `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 `str` :param `model_family`: model type that needs to be handled :type `model_family`: `str` or `ModelFamily`  
**Returns** `ModelFamily`

5.10 Tuners

---

<code>Tuner</code>	Defines API for Tuners.
<code>SKOptTuner</code>	Bayesian Optimizer.
<code>GridSearchTuner</code>	Grid Search Optimizer.
<code>RandomSearchTuner</code>	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**

---

<code>__init__</code>	Base Tuner class
-----------------------	------------------

---

Continued on next page

Table 143 – continued from previous page

<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**

`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**

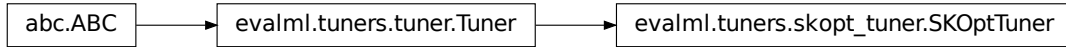
`Tuner.propose()`

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

## 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

## 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()`

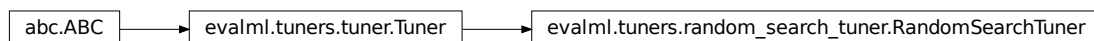
Returns parameters from `_grid_points` iterations

If all possible combinations of parameters have been scored, then `NoParamsException` is raised.

**Returns** proposed pipeline parameters

**Return type** dict

## 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.

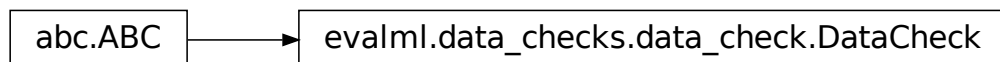
**Returns** proposed pipeline parameters

**Return type** dict

## 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 labels contain missing or invalid data.
<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>LabelLeakageDataCheck</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 an Isolation Forest to obtain the anomaly score of each index and then using IQR to determine score anomalies.
<i>NoVarianceDataCheck</i>	Check if any of the features or labels have no variance.

**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:

<i>validate</i>	Inspects and validates the input data, runs any necessary calculations or algorithms, and returns a list of warnings and errors if applicable.
-----------------	--

#### evalml.data\_checks.DataCheck.validate

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.

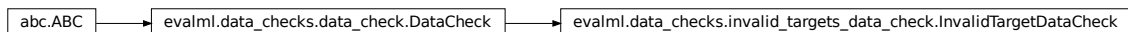
##### Parameters

- **X** (*pd.DataFrame*) – the input data of shape [n\_samples, n\_features]
- **y** (*pd.Series*, *optional*) – the target data of length [n\_samples]

**Returns** list of DataCheckError and DataCheckWarning objects

**Return type** list (*DataCheckMessage*)

#### evalml.data\_checks.InvalidTargetDataCheck



**class** evalml.data\_checks.InvalidTargetDataCheck

Checks if the target labels contain missing or invalid data.

**name** = 'InvalidTargetDataCheck'

#### Instance attributes

#### Methods:

<code>validate</code>	Checks if the target labels contain missing or invalid data.
-----------------------	--

### `evalml.data_checks.InvalidTargetDataCheck.validate`

`InvalidTargetDataCheck.validate(X, y)`  
Checks if the target labels contain missing or invalid data.

#### Parameters

- **X** (`pd.DataFrame`, `pd.Series`, `np.array`, `list`) – Features. Ignored.
- **y** – Target labels to check for invalid data.

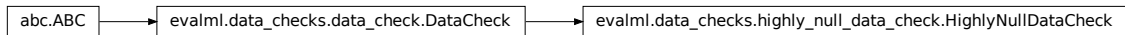
**Returns** list with `DataCheckErrors` if any invalid data is found in target labels.

**Return type** list (`DataCheckError`)

#### Example

```
>>> X = pd.DataFrame({})
>>> y = pd.Series([0, 1, None, None])
>>> target_check = InvalidTargetDataCheck()
>>> assert target_check.validate(X, y) == [DataCheckError("2 row(s) (50.0%)
of target values are null", "InvalidTargetDataCheck")]
```

### `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** (*pd.DataFrame, pd.Series, np.array, list*) – features
- **y** – Ignored.

**Returns** list with a `DataCheckWarning` if there are any highly-null columns.

**Return type** list (*DataCheckWarning*)

**Example**

```
>>> 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) == [DataCheckWarning("Column 'lots_of_null"
↳ ' is 80.0% or more null", "HighlyNullDataCheck")]
```

**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**

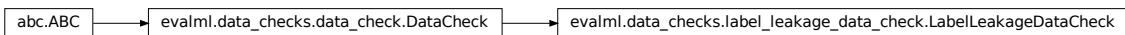
- `X` (*pd.DataFrame*) – The input features to check
- `threshold` (*float*) – the probability threshold to be considered an ID column. Defaults to 1.0

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

### Example

```
>>> 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) == [DataCheckWarning("Column 'df_id' is
↳ 100.0% or more likely to be an ID column", "IDColumnsDataCheck")]
```

### `evalml.data_checks.LabelLeakageDataCheck`





**class** evalml.data\_checks.LabelLeakageDataCheck (*pct\_corr\_threshold=0.95*)

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

**name** = 'LabelLeakageDataCheck'

### 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.LabelLeakageDataCheck.\_\_init\_\_

LabelLeakageDataCheck.**\_\_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.LabelLeakageDataCheck.validate

LabelLeakageDataCheck.**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** (*pd.DataFrame*) – The input features to check
- **y** (*pd.Series*) – The labels

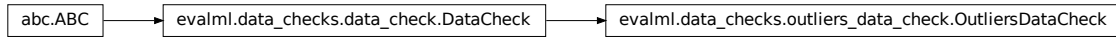
**Returns** list with a DataCheckWarning if there is label leakage detected.

**Return type** list (*DataCheckWarning*)

### Example

```
>>> 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])
>>> label_leakage_check = LabelLeakageDataCheck(pct_corr_threshold=0.8)
>>> assert label_leakage_check.validate(X, y) == [DataCheckWarning("Column
↪ 'leak' is 80.0% or more correlated with the target", "LabelLeakageDataCheck
↪ ")]
```

## evalml.data\_checks.OutliersDataCheck



**class** evalml.data\_checks.OutliersDataCheck (random\_state=0)

Checks if there are any outliers in input data by using an Isolation Forest to obtain the anomaly score of each index and then using IQR to determine score anomalies. Indices with score anomalies are considered 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 an Isolation Forest to obtain the anomaly score of each index and then using IQR to determine score 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 an Isolation Forest to obtain the anomaly score of each index and then using IQR to determine score anomalies. Indices with score anomalies are considered outliers.

#### Parameters

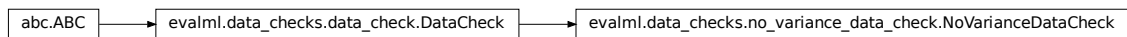
- **X** (*pd.DataFrame*) – features
- **y** – Ignored.

**Returns** A set of indices that may have outlier data.

### Example

```
>>> df = pd.DataFrame({
...     'x': [1, 2, 3, 40, 5],
...     'y': [6, 7, 8, 990, 10],
...     'z': [-1, -2, -3, -1201, -4]
... })
>>> outliers_check = OutliersDataCheck()
>>> assert outliers_check.validate(df) == [DataCheckWarning("Row '3' is_
↳ likely to have outlier data", "OutliersDataCheck")]
```

### evalml.data\_checks.NoVarianceDataCheck



**class** evalml.data\_checks.NoVarianceDataCheck (*count\_nan\_as\_value=False*)

Check if any of the features or labels have no variance.

**name** = 'NoVarianceDataCheck'

#### Instance attributes

#### Methods:

<code>__init__</code>	Check if any of the features or labels have no variance.
<code>validate</code>	Check if any of the features or if the labels have no variance (1 unique value).

### evalml.data\_checks.NoVarianceDataCheck.\_\_init\_\_

NoVarianceDataCheck.**\_\_init\_\_** (*count\_nan\_as\_value=False*)

Check if any of the features or labels 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 any of the features or if the labels have no variance (1 unique value).

**Parameters**

- **x** (*pd.DataFrame*) – The input features.
- **y** (*pd.Series*) – The labels.

**Returns** list (*DataCheckWarning* or *DataCheckError*), list of warnings/errors corresponding to features or labels with no variance.

<i>DataChecks</i>	A collection of data checks.
<i>DefaultDataChecks</i>	A collection of basic data checks that is used by AutoML by default.

## evalml.data\_checks.DataChecks

evalml.data\_checks.data\_checks.DataChecks

**class** evalml.data\_checks.DataChecks (*data\_checks=None*)  
A collection of data checks.

### Methods

<i>__init__</i>	A collection of data checks.
<i>validate</i>	Inspects and validates the input data against data checks and returns a list of warnings and errors if applicable.

### evalml.data\_checks.DataChecks.\_\_init\_\_

DataChecks.**\_\_init\_\_** (*data\_checks=None*)  
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.

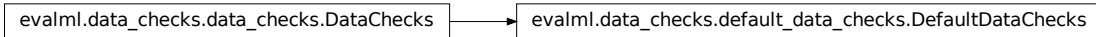
**Parameters**

- **X** (*pd.DataFrame*) – the input data of shape [n\_samples, n\_features]
- **y** (*pd.Series*) – the target labels of length [n\_samples]

**Returns** list containing DataCheckMessage objects

**Return type** list (*DataCheckMessage*)

## evalml.data\_checks.DefaultDataChecks



**class** evalml.data\_checks.DefaultDataChecks (*data\_checks=None*)

A collection of basic data checks that is used by AutoML by default.

Includes HighlyNullDataCheck, IDColumnsDataCheck, LabelLeakageDataCheck, InvalidTargetDataCheck, and NoVarianceDataCheck.

### 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\_\_** (*data\_checks=None*)

A collection of basic data checks.

**Parameters** **data\_checks** (*list (DataCheck)*) – Ignored.

### 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** (*pd.DataFrame*) – the input data of shape [n\_samples, n\_features]
- **y** (*pd.Series*) – the target labels of length [n\_samples]

**Returns** list containing DataCheckMessage objects

**Return type** list (*DataCheckMessage*)

### 5.11.2 Data Check Messages

<i>DataCheckMessage</i>	Base class for all DataCheckMessages.
<i>DataCheckError</i>	DataCheckMessage subclass for errors returned by data checks.
<i>DataCheckWarning</i>	DataCheckMessage subclass for warnings returned by data checks.

#### evalml.data\_checks.DataCheckMessage

evalml.data\_checks.data\_check\_message.DataCheckMessage

**class** evalml.data\_checks.DataCheckMessage(*message*, *data\_check\_name*)  
Base class for all DataCheckMessages.

**message\_type** = None

#### Methods:

<code>__init__</code>	Message returned by a DataCheck, tagged by name.”
<code>__str__</code>	String representation of data check message, equivalent to self.message attribute.
<code>__eq__</code>	Checks for equality.

#### evalml.data\_checks.DataCheckMessage.\_\_init\_\_

DataCheckMessage.**\_\_init\_\_**(*message*, *data\_check\_name*)  
Message returned by a DataCheck, tagged by name.”

##### Parameters

- **message** (*str*) – Message string
- **data\_check\_name** (*str*) – Name of data check

#### 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 their message type and message are equivalent.

**evalml.data\_checks.DataCheckError**

**class** `evalml.data_checks.DataCheckError` (*message*, *data\_check\_name*)

`DataCheckMessage` subclass for errors returned by data checks.

**message\_type** = 'error'

**Methods:**

<code>__init__</code>	Message returned by a <code>DataCheck</code> , tagged by name.”
<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.DataCheckError.\_\_init\_\_**

`DataCheckError.__init__` (*message*, *data\_check\_name*)

Message returned by a `DataCheck`, tagged by name.”

**Parameters**

- **message** (*str*) – Message string
- **data\_check\_name** (*str*) – Name of data check

**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 their message type and message are equivalent.

## evalml.data\_checks.DataCheckWarning



**class** evalml.data\_checks.DataCheckWarning(*message*, *data\_check\_name*)  
DataCheckMessage subclass for warnings returned by data checks.  
**message\_type** = 'warning'

### Methods:

<code>__init__</code>	Message returned by a DataCheck, tagged by name.”
<code>__str__</code>	String representation of data check message, equivalent to self.message attribute.
<code>__eq__</code>	Checks for equality.

### evalml.data\_checks.DataCheckWarning.\_\_init\_\_

DataCheckWarning.**\_\_init\_\_**(*message*, *data\_check\_name*)  
Message returned by a DataCheck, tagged by name.”

#### Parameters

- **message** (*str*) – Message string
- **data\_check\_name** (*str*) – Name of data check

### 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 their message type and message are equivalent.

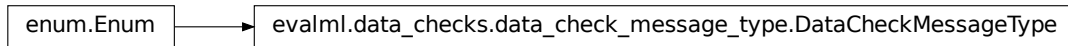
## 5.11.3 Data Check Message Types

---

<i>DataCheckMessageType</i>	Enum for type of data check message: WARNING or ERROR.
-----------------------------	--

---



**evalml.data\_checks.DataCheckMessageType**

**class** evalml.data\_checks.DataCheckMessageType  
 Enum for type of data check message: WARNING or ERROR.

## 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.

**evalml.utils.import\_or\_raise**

`evalml.utils.import_or_raise(library, error_msg=None)`  
 Attempts to import the requested library by name. If the import fails, raises an `ImportError`.

**Parameters**

- **library** (*str*) – the name of the library
- **error\_msg** (*str*) – error message to return if the import fails

**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`

## RELEASE NOTES

### Future Releases

- Enhancements
- Fixes
- Changes
- Documentation Changes
- Testing Changes

#### 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 *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 *feature\_threshold*, which allows for thresholding feature importance #1097
- **Documentation Changes**
    - Update setup.py URL to point to the github repo #1037
    - Added tutorial for using the cost-benefit matrix objective #1088
  - **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**

- Update 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
- Update FAQ section formatting #997
- Update release process documentation #1003

- **Testing Changes**

- Moved *predict\_proba* and *predict* tests regarding string / categorical targets to *test\_pipelines.py* [#972](#)
- Fix 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/multiseries 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 PipelinePlots 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

<code>__eq__()</code> ( <i>evalml.data_checks.DataCheckError</i> method), 283	<code>__init__()</code> ( <i>evalml.pipelines.BaselineRegressionPipeline</i> method), 104
<code>__eq__()</code> ( <i>evalml.data_checks.DataCheckMessage</i> method), 283	<code>__init__()</code> ( <i>evalml.pipelines.BinaryClassificationPipeline</i> method), 78
<code>__eq__()</code> ( <i>evalml.data_checks.DataCheckWarning</i> method), 284	<code>__init__()</code> ( <i>evalml.pipelines.ClassificationPipeline</i> method), 75
<code>__init__()</code> ( <i>evalml.automl.AutoMLSearch</i> method), 64	<code>__init__()</code> ( <i>evalml.pipelines.MeanBaselineRegressionPipeline</i> method), 108
<code>__init__()</code> ( <i>evalml.automl.automl_algorithm.AutoMLAlgorithm</i> method), 68	<code>__init__()</code> ( <i>evalml.pipelines.ModeBaselineBinaryPipeline</i> method), 96
<code>__init__()</code> ( <i>evalml.automl.automl_algorithm.IterativeAlgorithm</i> method), 70	<code>__init__()</code> ( <i>evalml.pipelines.ModeBaselineMulticlassPipeline</i> method), 100
<code>__init__()</code> ( <i>evalml.data_checks.DataCheckError</i> method), 283	<code>__init__()</code> ( <i>evalml.pipelines.MulticlassClassificationPipeline</i> method), 82
<code>__init__()</code> ( <i>evalml.data_checks.DataCheckMessage</i> method), 282	<code>__init__()</code> ( <i>evalml.pipelines.PipelineBase</i> method), 72
<code>__init__()</code> ( <i>evalml.data_checks.DataCheckWarning</i> method), 284	<code>__init__()</code> ( <i>evalml.pipelines.RegressionPipeline</i> method), 85
<code>__init__()</code> ( <i>evalml.data_checks.DataChecks</i> method), 280	<code>__init__()</code> ( <i>evalml.pipelines.components.BaselineClassifier</i> method), 166
<code>__init__()</code> ( <i>evalml.data_checks.DefaultDataChecks</i> method), 281	<code>__init__()</code> ( <i>evalml.pipelines.components.BaselineRegressor</i> method), 184
<code>__init__()</code> ( <i>evalml.data_checks.HighlyNullDataCheck</i> method), 275	<code>__init__()</code> ( <i>evalml.pipelines.components.CatBoostClassifier</i> method), 151
<code>__init__()</code> ( <i>evalml.data_checks.IDColumnsDataCheck</i> method), 276	<code>__init__()</code> ( <i>evalml.pipelines.components.CatBoostRegressor</i> method), 169
<code>__init__()</code> ( <i>evalml.data_checks.LabelLeakageDataCheck</i> method), 277	<code>__init__()</code> ( <i>evalml.pipelines.components.ComponentBase</i> method), 111
<code>__init__()</code> ( <i>evalml.data_checks.NoVarianceDataCheck</i> method), 279	<code>__init__()</code> ( <i>evalml.pipelines.components.DateTimeFeaturizer</i> method), 146
<code>__init__()</code> ( <i>evalml.data_checks.OutliersDataCheck</i> method), 278	<code>__init__()</code> ( <i>evalml.pipelines.components.DropColumns</i> method), 119
<code>__init__()</code> ( <i>evalml.objectives.CostBenefitMatrix</i> method), 206	<code>__init__()</code> ( <i>evalml.pipelines.components.DropNullColumns</i> method), 143
<code>__init__()</code> ( <i>evalml.objectives.FraudCost</i> method), 201	<code>__init__()</code> ( <i>evalml.pipelines.components.ElasticNetClassifier</i> method), 154
<code>__init__()</code> ( <i>evalml.objectives.LeadScoring</i> method), 204	<code>__init__()</code> ( <i>evalml.pipelines.components.ElasticNetRegressor</i> method), 172
<code>__init__()</code> ( <i>evalml.pipelines.BaselineBinaryPipeline</i> method), 89	<code>__init__()</code> ( <i>evalml.pipelines.components.Estimator</i> method), 116
<code>__init__()</code> ( <i>evalml.pipelines.BaselineMulticlassPipeline</i> method), 93	<code>__init__()</code> ( <i>evalml.pipelines.components.ExtraTreesClassifier</i> method), 116

`method`), 156  
`__init__()` (`evalml.pipelines.components.ExtraTreesRegressor` `method`), 177  
`__init__()` (`evalml.pipelines.components.Imputer` `method`), 130  
`__init__()` (`evalml.pipelines.components.LinearRegressor` `method`), 174  
`__init__()` (`evalml.pipelines.components.LogisticRegressionClassifier` `method`), 161  
`__init__()` (`evalml.pipelines.components.OneHotEncoder` `method`), 124  
`__init__()` (`evalml.pipelines.components.PerColumnImputer` `method`), 127  
`__init__()` (`evalml.pipelines.components.RFClassifier` `method`), 141  
`__init__()` (`evalml.pipelines.components.RFRegressor` `method`), 138  
`__init__()` (`evalml.pipelines.components.RandomForestClassifier` `method`), 159  
`__init__()` (`evalml.pipelines.components.RandomForestRegressor` `method`), 179  
`__init__()` (`evalml.pipelines.components.SelectColumns` `method`), 122  
`__init__()` (`evalml.pipelines.components.SimpleImputer` `method`), 133  
`__init__()` (`evalml.pipelines.components.StandardScaler` `method`), 135  
`__init__()` (`evalml.pipelines.components.TextFeaturizer` `method`), 148  
`__init__()` (`evalml.pipelines.components.Transformer` `method`), 113  
`__init__()` (`evalml.pipelines.components.XGBoostClassifier` `method`), 164  
`__init__()` (`evalml.pipelines.components.XGBoostRegressor` `method`), 182  
`__init__()` (`evalml.tuners.GridSearchTuner` `method`), 269  
`__init__()` (`evalml.tuners.RandomSearchTuner` `method`), 271  
`__init__()` (`evalml.tuners.SKOptTuner` `method`), 268  
`__init__()` (`evalml.tuners.Tuner` `method`), 267  
`__str__()` (`evalml.data_checks.DataCheckError` `method`), 283  
`__str__()` (`evalml.data_checks.DataCheckMessage` `method`), 282  
`__str__()` (`evalml.data_checks.DataCheckWarning` `method`), 284

## A

`AccuracyBinary` (class in `evalml.objectives`), 209  
`AccuracyMulticlass` (class in `evalml.objectives`), 211  
`add()` (`evalml.tuners.GridSearchTuner` `method`), 270

`add()` (`evalml.tuners.RandomSearchTuner` `method`), 271  
`add()` (`evalml.tuners.SKOptTuner` `method`), 268  
`add()` (`evalml.tuners.Tuner` `method`), 267  
`add_result()` (`evalml.automl.automl_algorithm.AutoMLAlgorithm` `method`), 69  
`add_result()` (`evalml.automl.automl_algorithm.IterativeAlgorithm` `method`), 70  
`add_to_rankings()` (`evalml.automl.AutoMLSearch` `method`), 66  
`allowed_model_families()` (in module `evalml.pipelines.components.utils`), 117  
`AUC` (class in `evalml.objectives`), 213  
`AUCFromModel` (class in `evalml.objectives`), 215  
`AUCMicro` (class in `evalml.objectives`), 216  
`AUCMacro` (class in `evalml.objectives`), 218  
`AutoMLAlgorithm` (class in `evalml.automl.automl_algorithm`), 68  
`AutoMLSearch` (class in `evalml.automl`), 64

## B

`BalancedAccuracyBinary` (class in `evalml.objectives`), 219  
`BalancedAccuracyMulticlass` (class in `evalml.objectives`), 222  
`BaselineBinaryPipeline` (class in `evalml.pipelines`), 88  
`BaselineClassifier` (class in `evalml.pipelines.components`), 166  
`BaselineMulticlassPipeline` (class in `evalml.pipelines`), 92  
`BaselineRegressionPipeline` (class in `evalml.pipelines`), 103  
`BaselineRegressor` (class in `evalml.pipelines.components`), 183  
`binary_objective_vs_threshold()` (in module `evalml.model_understanding`), 190  
`BinaryClassificationObjective` (class in `evalml.objectives`), 195  
`BinaryClassificationPipeline` (class in `evalml.pipelines`), 78

## C

`calculate_percent_difference()` (`evalml.objectives.AccuracyBinary` class `method`), 209  
`calculate_percent_difference()` (`evalml.objectives.AccuracyMulticlass` class `method`), 212  
`calculate_percent_difference()` (`evalml.objectives.AUC` class `method`), 213  
`calculate_percent_difference()` (`evalml.objectives.AUCMacro` class `method`), 215

`calculate_percent_difference()`  
     (`evalml.objectives.AUCMicro` class method), 217  
`calculate_percent_difference()`  
     (`evalml.objectives.AUCWeighted` class method), 218  
`calculate_percent_difference()`  
     (`evalml.objectives.BalancedAccuracyBinary` class method), 220  
`calculate_percent_difference()`  
     (`evalml.objectives.BalancedAccuracyMulticlass` class method), 222  
`calculate_percent_difference()`  
     (`evalml.objectives.BinaryClassificationObjective` class method), 195  
`calculate_percent_difference()`  
     (`evalml.objectives.CostBenefitMatrix` class method), 206  
`calculate_percent_difference()`  
     (`evalml.objectives.ExpVariance` class method), 260  
`calculate_percent_difference()`  
     (`evalml.objectives.F1` class method), 224  
`calculate_percent_difference()`  
     (`evalml.objectives.F1Macro` class method), 227  
`calculate_percent_difference()`  
     (`evalml.objectives.F1Micro` class method), 226  
`calculate_percent_difference()`  
     (`evalml.objectives.F1Weighted` class method), 229  
`calculate_percent_difference()`  
     (`evalml.objectives.FraudCost` class method), 201  
`calculate_percent_difference()`  
     (`evalml.objectives.LeadScoring` class method), 204  
`calculate_percent_difference()`  
     (`evalml.objectives.LogLossBinary` class method), 230  
`calculate_percent_difference()`  
     (`evalml.objectives.LogLossMulticlass` class method), 232  
`calculate_percent_difference()`  
     (`evalml.objectives.MAE` class method), 253  
`calculate_percent_difference()`  
     (`evalml.objectives.MaxError` class method), 259  
`calculate_percent_difference()`  
     (`evalml.objectives.MCCBinary` class method), 234  
`calculate_percent_difference()`  
     (`evalml.objectives.MCCMulticlass` class method), 236  
`calculate_percent_difference()`  
     (`evalml.objectives.MeanSquaredLogError` class method), 256  
`calculate_percent_difference()`  
     (`evalml.objectives.MedianAE` class method), 257  
`calculate_percent_difference()`  
     (`evalml.objectives.MSE` class method), 254  
`calculate_percent_difference()`  
     (`evalml.objectives.MulticlassClassificationObjective` class method), 198  
`calculate_percent_difference()`  
     (`evalml.objectives.ObjectiveBase` class method), 194  
`calculate_percent_difference()`  
     (`evalml.objectives.Precision` class method), 238  
`calculate_percent_difference()`  
     (`evalml.objectives.PrecisionMacro` class method), 241  
`calculate_percent_difference()`  
     (`evalml.objectives.PrecisionMicro` class method), 240  
`calculate_percent_difference()`  
     (`evalml.objectives.PrecisionWeighted` class method), 243  
`calculate_percent_difference()`  
     (`evalml.objectives.R2` class method), 251  
`calculate_percent_difference()`  
     (`evalml.objectives.Recall` class method), 244  
`calculate_percent_difference()`  
     (`evalml.objectives.RecallMacro` class method), 248  
`calculate_percent_difference()`  
     (`evalml.objectives.RecallMicro` class method), 246  
`calculate_percent_difference()`  
     (`evalml.objectives.RecallWeighted` class method), 249  
`calculate_percent_difference()`  
     (`evalml.objectives.RegressionObjective` class method), 199  
`calculate_percent_difference()`  
     (`evalml.objectives.RootMeanSquaredError` class method), 262  
`calculate_percent_difference()`  
     (`evalml.objectives.RootMeanSquaredLogError` class method), 263  
`calculate_permutation_importance()` (in module `evalml.model_understanding`), 189  
`CatBoostClassifier` (class in `evalml.pipelines.components`), 151

CatBoostRegressor	(class in <code>evalml.pipelines.components</code> ), 168	<code>clone()</code> ( <code>evalml.pipelines.components.RFClassifierSelectFromModel</code> method), 141
ClassificationPipeline	(class in <code>evalml.pipelines</code> ), 74	<code>clone()</code> ( <code>evalml.pipelines.components.RFRegressorSelectFromModel</code> method), 138
<code>clone()</code>	( <code>evalml.pipelines.BaselineBinaryPipeline</code> method), 89	<code>clone()</code> ( <code>evalml.pipelines.components.SelectColumns</code> method), 122
<code>clone()</code>	( <code>evalml.pipelines.BaselineMulticlassPipeline</code> method), 93	<code>clone()</code> ( <code>evalml.pipelines.components.SimpleImputer</code> method), 133
<code>clone()</code>	( <code>evalml.pipelines.BaselineRegressionPipeline</code> method), 104	<code>clone()</code> ( <code>evalml.pipelines.components.StandardScaler</code> method), 135
<code>clone()</code>	( <code>evalml.pipelines.BinaryClassificationPipeline</code> method), 79	<code>clone()</code> ( <code>evalml.pipelines.components.TextFeaturizer</code> method), 149
<code>clone()</code>	( <code>evalml.pipelines.ClassificationPipeline</code> method), 75	<code>clone()</code> ( <code>evalml.pipelines.components.Transformer</code> method), 113
<code>clone()</code>	( <code>evalml.pipelines.components.BaselineClassifier</code> method), 167	<code>clone()</code> ( <code>evalml.pipelines.components.XGBoostClassifier</code> method), 164
<code>clone()</code>	( <code>evalml.pipelines.components.BaselineRegressor</code> method), 184	<code>clone()</code> ( <code>evalml.pipelines.components.XGBoostRegressor</code> method), 182
<code>clone()</code>	( <code>evalml.pipelines.components.CatBoostClassifier</code> method), 152	<code>clone()</code> ( <code>evalml.pipelines.MeanBaselineRegressionPipeline</code> method), 108
<code>clone()</code>	( <code>evalml.pipelines.components.CatBoostRegressor</code> method), 169	<code>clone()</code> ( <code>evalml.pipelines.ModeBaselineBinaryPipeline</code> method), 97
<code>clone()</code>	( <code>evalml.pipelines.components.ComponentBase</code> method), 111	<code>clone()</code> ( <code>evalml.pipelines.ModeBaselineMulticlassPipeline</code> method), 100
<code>clone()</code>	( <code>evalml.pipelines.components.DateTimeFeaturizer</code> method), 146	<code>clone()</code> ( <code>evalml.pipelines.MulticlassClassificationPipeline</code> method), 82
<code>clone()</code>	( <code>evalml.pipelines.components.DropColumns</code> method), 119	<code>clone()</code> ( <code>evalml.pipelines.PipelineBase</code> method), 72
<code>clone()</code>	( <code>evalml.pipelines.components.DropNullColumns</code> method), 143	<code>clone()</code> ( <code>evalml.pipelines.RegressionPipeline</code> method), 85
<code>clone()</code>	( <code>evalml.pipelines.components.ElasticNetClassifier</code> method), 154	<code>component_graph</code> ( <code>evalml.pipelines.BaselineBinaryPipeline</code> attribute), 88
<code>clone()</code>	( <code>evalml.pipelines.components.ElasticNetRegressor</code> method), 172	<code>component_graph</code> ( <code>evalml.pipelines.BaselineMulticlassPipeline</code> attribute), 92
<code>clone()</code>	( <code>evalml.pipelines.components.Estimator</code> method), 116	<code>component_graph</code> ( <code>evalml.pipelines.BaselineRegressionPipeline</code> attribute), 103
<code>clone()</code>	( <code>evalml.pipelines.components.ExtraTreesClassifier</code> method), 157	<code>component_graph</code> ( <code>evalml.pipelines.MeanBaselineRegressionPipeline</code> attribute), 107
<code>clone()</code>	( <code>evalml.pipelines.components.ExtraTreesRegressor</code> method), 177	<code>component_graph</code> ( <code>evalml.pipelines.ModeBaselineBinaryPipeline</code> attribute), 95
<code>clone()</code>	( <code>evalml.pipelines.components.Imputer</code> method), 130	<code>component_graph</code> ( <code>evalml.pipelines.ModeBaselineMulticlassPipeline</code> attribute), 99
<code>clone()</code>	( <code>evalml.pipelines.components.LinearRegressor</code> method), 174	<code>ComponentBase</code> (class in <code>evalml.pipelines.components</code> ), 111
<code>clone()</code>	( <code>evalml.pipelines.components.LogisticRegressionClassifier</code> method), 162	<code>confusion_matrix()</code> (in module <code>evalml.model_understanding</code> ), 186
<code>clone()</code>	( <code>evalml.pipelines.components.OneHotEncoder</code> method), 125	<code>convert_to_seconds()</code> (in module <code>evalml.utils</code> ), 285
<code>clone()</code>	( <code>evalml.pipelines.components.PerColumnImputer</code> method), 128	<code>CostBenefitMatrix</code> (class in <code>evalml.objectives</code> ), 206
<code>clone()</code>	( <code>evalml.pipelines.components.RandomForestClassifier</code> method), 159	<code>custom_hyperparameters</code> ( <code>evalml.pipelines.BaselineBinaryPipeline</code> attribute), 88
<code>clone()</code>	( <code>evalml.pipelines.components.RandomForestRegressor</code> method), 179	<code>extract_hyperparameters</code> ( <code>evalml.pipelines.BaselineMulticlassPipeline</code> attribute), 91



- attribute*), 92
- `custom_hyperparameters` (*evalml.pipelines.BaselineRegressionPipeline attribute*), 103
- `custom_hyperparameters` (*evalml.pipelines.MeanBaselineRegressionPipeline attribute*), 107
- `custom_hyperparameters` (*evalml.pipelines.ModeBaselineBinaryPipeline attribute*), 96
- `custom_hyperparameters` (*evalml.pipelines.ModeBaselineMulticlassPipeline attribute*), 99
- `custom_name` (*evalml.pipelines.BaselineBinaryPipeline attribute*), 88
- `custom_name` (*evalml.pipelines.BaselineMulticlassPipeline attribute*), 92
- `custom_name` (*evalml.pipelines.BaselineRegressionPipeline attribute*), 103
- `custom_name` (*evalml.pipelines.MeanBaselineRegressionPipeline attribute*), 107
- `custom_name` (*evalml.pipelines.ModeBaselineBinaryPipeline attribute*), 95
- `custom_name` (*evalml.pipelines.ModeBaselineMulticlassPipeline attribute*), 99
- D**
- `DataCheck` (class in *evalml.data\_checks*), 272
- `DataCheckError` (class in *evalml.data\_checks*), 283
- `DataCheckMessage` (class in *evalml.data\_checks*), 282
- `DataCheckMessageType` (class in *evalml.data\_checks*), 285
- `DataChecks` (class in *evalml.data\_checks*), 280
- `DataCheckWarning` (class in *evalml.data\_checks*), 284
- `DateTimeFeaturizer` (class in *evalml.pipelines.components*), 145
- `decision_function()` (*evalml.objectives.AccuracyBinary method*), 210
- `decision_function()` (*evalml.objectives.AUC method*), 213
- `decision_function()` (*evalml.objectives.BalancedAccuracyBinary method*), 220
- `decision_function()` (*evalml.objectives.BinaryClassificationObjective method*), 196
- `decision_function()` (*evalml.objectives.CostBenefitMatrix method*), 207
- `decision_function()` (*evalml.objectives.F1 method*), 224
- `decision_function()` (*evalml.objectives.FraudCost method*), 202
- `decision_function()` (*evalml.objectives.LeadScoring method*), 204
- `decision_function()` (*evalml.objectives.LogLossBinary method*), 231
- `decision_function()` (*evalml.objectives.MCCBinary method*), 234
- `decision_function()` (*evalml.objectives.Precision method*), 238
- `decision_function()` (*evalml.objectives.Recall method*), 245
- `default_parameters` (*evalml.pipelines.BaselineBinaryPipeline attribute*), 88
- `default_parameters` (*evalml.pipelines.BaselineMulticlassPipeline attribute*), 92
- `default_parameters` (*evalml.pipelines.BaselineRegressionPipeline attribute*), 103
- `default_parameters` (*evalml.pipelines.components.BaselineClassifier attribute*), 166
- `default_parameters` (*evalml.pipelines.components.BaselineRegressor attribute*), 184
- `default_parameters` (*evalml.pipelines.components.CatBoostClassifier attribute*), 151
- `default_parameters` (*evalml.pipelines.components.CatBoostRegressor attribute*), 169
- `default_parameters` (*evalml.pipelines.components.DateTimeFeaturizer attribute*), 145
- `default_parameters` (*evalml.pipelines.components.DropColumns attribute*), 119
- `default_parameters` (*evalml.pipelines.components.DropNullColumns attribute*), 143
- `default_parameters` (*evalml.pipelines.components.ElasticNetClassifier attribute*), 153
- `default_parameters` (*evalml.pipelines.components.ElasticNetRegressor attribute*), 171
- `default_parameters` (*evalml.pipelines.components.ExtraTreesClassifier attribute*), 156

default_parameters ( <i>evalml.pipelines.components.ExtraTreesRegressor</i> attribute), 176	default_parameters ( <i>evalml.pipelines.ModeBaselineMulticlassPipeline</i> attribute), 99
default_parameters ( <i>evalml.pipelines.components.Imputer</i> at- tribute), 129	DefaultDataChecks (class in <i>evalml.data_checks</i> ), 281
default_parameters ( <i>evalml.pipelines.components.LinearRegressor</i> attribute), 174	describe() ( <i>evalml.pipelines.BaselineBinaryPipeline</i> method), 89
default_parameters ( <i>evalml.pipelines.components.LogisticRegressionClassifier</i> attribute), 161	describe() ( <i>evalml.pipelines.BaselineMulticlassPipeline</i> method), 93
default_parameters ( <i>evalml.pipelines.components.OneHotEncoder</i> attribute), 124	describe() ( <i>evalml.pipelines.BaselineRegressionPipeline</i> method), 104
default_parameters ( <i>evalml.pipelines.components.PerColumnImputer</i> attribute), 127	describe() ( <i>evalml.pipelines.BinaryClassificationPipeline</i> method), 79
default_parameters ( <i>evalml.pipelines.components.RandomForestClassifier</i> attribute), 158	describe() ( <i>evalml.pipelines.ClassificationPipeline</i> method), 75
default_parameters ( <i>evalml.pipelines.components.RandomForestRegressor</i> attribute), 179	describe() ( <i>evalml.pipelines.components.BaselineClassifier</i> method), 167
default_parameters ( <i>evalml.pipelines.components.RFClassifierSelectFromModel</i> attribute), 140	describe() ( <i>evalml.pipelines.components.BaselineRegressor</i> method), 185
default_parameters ( <i>evalml.pipelines.components.RFRegressorSelectFromModel</i> attribute), 137	describe() ( <i>evalml.pipelines.components.CatBoostClassifier</i> method), 152
default_parameters ( <i>evalml.pipelines.components.SelectColumns</i> attribute), 121	describe() ( <i>evalml.pipelines.components.CatBoostRegressor</i> method), 170
default_parameters ( <i>evalml.pipelines.components.SimpleImputer</i> attribute), 132	describe() ( <i>evalml.pipelines.components.ComponentBase</i> method), 112
default_parameters ( <i>evalml.pipelines.components.StandardScaler</i> attribute), 135	describe() ( <i>evalml.pipelines.components.DateTimeFeaturizer</i> method), 146
default_parameters ( <i>evalml.pipelines.components.TextFeaturizer</i> attribute), 148	describe() ( <i>evalml.pipelines.components.DropColumns</i> method), 119
default_parameters ( <i>evalml.pipelines.components.XGBoostClassifier</i> attribute), 163	describe() ( <i>evalml.pipelines.components.DropNullColumns</i> method), 144
default_parameters ( <i>evalml.pipelines.components.XGBoostRegressor</i> attribute), 181	describe() ( <i>evalml.pipelines.components.ElasticNetClassifier</i> method), 154
default_parameters ( <i>evalml.pipelines.MeanBaselineRegressionPipeline</i> attribute), 107	describe() ( <i>evalml.pipelines.components.ElasticNetRegressor</i> method), 172
default_parameters ( <i>evalml.pipelines.ModeBaselineBinaryPipeline</i> attribute), 96	describe() ( <i>evalml.pipelines.components.Estimator</i> method), 116
	describe() ( <i>evalml.pipelines.components.ExtraTreesClassifier</i> method), 157
	describe() ( <i>evalml.pipelines.components.ExtraTreesRegressor</i> method), 177
	describe() ( <i>evalml.pipelines.components.Imputer</i> method), 131
	describe() ( <i>evalml.pipelines.components.LinearRegressor</i> method), 175
	describe() ( <i>evalml.pipelines.components.LogisticRegressionClassifier</i> method), 162
	describe() ( <i>evalml.pipelines.components.OneHotEncoder</i> method), 125
	describe() ( <i>evalml.pipelines.components.PerColumnImputer</i> method), 128
	describe() ( <i>evalml.pipelines.components.RandomForestClassifier</i> method), 159
	describe() ( <i>evalml.pipelines.components.RandomForestRegressor</i> method), 179

method), 179

describe() (evalml.pipelines.components.RFClassifierSelectFromModel method), 141

describe() (evalml.pipelines.components.RFRegressorSelectFromModel method), 138

describe() (evalml.pipelines.components.SelectColumns method), 122

describe() (evalml.pipelines.components.SimpleImputer method), 133

describe() (evalml.pipelines.components.StandardScaler method), 136

describe() (evalml.pipelines.components.TextFeaturizer method), 149

describe() (evalml.pipelines.components.Transformer method), 114

describe() (evalml.pipelines.components.XGBoostClassifier method), 164

describe() (evalml.pipelines.components.XGBoostRegressor method), 182

describe() (evalml.pipelines.MeanBaselineRegressionPipeline method), 108

describe() (evalml.pipelines.ModeBaselineBinaryPipeline method), 97

describe() (evalml.pipelines.ModeBaselineMulticlassPipeline method), 101

describe() (evalml.pipelines.MulticlassClassificationPipeline method), 82

describe() (evalml.pipelines.PipelineBase method), 72

describe() (evalml.pipelines.RegressionPipeline method), 86

describe\_pipeline() (evalml.automl.AutoMLSearch method), 66

drop\_nan\_target\_rows() (in module evalml.preprocessing), 62

DropColumns (class in evalml.pipelines.components), 118

DropNullColumns (class in evalml.pipelines.components), 143

**E**

ElasticNetClassifier (class in evalml.pipelines.components), 153

ElasticNetRegressor (class in evalml.pipelines.components), 171

Estimator (class in evalml.pipelines.components), 115

explain\_prediction() (in module evalml.model\_understanding.prediction\_explanations), 191

explain\_predictions() (in module evalml.model\_understanding.prediction\_explanations), 192

explain\_predictions\_best\_worst() (in module evalml.model\_understanding.prediction\_explanations), 192

ExtraTreesClassifier (class in evalml.pipelines.components), 156

ExtraTreesRegressor (class in evalml.pipelines.components), 176

**F**

F1 (class in evalml.objectives), 223

F1Macro (class in evalml.objectives), 227

F1Micro (class in evalml.objectives), 225

F1Weighted (class in evalml.objectives), 228

fit() (evalml.pipelines.BaselineBinaryPipeline method), 90

fit() (evalml.pipelines.BaselineMulticlassPipeline method), 93

fit() (evalml.pipelines.BaselineRegressionPipeline method), 105

fit() (evalml.pipelines.BinaryClassificationPipeline method), 79

fit() (evalml.pipelines.ClassificationPipeline method), 76

fit() (evalml.pipelines.components.BaselineClassifier method), 167

fit() (evalml.pipelines.components.BaselineRegressor method), 185

fit() (evalml.pipelines.components.CatBoostClassifier method), 152

fit() (evalml.pipelines.components.CatBoostRegressor method), 170

fit() (evalml.pipelines.components.ComponentBase method), 112

fit() (evalml.pipelines.components.DateTimeFeaturizer method), 146

fit() (evalml.pipelines.components.DropColumns method), 120

fit() (evalml.pipelines.components.DropNullColumns method), 144

fit() (evalml.pipelines.components.ElasticNetClassifier method), 154

fit() (evalml.pipelines.components.ElasticNetRegressor method), 172

fit() (evalml.pipelines.components.Estimator method), 116

fit() (evalml.pipelines.components.ExtraTreesClassifier method), 157

fit() (evalml.pipelines.components.ExtraTreesRegressor method), 177

fit() (evalml.pipelines.components.Imputer method), 131

fit() (evalml.pipelines.components.LinearRegressor method), 175

[fit\(\) \(evalml.pipelines.components.LogisticRegressionClassifier method\), 162](#)  
[fit\(\) \(evalml.pipelines.components.OneHotEncoder method\), 125](#)  
[fit\(\) \(evalml.pipelines.components.PerColumnImputer method\), 128](#)  
[fit\(\) \(evalml.pipelines.components.RandomForestClassifier method\), 159](#)  
[fit\(\) \(evalml.pipelines.components.RandomForestRegressor method\), 180](#)  
[fit\(\) \(evalml.pipelines.components.RFClassifierSelectFromModel method\), 141](#)  
[fit\(\) \(evalml.pipelines.components.RFRegressorSelectFromModel method\), 139](#)  
[fit\(\) \(evalml.pipelines.components.SelectColumns method\), 122](#)  
[fit\(\) \(evalml.pipelines.components.SimpleImputer method\), 134](#)  
[fit\(\) \(evalml.pipelines.components.StandardScaler method\), 136](#)  
[fit\(\) \(evalml.pipelines.components.TextFeaturizer method\), 149](#)  
[fit\(\) \(evalml.pipelines.components.Transformer method\), 114](#)  
[fit\(\) \(evalml.pipelines.components.XGBoostClassifier method\), 164](#)  
[fit\(\) \(evalml.pipelines.components.XGBoostRegressor method\), 182](#)  
[fit\(\) \(evalml.pipelines.MeanBaselineRegressionPipeline method\), 108](#)  
[fit\(\) \(evalml.pipelines.ModeBaselineBinaryPipeline method\), 97](#)  
[fit\(\) \(evalml.pipelines.ModeBaselineMulticlassPipeline method\), 101](#)  
[fit\(\) \(evalml.pipelines.MulticlassClassificationPipeline method\), 82](#)  
[fit\(\) \(evalml.pipelines.PipelineBase method\), 72](#)  
[fit\(\) \(evalml.pipelines.RegressionPipeline method\), 86](#)  
[fit\\_transform\(\) \(evalml.pipelines.components.DateTimeFeaturizer method\), 147](#)  
[fit\\_transform\(\) \(evalml.pipelines.components.DropColumns method\), 120](#)  
[fit\\_transform\(\) \(evalml.pipelines.components.DropNullColumns method\), 144](#)  
[fit\\_transform\(\) \(evalml.pipelines.components.Imputer method\), 131](#)  
[fit\\_transform\(\) \(evalml.pipelines.components.OneHotEncoder method\), 125](#)  
[fit\\_transform\(\) \(evalml.pipelines.components.PerColumnImputer method\), 128](#)  
[fit\\_transform\(\) \(evalml.pipelines.components.RFClassifierSelectFromModel method\), 142](#)  
[fit\\_transform\(\) \(evalml.pipelines.components.RFRegressorSelectFromModel method\), 139](#)  
[fit\\_transform\(\) \(evalml.pipelines.components.SelectColumns method\), 122](#)  
[fit\\_transform\(\) \(evalml.pipelines.components.SimpleImputer method\), 134](#)  
[fit\\_transform\(\) \(evalml.pipelines.components.StandardScaler method\), 136](#)  
[fit\\_transform\(\) \(evalml.pipelines.components.TextFeaturizer method\), 149](#)  
[fit\\_transform\(\) \(evalml.pipelines.components.Transformer method\), 114](#)  
[fit\\_transform\(\) \(evalml.pipelines.components.XGBoostClassifier method\), 164](#)  
[fit\\_transform\(\) \(evalml.pipelines.components.XGBoostRegressor method\), 182](#)  
[fit\\_transform\(\) \(evalml.pipelines.MeanBaselineRegressionPipeline method\), 108](#)  
[fit\\_transform\(\) \(evalml.pipelines.ModeBaselineBinaryPipeline method\), 97](#)  
[fit\\_transform\(\) \(evalml.pipelines.ModeBaselineMulticlassPipeline method\), 101](#)  
[fit\\_transform\(\) \(evalml.pipelines.MulticlassClassificationPipeline method\), 82](#)  
[fit\\_transform\(\) \(evalml.pipelines.PipelineBase method\), 72](#)  
[fit\\_transform\(\) \(evalml.pipelines.RegressionPipeline method\), 86](#)  
[get\\_estimator\\_names\(\) \(evalml.pipelines.components.DateFeaturizer method\), 126](#)  
[get\\_estimator\\_names\(\) \(evalml.pipelines.components.OneHotEncoder method\), 126](#)  
[get\\_estimator\\_names\(\) \(evalml.pipelines.components.RFClassifierSelectFromModel method\), 142](#)  
[get\\_estimator\\_names\(\) \(evalml.pipelines.components.RFRegressorSelectFromModel method\), 139](#)  
[get\\_pipeline\(\) \(evalml.automl.AutoMLSearch method\), 66](#)  
[get\\_random\\_seed\(\) \(in module evalml.utils\), 286](#)  
[get\\_random\\_state\(\) \(in module evalml.utils\), 285](#)  
[graph\(\) \(evalml.pipelines.BaselineBinaryPipeline method\), 90](#)  
[graph\(\) \(evalml.pipelines.BaselineMulticlassPipeline method\), 94](#)  
[graph\(\) \(evalml.pipelines.BaselineRegressionPipeline method\), 105](#)  
[graph\(\) \(evalml.pipelines.BinaryClassificationPipeline method\), 79](#)  
[graph\(\) \(evalml.pipelines.ClassificationPipeline method\), 76](#)  
[graph\(\) \(evalml.pipelines.MeanBaselineRegressionPipeline method\), 109](#)  
[graph\(\) \(evalml.pipelines.ModeBaselineBinaryPipeline method\), 97](#)  
[graph\(\) \(evalml.pipelines.ModeBaselineMulticlassPipeline method\), 101](#)  
[graph\(\) \(evalml.pipelines.MulticlassClassificationPipeline method\), 83](#)  
[graph\(\) \(evalml.pipelines.PipelineBase method\), 73](#)  
[graph\(\) \(evalml.pipelines.RegressionPipeline method\), 86](#)  
[get\\_estimators\(\) \(in module evalml.pipelines.components.utils\), 118](#)  
[get\\_feature\\_names\(\) \(evalml.pipelines.components.OneHotEncoder method\), 126](#)  
[get\\_feature\\_names\(\) \(evalml.pipelines.components.RFClassifierSelectFromModel method\), 142](#)  
[get\\_feature\\_names\(\) \(evalml.pipelines.components.RFRegressorSelectFromModel method\), 139](#)  
[get\\_pipeline\(\) \(evalml.automl.AutoMLSearch method\), 66](#)  
[get\\_random\\_seed\(\) \(in module evalml.utils\), 286](#)  
[get\\_random\\_state\(\) \(in module evalml.utils\), 285](#)  
[graph\(\) \(evalml.pipelines.BaselineBinaryPipeline method\), 90](#)  
[graph\(\) \(evalml.pipelines.BaselineMulticlassPipeline method\), 94](#)  
[graph\(\) \(evalml.pipelines.BaselineRegressionPipeline method\), 105](#)

- [graph\(\)](#) (*evalml.pipelines.BinaryClassificationPipeline* method), 79  
[graph\(\)](#) (*evalml.pipelines.ClassificationPipeline* method), 76  
[graph\(\)](#) (*evalml.pipelines.MeanBaselineRegressionPipeline* method), 109  
[graph\(\)](#) (*evalml.pipelines.ModeBaselineBinaryPipeline* method), 97  
[graph\(\)](#) (*evalml.pipelines.ModeBaselineMulticlassPipeline* method), 101  
[graph\(\)](#) (*evalml.pipelines.MulticlassClassificationPipeline* method), 83  
[graph\(\)](#) (*evalml.pipelines.PipelineBase* method), 73  
[graph\(\)](#) (*evalml.pipelines.RegressionPipeline* method), 86  
[graph\\_binary\\_objective\\_vs\\_threshold\(\)](#) (in module *evalml.model\_understanding*), 190  
[graph\\_confusion\\_matrix\(\)](#) (in module *evalml.model\_understanding*), 189  
[graph\\_feature\\_importance\(\)](#) (*evalml.pipelines.BaselineBinaryPipeline* method), 90  
[graph\\_feature\\_importance\(\)](#) (*evalml.pipelines.BaselineMulticlassPipeline* method), 94  
[graph\\_feature\\_importance\(\)](#) (*evalml.pipelines.BaselineRegressionPipeline* method), 105  
[graph\\_feature\\_importance\(\)](#) (*evalml.pipelines.BinaryClassificationPipeline* method), 80  
[graph\\_feature\\_importance\(\)](#) (*evalml.pipelines.ClassificationPipeline* method), 76  
[graph\\_feature\\_importance\(\)](#) (*evalml.pipelines.MeanBaselineRegressionPipeline* method), 109  
[graph\\_feature\\_importance\(\)](#) (*evalml.pipelines.ModeBaselineBinaryPipeline* method), 98  
[graph\\_feature\\_importance\(\)](#) (*evalml.pipelines.ModeBaselineMulticlassPipeline* method), 101  
[graph\\_feature\\_importance\(\)](#) (*evalml.pipelines.MulticlassClassificationPipeline* method), 83  
[graph\\_feature\\_importance\(\)](#) (*evalml.pipelines.PipelineBase* method), 73  
[graph\\_feature\\_importance\(\)](#) (*evalml.pipelines.RegressionPipeline* method), 87  
[graph\\_permutation\\_importance\(\)](#) (in module *evalml.model\_understanding*), 189  
[graph\\_precision\\_recall\\_curve\(\)](#) (in module *evalml.model\_understanding*), 187  
[graph\\_roc\\_curve\(\)](#) (in module *evalml.model\_understanding*), 188  
[GridSearchTuner](#) (class in *evalml.tuners*), 269
- ## H
- [handle\\_model\\_family\(\)](#) (in module *evalml.model\_family*), 266  
[handle\\_problem\\_types\(\)](#) (in module *evalml.problem\_types*), 265  
[HighlyNullDataCheck](#) (class in *evalml.data\_checks*), 274  
[hyperparameter\\_ranges](#) (*evalml.pipelines.components.BaselineClassifier* attribute), 166  
[hyperparameter\\_ranges](#) (*evalml.pipelines.components.BaselineRegressor* attribute), 184  
[hyperparameter\\_ranges](#) (*evalml.pipelines.components.CatBoostClassifier* attribute), 151  
[hyperparameter\\_ranges](#) (*evalml.pipelines.components.CatBoostRegressor* attribute), 169  
[hyperparameter\\_ranges](#) (*evalml.pipelines.components.DateTimeFeaturizer* attribute), 145  
[hyperparameter\\_ranges](#) (*evalml.pipelines.components.DropColumns* attribute), 119  
[hyperparameter\\_ranges](#) (*evalml.pipelines.components.DropNullColumns* attribute), 143  
[hyperparameter\\_ranges](#) (*evalml.pipelines.components.ElasticNetClassifier* attribute), 153  
[hyperparameter\\_ranges](#) (*evalml.pipelines.components.ElasticNetRegressor* attribute), 171  
[hyperparameter\\_ranges](#) (*evalml.pipelines.components.ExtraTreesClassifier* attribute), 156  
[hyperparameter\\_ranges](#) (*evalml.pipelines.components.ExtraTreesRegressor* attribute), 176  
[hyperparameter\\_ranges](#) (*evalml.pipelines.components.Imputer* attribute), 129  
[hyperparameter\\_ranges](#) (*evalml.pipelines.components.LinearRegressor* attribute), 174  
[hyperparameter\\_ranges](#) (*evalml.pipelines.components.LogisticRegressionClassifier*



[attribute](#)), 161  
[hyperparameter\\_ranges](#) ([evalml.pipelines.components.OneHotEncoder](#) [attribute](#)), 124  
[hyperparameter\\_ranges](#) ([evalml.pipelines.components.PerColumnImputer](#) [attribute](#)), 127  
[hyperparameter\\_ranges](#) ([evalml.pipelines.components.RandomForestClassifier](#) [attribute](#)), 158  
[hyperparameter\\_ranges](#) ([evalml.pipelines.components.RandomForestRegressor](#) [attribute](#)), 179  
[hyperparameter\\_ranges](#) ([evalml.pipelines.components.RFClassifierSelectFromModel](#) [attribute](#)), 140  
[hyperparameter\\_ranges](#) ([evalml.pipelines.components.RFRegressorSelectFromModel](#) [attribute](#)), 137  
[hyperparameter\\_ranges](#) ([evalml.pipelines.components.SelectColumns](#) [attribute](#)), 121  
[hyperparameter\\_ranges](#) ([evalml.pipelines.components.SimpleImputer](#) [attribute](#)), 132  
[hyperparameter\\_ranges](#) ([evalml.pipelines.components.StandardScaler](#) [attribute](#)), 135  
[hyperparameter\\_ranges](#) ([evalml.pipelines.components.TextFeaturizer](#) [attribute](#)), 148  
[hyperparameter\\_ranges](#) ([evalml.pipelines.components.XGBoostClassifier](#) [attribute](#)), 163  
[hyperparameter\\_ranges](#) ([evalml.pipelines.components.XGBoostRegressor](#) [attribute](#)), 181  
[hyperparameters](#) ([evalml.pipelines.BaselineBinaryPipeline](#) [attribute](#)), 88  
[hyperparameters](#) ([evalml.pipelines.BaselineMulticlassPipeline](#) [attribute](#)), 92  
[hyperparameters](#) ([evalml.pipelines.BaselineRegressionPipeline](#) [attribute](#)), 103  
[hyperparameters](#) ([evalml.pipelines.MeanBaselineRegressionPipeline](#) [attribute](#)), 107  
[hyperparameters](#) ([evalml.pipelines.ModeBaselineBinaryPipeline](#) [attribute](#)), 96  
[hyperparameters](#) ([evalml.pipelines.ModeBaselineMulticlassPipeline](#) [attribute](#)), 99  
  
**I**  
[IDColumnsDataCheck](#) (class in [evalml.data\\_checks](#)), 275  
[import\\_or\\_raise\(\)](#) (in module [evalml.utils](#)), 285  
  
[Imputer](#) (class in [evalml.pipelines.components](#)), 129  
[InvalidTargetDataCheck](#) (class in [evalml.data\\_checks](#)), 273  
[is\\_search\\_space\\_exhausted\(\)](#) ([evalml.tuners.GridSearchTuner](#) method), 270  
[is\\_search\\_space\\_exhausted\(\)](#) ([evalml.tuners.RandomSearchTuner](#) method), 272  
[is\\_search\\_space\\_exhausted\(\)](#) ([evalml.tuners.SKOptTuner](#) method), 268  
[is\\_search\\_space\\_exhausted\(\)](#) ([evalml.tuners.Tuner](#) method), 267  
  
[IterativeAlgorithm](#) (class in [evalml.automl.automl\\_algorithm](#)), 69  
  
**L**  
[LaplaceDistribution\(\)](#) (in module [evalml.preprocessing](#)), 62  
[LabelLeakageDataCheck](#) (class in [evalml.data\\_checks](#)), 276  
[LeadScoring](#) (class in [evalml.objectives](#)), 203  
[LinearRegressor](#) (class in [evalml.pipelines.components](#)), 173  
[load\(\)](#) ([evalml.automl.AutoMLSearch](#) static method), 66  
[load\(\)](#) ([evalml.pipelines.BaselineBinaryPipeline](#) static method), 90  
[load\(\)](#) ([evalml.pipelines.BaselineMulticlassPipeline](#) static method), 94  
[load\(\)](#) ([evalml.pipelines.BaselineRegressionPipeline](#) static method), 106  
[load\(\)](#) ([evalml.pipelines.BinaryClassificationPipeline](#) static method), 80  
[load\(\)](#) ([evalml.pipelines.ClassificationPipeline](#) static method), 77  
[load\(\)](#) ([evalml.pipelines.components.BaselineClassifier](#) static method), 167  
[load\(\)](#) ([evalml.pipelines.components.BaselineRegressor](#) static method), 185  
[load\(\)](#) ([evalml.pipelines.components.CatBoostClassifier](#) static method), 152  
[load\(\)](#) ([evalml.pipelines.components.CatBoostRegressor](#) static method), 170  
[load\(\)](#) ([evalml.pipelines.components.ComponentBase](#) static method), 112  
[load\(\)](#) ([evalml.pipelines.components.DateFeaturizer](#) static method), 147  
[load\(\)](#) ([evalml.pipelines.components.DropColumns](#) static method), 120  
[load\(\)](#) ([evalml.pipelines.components.DropNullColumns](#) static method), 144  
[load\(\)](#) ([evalml.pipelines.components.ElasticNetClassifier](#) static method), 155

<code>load()</code> ( <code>evalml.pipelines.components.ElasticNetRegressor</code> static method), 173	<code>load_data()</code> (in module <code>evalml.preprocessing</code> ), 62
<code>load()</code> ( <code>evalml.pipelines.components.Estimator</code> static method), 116	<code>load_diabetes()</code> (in module <code>evalml.demos</code> ), 62
<code>load()</code> ( <code>evalml.pipelines.components.ExtraTreesClassifier</code> static method), 157	<code>load_fraud()</code> (in module <code>evalml.demos</code> ), 61
<code>load()</code> ( <code>evalml.pipelines.components.ExtraTreesRegressor</code> static method), 178	<code>load_wine()</code> (in module <code>evalml.demos</code> ), 61
<code>load()</code> ( <code>evalml.pipelines.components.Imputer</code> static method), 131	<code>LogisticRegressionClassifier</code> (class in <code>evalml.pipelines.components</code> ), 161
<code>load()</code> ( <code>evalml.pipelines.components.LinearRegressor</code> static method), 175	<code>LogLossBinary</code> (class in <code>evalml.objectives</code> ), 230
<code>load()</code> ( <code>evalml.pipelines.components.LogisticRegressionClassifier</code> static method), 162	<code>LogLossMulticlass</code> (class in <code>evalml.objectives</code> ), 232
<code>load()</code> ( <code>evalml.pipelines.components.OneHotEncoder</code> static method), 126	<b>M</b>
<code>load()</code> ( <code>evalml.pipelines.components.PerColumnImputer</code> static method), 129	<code>MAE</code> (class in <code>evalml.objectives</code> ), 252
<code>load()</code> ( <code>evalml.pipelines.components.RandomForestClassifier</code> static method), 160	<code>make_pipeline()</code> (in module <code>evalml.pipelines.utils</code> ), 110
<code>load()</code> ( <code>evalml.pipelines.components.RandomForestRegressor</code> static method), 180	<code>MaxError</code> (class in <code>evalml.objectives</code> ), 258
<code>load()</code> ( <code>evalml.pipelines.components.RFClassifierSelectFromModel</code> static method), 142	<code>MCCBinary</code> (class in <code>evalml.objectives</code> ), 233
<code>load()</code> ( <code>evalml.pipelines.components.RFRegressorSelectFromModel</code> static method), 139	<code>MCCMulticlass</code> (class in <code>evalml.objectives</code> ), 236
<code>load()</code> ( <code>evalml.pipelines.components.SelectColumns</code> static method), 123	<code>MeanBaselineRegressionPipeline</code> (class in <code>evalml.pipelines</code> ), 107
<code>load()</code> ( <code>evalml.pipelines.components.SimpleImputer</code> static method), 134	<code>MeanSquaredLogError</code> (class in <code>evalml.objectives</code> ), 255
<code>load()</code> ( <code>evalml.pipelines.components.StandardScaler</code> static method), 136	<code>MedianAE</code> (class in <code>evalml.objectives</code> ), 257
<code>load()</code> ( <code>evalml.pipelines.components.TextFeaturizer</code> static method), 150	<code>model_type</code> ( <code>evalml.data_checks.DataCheckError</code> attribute), 283
<code>load()</code> ( <code>evalml.pipelines.components.Transformer</code> static method), 114	<code>model_type</code> ( <code>evalml.data_checks.DataCheckMessage</code> attribute), 282
<code>load()</code> ( <code>evalml.pipelines.components.XGBoostClassifier</code> static method), 165	<code>message_type</code> ( <code>evalml.data_checks.DataCheckWarning</code> attribute), 284
<code>load()</code> ( <code>evalml.pipelines.components.XGBoostRegressor</code> static method), 183	<code>ModeBaselineBinaryPipeline</code> (class in <code>evalml.pipelines</code> ), 95
<code>load()</code> ( <code>evalml.pipelines.MeanBaselineRegressionPipeline</code> static method), 109	<code>ModeBaselineMulticlassPipeline</code> (class in <code>evalml.pipelines</code> ), 99
<code>load()</code> ( <code>evalml.pipelines.ModeBaselineBinaryPipeline</code> static method), 98	<code>model_family</code> ( <code>evalml.pipelines.BaselineBinaryPipeline</code> attribute), 88
<code>load()</code> ( <code>evalml.pipelines.ModeBaselineMulticlassPipeline</code> static method), 102	<code>model_family</code> ( <code>evalml.pipelines.BaselineMulticlassPipeline</code> attribute), 92
<code>load()</code> ( <code>evalml.pipelines.MulticlassClassificationPipeline</code> static method), 83	<code>model_family</code> ( <code>evalml.pipelines.BaselineRegressionPipeline</code> attribute), 103
<code>load()</code> ( <code>evalml.pipelines.PipelineBase</code> static method), 73	<code>model_family</code> ( <code>evalml.pipelines.components.BaselineClassifier</code> attribute), 166
<code>load()</code> ( <code>evalml.pipelines.RegressionPipeline</code> static method), 87	<code>model_family</code> ( <code>evalml.pipelines.components.BaselineRegressor</code> attribute), 184
<code>load_breast_cancer()</code> (in module <code>evalml.demos</code> ), 61	<code>model_family</code> ( <code>evalml.pipelines.components.CatBoostClassifier</code> attribute), 151
	<code>model_family</code> ( <code>evalml.pipelines.components.CatBoostRegressor</code> attribute), 169
	<code>model_family</code> ( <code>evalml.pipelines.components.DateTimeFeaturizer</code> attribute), 145
	<code>model_family</code> ( <code>evalml.pipelines.components.DropColumns</code> attribute), 119
	<code>model_family</code> ( <code>evalml.pipelines.components.DropNullColumns</code> attribute), 143
	<code>model_family</code> ( <code>evalml.pipelines.components.ElasticNetClassifier</code> attribute), 153

`model_family` (`evalml.pipelines.components.ElasticNetRegressor` attribute), 171  
`model_family` (`evalml.pipelines.components.ExtraTreesClassifier` attribute), 156  
`model_family` (`evalml.pipelines.components.ExtraTreesRegressor` attribute), 176  
`model_family` (`evalml.pipelines.components.Imputer` name attribute), 129  
`model_family` (`evalml.pipelines.components.LinearRegressor` name attribute), 174  
`model_family` (`evalml.pipelines.components.LogisticRegressionClassifier` attribute), 161  
`model_family` (`evalml.pipelines.components.OneHotEncoder` name attribute), 123  
`model_family` (`evalml.pipelines.components.PerColumnImputer` attribute), 127  
`model_family` (`evalml.pipelines.components.RandomForestClassifier` attribute), 158  
`model_family` (`evalml.pipelines.components.RandomForestRegressor` attribute), 179  
`model_family` (`evalml.pipelines.components.RFClassifierSelectFromModel` attribute), 140  
`model_family` (`evalml.pipelines.components.RFRegressorSelectFromModel` attribute), 137  
`model_family` (`evalml.pipelines.components.SelectColumnTransformer` name attribute), 121  
`model_family` (`evalml.pipelines.components.SimpleImputer` name attribute), 132  
`model_family` (`evalml.pipelines.components.StandardScaler` name attribute), 135  
`model_family` (`evalml.pipelines.components.TextFeaturizer` name attribute), 148  
`model_family` (`evalml.pipelines.components.XGBoostClassifier` attribute), 163  
`model_family` (`evalml.pipelines.components.XGBoostRegressor` attribute), 181  
`model_family` (`evalml.pipelines.MeanBaselineRegressionPipeline` attribute), 107  
`model_family` (`evalml.pipelines.ModeBaselineBinaryPipeline` attribute), 95  
`model_family` (`evalml.pipelines.ModeBaselineMulticlassPipeline` attribute), 99  
`ModelFamily` (class in `evalml.model_family`), 265  
`MSE` (class in `evalml.objectives`), 254  
`MulticlassClassificationObjective` (class in `evalml.objectives`), 197  
`MulticlassClassificationPipeline` (class in `evalml.pipelines`), 81  
  
**N**  
`name` (`evalml.data_checks.DataCheck` attribute), 273  
`name` (`evalml.data_checks.HighlyNullDataCheck` attribute), 274  
`name` (`evalml.data_checks.IDColumnsDataCheck` attribute), 275  
`name` (`evalml.data_checks.InvalidTargetDataCheck` attribute), 273  
`name` (`evalml.data_checks.LabelLeakageDataCheck` attribute), 277  
`name` (`evalml.data_checks.NoVarianceDataCheck` attribute), 279  
`name` (`evalml.data_checks.OutliersDataCheck` attribute), 278  
`name` (`evalml.pipelines.BaselineBinaryPipeline` attribute), 88  
`name` (`evalml.pipelines.BaselineMulticlassPipeline` attribute), 92  
`name` (`evalml.pipelines.BaselineRegressionPipeline` attribute), 103  
`name` (`evalml.pipelines.components.BaselineClassifier` attribute), 166  
`name` (`evalml.pipelines.components.BaselineRegressor` attribute), 183  
`name` (`evalml.pipelines.components.CatBoostClassifier` attribute), 151  
`name` (`evalml.pipelines.components.CatBoostRegressor` attribute), 169  
`name` (`evalml.pipelines.components.DateTimeFeaturizer` attribute), 145  
`name` (`evalml.pipelines.components.DropColumns` attribute), 118  
`name` (`evalml.pipelines.components.DropNullColumns` attribute), 143  
`name` (`evalml.pipelines.components.ElasticNetClassifier` attribute), 153  
`name` (`evalml.pipelines.components.ElasticNetRegressor` attribute), 171  
`name` (`evalml.pipelines.components.ExtraTreesClassifier` attribute), 156  
`name` (`evalml.pipelines.components.ExtraTreesRegressor` attribute), 176  
`name` (`evalml.pipelines.components.Imputer` attribute), 129  
`name` (`evalml.pipelines.components.LinearRegressor` attribute), 174  
`name` (`evalml.pipelines.components.LogisticRegressionClassifier` attribute), 161  
`name` (`evalml.pipelines.components.OneHotEncoder` attribute), 123  
`name` (`evalml.pipelines.components.PerColumnImputer` attribute), 126  
`name` (`evalml.pipelines.components.RandomForestClassifier` attribute), 158  
`name` (`evalml.pipelines.components.RandomForestRegressor` attribute), 178  
`name` (`evalml.pipelines.components.RFClassifierSelectFromModel` attribute), 140



name (*evalml.pipelines.components.RFRegressorSelectFromModel* attribute), 137  
 name (*evalml.pipelines.components.SelectColumns* attribute), 121  
 name (*evalml.pipelines.components.SimpleImputer* attribute), 132  
 name (*evalml.pipelines.components.StandardScaler* attribute), 135  
 name (*evalml.pipelines.components.TextFeaturizer* attribute), 148  
 name (*evalml.pipelines.components.XGBoostClassifier* attribute), 163  
 name (*evalml.pipelines.components.XGBoostRegressor* attribute), 181  
 name (*evalml.pipelines.MeanBaselineRegressionPipeline* attribute), 107  
 name (*evalml.pipelines.ModeBaselineBinaryPipeline* attribute), 95  
 name (*evalml.pipelines.ModeBaselineMulticlassPipeline* attribute), 99  
 next\_batch() (*evalml.automl.automl\_algorithm.AutoMLAlgorithm* method), 69  
 next\_batch() (*evalml.automl.automl\_algorithm.IterativeAlgorithm* method), 70  
 normalize\_confusion\_matrix() (in module *evalml.model\_understanding*), 187  
 NoVarianceDataCheck (class in *evalml.data\_checks*), 279  
 number\_of\_features() (in module *evalml.preprocessing*), 63

**O**

objective\_function() (*evalml.objectives.AccuracyBinary* method), 210  
 objective\_function() (*evalml.objectives.AccuracyMulticlass* method), 212  
 objective\_function() (*evalml.objectives.AUC* method), 214  
 objective\_function() (*evalml.objectives.AUCMacro* method), 216  
 objective\_function() (*evalml.objectives.AUCMicro* method), 217  
 objective\_function() (*evalml.objectives.AUCWeighted* method), 219  
 objective\_function() (*evalml.objectives.BalancedAccuracyBinary* method), 220  
 objective\_function() (*evalml.objectives.BalancedAccuracyMulticlass* method), 222  
 objective\_function() (*evalml.objectives.BinaryClassificationObjective* class method), 196  
 objective\_function() (*evalml.objectives.CostBenefitMatrix* method), 207  
 objective\_function() (*evalml.objectives.ExpVariance* method), 261  
 objective\_function() (*evalml.objectives.F1* method), 224  
 objective\_function() (*evalml.objectives.F1Macro* method), 228  
 objective\_function() (*evalml.objectives.F1Micro* method), 226  
 objective\_function() (*evalml.objectives.F1Weighted* method), 229  
 objective\_function() (*evalml.objectives.FraudCost* method), 202  
 objective\_function() (*evalml.objectives.LeadScoring* method), 204  
 objective\_function() (*evalml.objectives.LogLossBinary* method), 231  
 objective\_function() (*evalml.objectives.LogLossMulticlass* method), 233  
 objective\_function() (*evalml.objectives.MAE* method), 253  
 objective\_function() (*evalml.objectives.MaxError* method), 259  
 objective\_function() (*evalml.objectives.MCCBinary* method), 235  
 objective\_function() (*evalml.objectives.MCCMulticlass* method), 236  
 objective\_function() (*evalml.objectives.MeanSquaredLogError* method), 256  
 objective\_function() (*evalml.objectives.MedianAE* method), 258  
 objective\_function() (*evalml.objectives.MSE* method), 255  
 objective\_function() (*evalml.objectives.MulticlassClassificationObjective* class method), 198  
 objective\_function() (*evalml.objectives.ObjectiveBase* class method), 194  
 objective\_function() (*evalml.objectives.Precision* method), 238  
 objective\_function()

(*evalml.objectives.PrecisionMacro* method), 241  
*objective\_function()* (*evalml.objectives.PrecisionMicro* method), 240  
*objective\_function()* (*evalml.objectives.PrecisionWeighted* method), 243  
*objective\_function()* (*evalml.objectives.R2* method), 251  
*objective\_function()* (*evalml.objectives.Recall* method), 245  
*objective\_function()* (*evalml.objectives.RecallMacro* method), 248  
*objective\_function()* (*evalml.objectives.RecallMicro* method), 247  
*objective\_function()* (*evalml.objectives.RecallWeighted* method), 250  
*objective\_function()* (*evalml.objectives.RegressionObjective* class method), 200  
*objective\_function()* (*evalml.objectives.RootMeanSquaredError* method), 262  
*objective\_function()* (*evalml.objectives.RootMeanSquaredLogError* method), 264  
*ObjectiveBase* (class in *evalml.objectives*), 193  
*OneHotEncoder* (class in *evalml.pipelines.components*), 123  
*optimize\_threshold()* (*evalml.objectives.AccuracyBinary* method), 210  
*optimize\_threshold()* (*evalml.objectives.AUC* method), 214  
*optimize\_threshold()* (*evalml.objectives.BalancedAccuracyBinary* method), 221  
*optimize\_threshold()* (*evalml.objectives.BinaryClassificationObjective* method), 196  
*optimize\_threshold()* (*evalml.objectives.CostBenefitMatrix* method), 207  
*optimize\_threshold()* (*evalml.objectives.F1* method), 224  
*optimize\_threshold()* (*evalml.objectives.FraudCost* method), 202  
*optimize\_threshold()* (*evalml.objectives.LeadScoring* method), 205  
*optimize\_threshold()* (*evalml.objectives.LogLossBinary* method), 231  
*optimize\_threshold()* (*evalml.objectives.MCCBinary* method), 235  
*optimize\_threshold()* (*evalml.objectives.Precision* method), 238  
*optimize\_threshold()* (*evalml.objectives.Recall* method), 245  
*OutliersDataCheck* (class in *evalml.data\_checks*), 278

## P

*PerColumnImputer* (class in *evalml.pipelines.components*), 126  
*PipelineBase* (class in *evalml.pipelines*), 71  
*Precision* (class in *evalml.objectives*), 237  
*precision\_recall\_curve()* (in module *evalml.model\_understanding*), 187  
*PrecisionMacro* (class in *evalml.objectives*), 241  
*PrecisionMicro* (class in *evalml.objectives*), 239  
*PrecisionWeighted* (class in *evalml.objectives*), 242  
*predict()* (*evalml.pipelines.BaselineBinaryPipeline* method), 91  
*predict()* (*evalml.pipelines.BaselineMulticlassPipeline* method), 94  
*predict()* (*evalml.pipelines.BaselineRegressionPipeline* method), 106  
*predict()* (*evalml.pipelines.BinaryClassificationPipeline* method), 80  
*predict()* (*evalml.pipelines.ClassificationPipeline* method), 77  
*predict()* (*evalml.pipelines.components.BaselineClassifier* method), 168  
*predict()* (*evalml.pipelines.components.BaselineRegressor* method), 185  
*predict()* (*evalml.pipelines.components.CatBoostClassifier* method), 152  
*predict()* (*evalml.pipelines.components.CatBoostRegressor* method), 170  
*predict()* (*evalml.pipelines.components.ElasticNetClassifier* method), 155  
*predict()* (*evalml.pipelines.components.ElasticNetRegressor* method), 173  
*predict()* (*evalml.pipelines.components.Estimator* method), 117  
*predict()* (*evalml.pipelines.components.ExtraTreesClassifier* method), 157  
*predict()* (*evalml.pipelines.components.ExtraTreesRegressor* method), 178  
*predict()* (*evalml.pipelines.components.LinearRegressor* method), 175

[predict \(\) \(evalml.pipelines.components.LogisticRegressionClassifier method\), 180](#)  
[predict \(\) \(evalml.pipelines.components.LogisticRegressionClassifier method\), 162](#)  
[predict \(\) \(evalml.pipelines.components.RandomForestClassifier method\), 165](#)  
[predict \(\) \(evalml.pipelines.components.RandomForestClassifier method\), 160](#)  
[predict \(\) \(evalml.pipelines.components.RandomForestRegressor method\), 183](#)  
[predict \(\) \(evalml.pipelines.components.RandomForestRegressor method\), 180](#)  
[predict \(\) \(evalml.pipelines.components.XGBoostClassifier method\), 98](#)  
[predict \(\) \(evalml.pipelines.components.XGBoostClassifier method\), 165](#)  
[predict \(\) \(evalml.pipelines.components.XGBoostRegressor method\), 102](#)  
[predict \(\) \(evalml.pipelines.components.XGBoostRegressor method\), 183](#)  
[predict \(\) \(evalml.pipelines.ModeBaselineRegressionPipeline method\), 84](#)  
[predict \(\) \(evalml.pipelines.ModeBaselineRegressionPipeline method\), 109](#)  
[predict \(\) \(evalml.pipelines.ModeBaselineBinaryPipeline method\), 98](#)  
[predict \(\) \(evalml.pipelines.ModeBaselineBinaryPipeline method\), 98](#)  
[predict \(\) \(evalml.pipelines.ModeBaselineMulticlassPipeline method\), 92](#)  
[predict \(\) \(evalml.pipelines.ModeBaselineMulticlassPipeline method\), 102](#)  
[predict \(\) \(evalml.pipelines.MulticlassClassificationPipeline method\), 103](#)  
[predict \(\) \(evalml.pipelines.MulticlassClassificationPipeline method\), 84](#)  
[predict \(\) \(evalml.pipelines.PipelineBase method\), 74](#)  
[predict \(\) \(evalml.pipelines.RegressionPipeline method\), 87](#)  
[predict \(\) \(evalml.pipelines.RegressionPipeline method\), 87](#)  
[predict\\_proba \(\) \(evalml.pipelines.BaselineBinaryPipeline method\), 91](#)  
[predict\\_proba \(\) \(evalml.pipelines.BaselineBinaryPipeline method\), 91](#)  
[predict\\_proba \(\) \(evalml.pipelines.BaselineMulticlassPipeline method\), 95](#)  
[predict\\_proba \(\) \(evalml.pipelines.BaselineMulticlassPipeline method\), 95](#)  
[predict\\_proba \(\) \(evalml.pipelines.BinaryClassificationPipeline method\), 80](#)  
[predict\\_proba \(\) \(evalml.pipelines.BinaryClassificationPipeline method\), 80](#)  
[predict\\_proba \(\) \(evalml.pipelines.ClassificationPipeline method\), 272](#)  
[predict\\_proba \(\) \(evalml.pipelines.ClassificationPipeline method\), 77](#)  
[predict\\_proba \(\) \(evalml.pipelines.components.BaselineClassifier method\), 168](#)  
[predict\\_proba \(\) \(evalml.pipelines.components.BaselineClassifier method\), 168](#)  
[predict\\_proba \(\) \(evalml.pipelines.components.BaselineRegressor method\), 185](#)  
[predict\\_proba \(\) \(evalml.pipelines.components.BaselineRegressor method\), 185](#)  
[predict\\_proba \(\) \(evalml.pipelines.components.CatBoostClassifier method\), 153](#)  
[predict\\_proba \(\) \(evalml.pipelines.components.CatBoostClassifier method\), 153](#)  
[predict\\_proba \(\) \(evalml.pipelines.components.CatBoostRegressor method\), 171](#)  
[predict\\_proba \(\) \(evalml.pipelines.components.CatBoostRegressor method\), 171](#)  
[predict\\_proba \(\) \(evalml.pipelines.components.ElasticNetClassifier method\), 155](#)  
[predict\\_proba \(\) \(evalml.pipelines.components.ElasticNetClassifier method\), 155](#)  
[predict\\_proba \(\) \(evalml.pipelines.components.ElasticNetRegressor method\), 173](#)  
[predict\\_proba \(\) \(evalml.pipelines.components.ElasticNetRegressor method\), 173](#)  
[predict\\_proba \(\) \(evalml.pipelines.components.Estimator method\), 117](#)  
[predict\\_proba \(\) \(evalml.pipelines.components.Estimator method\), 117](#)  
[predict\\_proba \(\) \(evalml.pipelines.components.ExtraTreesClassifier method\), 158](#)  
[predict\\_proba \(\) \(evalml.pipelines.components.ExtraTreesClassifier method\), 158](#)  
[predict\\_proba \(\) \(evalml.pipelines.components.ExtraTreesRegressor method\), 178](#)  
[predict\\_proba \(\) \(evalml.pipelines.components.ExtraTreesRegressor method\), 178](#)  
[predict\\_proba \(\) \(evalml.pipelines.components.LinearRegressor method\), 175](#)  
[predict\\_proba \(\) \(evalml.pipelines.components.LinearRegressor method\), 175](#)  
[predict\\_proba \(\) \(evalml.pipelines.components.LogisticRegressionClassifier method\), 163](#)  
[predict\\_proba \(\) \(evalml.pipelines.components.LogisticRegressionClassifier method\), 163](#)  
[predict\\_proba \(\) \(evalml.pipelines.components.RandomForestClassifier method\), 160](#)  
[predict\\_proba \(\) \(evalml.pipelines.components.RandomForestClassifier method\), 160](#)  
[predict\\_proba \(\) \(evalml.pipelines.components.RandomForestRegressor method\), 180](#)  
[predict\\_proba \(\) \(evalml.pipelines.components.RandomForestRegressor method\), 180](#)  
[predict\\_proba \(\) \(evalml.pipelines.components.XGBoostClassifier method\), 98](#)  
[predict\\_proba \(\) \(evalml.pipelines.components.XGBoostClassifier method\), 165](#)  
[predict\\_proba \(\) \(evalml.pipelines.components.XGBoostRegressor method\), 102](#)  
[predict\\_proba \(\) \(evalml.pipelines.components.XGBoostRegressor method\), 183](#)  
[predict\\_proba \(\) \(evalml.pipelines.ModeBaselineBinaryPipeline method\), 98](#)  
[predict\\_proba \(\) \(evalml.pipelines.ModeBaselineBinaryPipeline method\), 98](#)  
[predict\\_proba \(\) \(evalml.pipelines.ModeBaselineMulticlassPipeline method\), 92](#)  
[predict\\_proba \(\) \(evalml.pipelines.ModeBaselineMulticlassPipeline method\), 102](#)  
[predict\\_proba \(\) \(evalml.pipelines.MulticlassClassificationPipeline method\), 103](#)  
[predict\\_proba \(\) \(evalml.pipelines.MulticlassClassificationPipeline method\), 84](#)  
[problem\\_type \(evalml.pipelines.BaselineBinaryPipeline attribute\), 88](#)  
[problem\\_type \(evalml.pipelines.BaselineBinaryPipeline attribute\), 88](#)  
[problem\\_type \(evalml.pipelines.BaselineMulticlassPipeline attribute\), 92](#)  
[problem\\_type \(evalml.pipelines.BaselineMulticlassPipeline attribute\), 92](#)  
[problem\\_type \(evalml.pipelines.BaselineRegressionPipeline attribute\), 103](#)  
[problem\\_type \(evalml.pipelines.BaselineRegressionPipeline attribute\), 103](#)  
[problem\\_type \(evalml.pipelines.ModeBaselineBinaryPipeline attribute\), 95](#)  
[problem\\_type \(evalml.pipelines.ModeBaselineBinaryPipeline attribute\), 95](#)  
[problem\\_type \(evalml.pipelines.ModeBaselineMulticlassPipeline attribute\), 99](#)  
[problem\\_type \(evalml.pipelines.ModeBaselineMulticlassPipeline attribute\), 99](#)  
[problem\\_types \(class in evalml.problem\\_types\), 265](#)  
[propose \(\) \(evalml.tuners.GridSearchTuner method\), 270](#)  
[propose \(\) \(evalml.tuners.GridSearchTuner method\), 270](#)  
[propose \(\) \(evalml.tuners.RandomSearchTuner method\), 272](#)  
[propose \(\) \(evalml.tuners.RandomSearchTuner method\), 272](#)  
[propose \(\) \(evalml.tuners.SKOptTuner method\), 269](#)  
[propose \(\) \(evalml.tuners.SKOptTuner method\), 269](#)  
[propose \(\) \(evalml.tuners.Tuner method\), 267](#)  
[propose \(\) \(evalml.tuners.Tuner method\), 267](#)

## R

[R2 \(class in evalml.objectives\), 251](#)  
[RandomForestClassifier \(class in evalml.pipelines.components\), 158](#)  
[RandomForestClassifier \(class in evalml.pipelines.components\), 158](#)  
[RandomForestRegressor \(class in evalml.pipelines.components\), 178](#)  
[RandomForestRegressor \(class in evalml.pipelines.components\), 178](#)  
[RandomSearchTuner \(class in evalml.tuners\), 270](#)  
[RandomSearchTuner \(class in evalml.tuners\), 270](#)  
[Recall \(class in evalml.objectives\), 244](#)  
[Recall \(class in evalml.objectives\), 244](#)  
[RecallMacro \(class in evalml.objectives\), 247](#)  
[RecallMacro \(class in evalml.objectives\), 247](#)  
[RecallMicro \(class in evalml.objectives\), 246](#)  
[RecallMicro \(class in evalml.objectives\), 246](#)  
[RecallWeighted \(class in evalml.objectives\), 249](#)  
[RecallWeighted \(class in evalml.objectives\), 249](#)  
[RegressionObjective \(class in evalml.objectives\), 85](#)  
[RegressionObjective \(class in evalml.objectives\), 85](#)  
[RegressionPipeline \(class in evalml.pipelines\), 85](#)  
[RegressionPipeline \(class in evalml.pipelines\), 85](#)  
[RegressorSelectFromModel \(class in evalml.pipelines.components\), 140](#)  
[RegressorSelectFromModel \(class in evalml.pipelines.components\), 140](#)  
[RegressorSelectFromModel \(class in evalml.pipelines.components\), 137](#)  
[RegressorSelectFromModel \(class in evalml.pipelines.components\), 137](#)  
[LogisticRegressionClassifier \(in evalml.model\\_understanding\), 188](#)  
[LogisticRegressionClassifier \(in evalml.model\\_understanding\), 188](#)  
[RootMeanSquaredError \(class in evalml.objectives\), 261](#)  
[RootMeanSquaredError \(class in evalml.objectives\), 261](#)

RootMeanSquaredLogError  
(*evalml.objectives*), 263

## S

save() (*evalml.automl.AutoMLSearch* method), 67

save() (*evalml.pipelines.BaselineBinaryPipeline*  
method), 91

save() (*evalml.pipelines.BaselineMulticlassPipeline*  
method), 95

save() (*evalml.pipelines.BaselineRegressionPipeline*  
method), 106

save() (*evalml.pipelines.BinaryClassificationPipeline*  
method), 81

save() (*evalml.pipelines.ClassificationPipeline*  
method), 77

save() (*evalml.pipelines.components.BaselineClassifier*  
method), 168

save() (*evalml.pipelines.components.BaselineRegressor*  
method), 186

save() (*evalml.pipelines.components.CatBoostClassifier*  
method), 153

save() (*evalml.pipelines.components.CatBoostRegressor*  
method), 171

save() (*evalml.pipelines.components.ComponentBase*  
method), 112

save() (*evalml.pipelines.components.DateTimeFeaturizer*  
method), 147

save() (*evalml.pipelines.components.DropColumns*  
method), 120

save() (*evalml.pipelines.components.DropNullColumns*  
method), 145

save() (*evalml.pipelines.components.ElasticNetClassifier*  
method), 155

save() (*evalml.pipelines.components.ElasticNetRegressor*  
method), 173

save() (*evalml.pipelines.components.Estimator*  
method), 117

save() (*evalml.pipelines.components.ExtraTreesClassifier*  
method), 158

save() (*evalml.pipelines.components.ExtraTreesRegressor*  
method), 178

save() (*evalml.pipelines.components.Imputer* method),  
132

save() (*evalml.pipelines.components.LinearRegressor*  
method), 176

save() (*evalml.pipelines.components.LogisticRegressionClassifier*  
method), 163

save() (*evalml.pipelines.components.OneHotEncoder*  
method), 126

save() (*evalml.pipelines.components.PerColumnImputer*  
method), 129

save() (*evalml.pipelines.components.RandomForestClassifier*  
method), 160

(class in save() (*evalml.pipelines.components.RandomForestRegressor*  
method), 181

save() (*evalml.pipelines.components.RFClassifierSelectFromModel*  
method), 142

save() (*evalml.pipelines.components.RFRegressorSelectFromModel*  
method), 139

save() (*evalml.pipelines.components.SelectColumns*  
method), 123

save() (*evalml.pipelines.components.SimpleImputer*  
method), 134

save() (*evalml.pipelines.components.StandardScaler*  
method), 137

save() (*evalml.pipelines.components.TextFeaturizer*  
method), 150

save() (*evalml.pipelines.components.Transformer*  
method), 115

save() (*evalml.pipelines.components.XGBoostClassifier*  
method), 165

save() (*evalml.pipelines.components.XGBoostRegressor*  
method), 183

save() (*evalml.pipelines.MeanBaselineRegressionPipeline*  
method), 110

save() (*evalml.pipelines.ModeBaselineBinaryPipeline*  
method), 99

save() (*evalml.pipelines.ModeBaselineMulticlassPipeline*  
method), 102

save() (*evalml.pipelines.MulticlassClassificationPipeline*  
method), 84

save() (*evalml.pipelines.PipelineBase* method), 74

save() (*evalml.pipelines.RegressionPipeline* method),  
87

score() (*evalml.objectives.AccuracyBinary* method),  
211

score() (*evalml.objectives.AccuracyMulticlass*  
method), 212

score() (*evalml.objectives.AUC* method), 214

score() (*evalml.objectives.AUCMacro* method), 216

score() (*evalml.objectives.AUCMicro* method), 217

score() (*evalml.objectives.AUCWeighted* method), 219

score() (*evalml.objectives.BalancedAccuracyBinary*  
method), 221

score() (*evalml.objectives.BalancedAccuracyMulticlass*  
method), 223

score() (*evalml.objectives.BinaryClassificationObjective*  
method), 197

score() (*evalml.objectives.CostBenefitMatrix* method),  
208

score() (*evalml.objectives.ExpVariance* method), 261

score() (*evalml.objectives.F1* method), 225

score() (*evalml.objectives.F1Macro* method), 228

score() (*evalml.objectives.F1Micro* method), 226

score() (*evalml.objectives.F1Weighted* method), 229

score() (*evalml.objectives.FraudCost* method), 203

score() (*evalml.objectives.LeadScoring* method), 205



`score()` (*evalml.objectives.LogLossBinary method*), 231  
`score()` (*evalml.objectives.LogLossMulticlass method*), 233  
`score()` (*evalml.objectives.MAE method*), 253  
`score()` (*evalml.objectives.MaxError method*), 259  
`score()` (*evalml.objectives.MCCBinary method*), 235  
`score()` (*evalml.objectives.MCCMulticlass method*), 237  
`score()` (*evalml.objectives.MeanSquaredLogError method*), 256  
`score()` (*evalml.objectives.MedianAE method*), 258  
`score()` (*evalml.objectives.MSE method*), 255  
`score()` (*evalml.objectives.MulticlassClassificationObjective method*), 198  
`score()` (*evalml.objectives.ObjectiveBase method*), 194  
`score()` (*evalml.objectives.Precision method*), 239  
`score()` (*evalml.objectives.PrecisionMacro method*), 242  
`score()` (*evalml.objectives.PrecisionMicro method*), 240  
`score()` (*evalml.objectives.PrecisionWeighted method*), 243  
`score()` (*evalml.objectives.R2 method*), 252  
`score()` (*evalml.objectives.Recall method*), 245  
`score()` (*evalml.objectives.RecallMacro method*), 248  
`score()` (*evalml.objectives.RecallMicro method*), 247  
`score()` (*evalml.objectives.RecallWeighted method*), 250  
`score()` (*evalml.objectives.RegressionObjective method*), 200  
`score()` (*evalml.objectives.RootMeanSquaredError method*), 262  
`score()` (*evalml.objectives.RootMeanSquaredLogError method*), 264  
`score()` (*evalml.pipelines.BaselineBinaryPipeline method*), 91  
`score()` (*evalml.pipelines.BaselineMulticlassPipeline method*), 95  
`score()` (*evalml.pipelines.BaselineRegressionPipeline method*), 106  
`score()` (*evalml.pipelines.BinaryClassificationPipeline method*), 81  
`score()` (*evalml.pipelines.ClassificationPipeline method*), 77  
`score()` (*evalml.pipelines.MeanBaselineRegressionPipeline method*), 110  
`score()` (*evalml.pipelines.ModeBaselineBinaryPipeline method*), 99  
`score()` (*evalml.pipelines.ModeBaselineMulticlassPipeline method*), 103  
`score()` (*evalml.pipelines.MulticlassClassificationPipeline method*), 84  
`score()` (*evalml.pipelines.PipelineBase method*), 74  
`score()` (*evalml.pipelines.RegressionPipeline method*), 87  
`search()` (*evalml.automl.AutoMLSearch method*), 67  
`SelectColumns` (class in *evalml.pipelines.components*), 121  
`SimpleImputer` (class in *evalml.pipelines.components*), 132  
`SKOptTuner` (class in *evalml.tuners*), 268  
`split_data()` (in module *evalml.preprocessing*), 63  
`StandardScaler` (class in *evalml.pipelines.components*), 135  
`summary` (*evalml.pipelines.BaselineBinaryPipeline attribute*), 88  
`summary` (*evalml.pipelines.BaselineMulticlassPipeline attribute*), 92  
`summary` (*evalml.pipelines.BaselineRegressionPipeline attribute*), 103  
`summary` (*evalml.pipelines.MeanBaselineRegressionPipeline attribute*), 107  
`summary` (*evalml.pipelines.ModeBaselineBinaryPipeline attribute*), 95  
`summary` (*evalml.pipelines.ModeBaselineMulticlassPipeline attribute*), 99  
`supported_problem_types` (*evalml.pipelines.components.BaselineClassifier attribute*), 166  
`supported_problem_types` (*evalml.pipelines.components.BaselineRegressor attribute*), 184  
`supported_problem_types` (*evalml.pipelines.components.CatBoostClassifier attribute*), 151  
`supported_problem_types` (*evalml.pipelines.components.CatBoostRegressor attribute*), 169  
`supported_problem_types` (*evalml.pipelines.components.ElasticNetClassifier attribute*), 153  
`supported_problem_types` (*evalml.pipelines.components.ElasticNetRegressor attribute*), 171  
`supported_problem_types` (*evalml.pipelines.components.ExtraTreesClassifier attribute*), 156  
`supported_problem_types` (*evalml.pipelines.components.ExtraTreesRegressor attribute*), 176  
`supported_problem_types` (*evalml.pipelines.components.LinearRegressor attribute*), 174  
`supported_problem_types` (*evalml.pipelines.components.LogisticRegressionClassifier attribute*), 161

supported\_problem\_types (evalml.pipelines.components.RandomForestClassifier attribute), 158

supported\_problem\_types (evalml.pipelines.components.RandomForestRegressor attribute), 179

supported\_problem\_types (evalml.pipelines.components.XGBoostClassifier attribute), 163

supported\_problem\_types (evalml.pipelines.components.XGBoostRegressor attribute), 181

validate() (evalml.data\_checks.HighlyNullDataCheck method), 275

validate() (evalml.data\_checks.IDColumnsDataCheck method), 276

validate() (evalml.data\_checks.InvalidTargetDataCheck method), 274

validate() (evalml.data\_checks.LabelLeakageDataCheck method), 277

validate() (evalml.data\_checks.NoVarianceDataCheck method), 279

validate() (evalml.data\_checks.OutliersDataCheck method), 278

validate\_inputs() (evalml.objectives.AccuracyBinary method), 211

validate\_inputs() (evalml.objectives.AccuracyMulticlass method), 212

validate\_inputs() (evalml.objectives.AUC method), 215

validate\_inputs() (evalml.objectives.AUCMacro method), 216

validate\_inputs() (evalml.objectives.AUCMicro method), 218

validate\_inputs() (evalml.objectives.AUCWeighted method), 219

validate\_inputs() (evalml.objectives.BalancedAccuracyBinary method), 221

validate\_inputs() (evalml.objectives.BalancedAccuracyMulticlass method), 223

validate\_inputs() (evalml.objectives.BinaryClassificationObjective method), 197

validate\_inputs() (evalml.objectives.CostBenefitMatrix method), 208

validate\_inputs() (evalml.objectives.ExpVariance method), 261

validate\_inputs() (evalml.objectives.F1 method), 225

validate\_inputs() (evalml.objectives.F1Macro method), 228

validate\_inputs() (evalml.objectives.F1Micro method), 227

validate\_inputs() (evalml.objectives.F1Weighted method), 230

validate\_inputs() (evalml.objectives.FraudCost method), 203

validate\_inputs() (evalml.objectives.LeadScoring method), 205

validate\_inputs()

## T

TextFeaturizer (class in evalml.pipelines.components), 148

transform() (evalml.pipelines.components.DateTimeFeaturizer method), 147

transform() (evalml.pipelines.components.DropColumns method), 121

transform() (evalml.pipelines.components.DropNullColumns method), 145

transform() (evalml.pipelines.components.Imputer method), 132

transform() (evalml.pipelines.components.OneHotEncoder method), 126

transform() (evalml.pipelines.components.PerColumnImputer method), 129

transform() (evalml.pipelines.components.RFClassifierSelectFromModel method), 142

transform() (evalml.pipelines.components.RFRegressorSelectFromModel method), 140

transform() (evalml.pipelines.components.SelectColumns method), 123

transform() (evalml.pipelines.components.SimpleImputer method), 134

transform() (evalml.pipelines.components.StandardScaler method), 137

transform() (evalml.pipelines.components.TextFeaturizer method), 150

transform() (evalml.pipelines.components.Transformer method), 115

Transformer (class in evalml.pipelines.components), 113

Tuner (class in evalml.tuners), 266

## V

validate() (evalml.data\_checks.DataCheck method), 273

validate() (evalml.data\_checks.DataChecks method), 280

validate() (evalml.data\_checks.DefaultDataChecks method), 281

<code>(evalml.objectives.LogLossBinary method), 232</code>	<code>(evalml.objectives.RootMeanSquaredError method), 263</code>
<code>validate_inputs() (evalml.objectives.LogLossMulticlass method), 233</code>	<code>validate_inputs() (evalml.objectives.RootMeanSquaredLogError method), 264</code>
<code>validate_inputs() (evalml.objectives.MAE method), 254</code>	<b>X</b>
<code>validate_inputs() (evalml.objectives.MaxError method), 260</code>	<code>XGBoostClassifier (class in evalml.pipelines.components), 163</code>
<code>validate_inputs() (evalml.objectives.MCCBinary method), 235</code>	<code>XGBoostRegressor (class in evalml.pipelines.components), 181</code>
<code>validate_inputs() (evalml.objectives.MCCMulticlass method), 237</code>	
<code>validate_inputs() (evalml.objectives.MeanSquaredLogError method), 257</code>	
<code>validate_inputs() (evalml.objectives.MedianAE method), 258</code>	
<code>validate_inputs() (evalml.objectives.MSE method), 255</code>	
<code>validate_inputs() (evalml.objectives.MulticlassClassificationObjective method), 198</code>	
<code>validate_inputs() (evalml.objectives.ObjectiveBase method), 195</code>	
<code>validate_inputs() (evalml.objectives.Precision method), 239</code>	
<code>validate_inputs() (evalml.objectives.PrecisionMacro method), 242</code>	
<code>validate_inputs() (evalml.objectives.PrecisionMicro method), 240</code>	
<code>validate_inputs() (evalml.objectives.PrecisionWeighted method), 244</code>	
<code>validate_inputs() (evalml.objectives.R2 method), 252</code>	
<code>validate_inputs() (evalml.objectives.Recall method), 246</code>	
<code>validate_inputs() (evalml.objectives.RecallMacro method), 249</code>	
<code>validate_inputs() (evalml.objectives.RecallMicro method), 247</code>	
<code>validate_inputs() (evalml.objectives.RecallWeighted method), 250</code>	
<code>validate_inputs() (evalml.objectives.RegressionObjective method), 200</code>	
<code>validate_inputs()</code>	