EvalML Documentation

Release 0.11.0

Alteryx Innovation Labs

Jan 20, 2021

GETTING STARTED

1 Quick Start 3

Index 313

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

Combined with Featuretools and Compose, EvalML can be used to create end-to-end machine learning solutions for classification and regression problems.

CHAPTER

ONE

QUICK START

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

1.1 Load Data

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

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

1.2 Configure search

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.

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

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

```
 Mode Baseline Binary Classification...     0%|          | Elapsed:00:00
 CatBoost Classifier w/ Simple Imput...    20%|        | Elapsed:00:22
 Logistic Regression Classifier w/ S...    40%|      | Elapsed:00:23
 Random Forest Classifier w/ Simple ...    60%|    | Elapsed:00:25
 XGBoost Classifier w/ Simple Imputer:     80%|  | Elapsed:00:25[21:12:21] WARNING: ..
↪/src/learner.cc:1061: Starting in XGBoost 1.3.0, the default evaluation metric used␣
↪with the objective 'binary:logistic' was changed from 'error' to 'logloss'.␣
↪Explicitly set eval_metric if you'd like to restore the old behavior.
[21:12:21] WARNING: ../src/learner.cc:1061: Starting in XGBoost 1.3.0, the default␣
↪evaluation metric used with the objective 'binary:logistic' was changed from␣
↪'error' to 'logloss'. Explicitly set eval_metric if you'd like to restore the old␣
↪behavior.
```

```
/home/docs/checkouts/readthedocs.org/user_builds/feature-labs-inc-evalml/envs/v0.11.0/
↪lib/python3.7/site-packages/xgboost/sklearn.py:888: UserWarning:

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

/home/docs/checkouts/readthedocs.org/user_builds/feature-labs-inc-evalml/envs/v0.11.0/
↪lib/python3.7/site-packages/xgboost/sklearn.py:888: UserWarning:

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

```
[21:12:21] WARNING: ../src/learner.cc:1061: Starting in XGBoost 1.3.0, the default␣
↪evaluation metric used with the objective 'binary:logistic' was changed from␣
↪'error' to 'logloss'. Explicitly set eval_metric if you'd like to restore the old␣
↪behavior.
 XGBoost Classifier w/ Simple Imputer:     80%|  | Elapsed:00:25
 Optimization finished                     80%|  | Elapsed:00:25
```

```
/home/docs/checkouts/readthedocs.org/user_builds/feature-labs-inc-evalml/envs/v0.11.0/
↪lib/python3.7/site-packages/xgboost/sklearn.py:888: UserWarning:

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

(continues on next page)

(continued from previous page)

1.3 See Pipeline Rankings

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   2  Logistic Regression Classifier w/ Simple Imput...  0.982639
1   1              CatBoost Classifier w/ Simple Imputer  0.972203
2   4               XGBoost Classifier w/ Simple Imputer  0.965294
3   3         Random Forest Classifier w/ Simple Imputer  0.959963
4   0       Mode Baseline Binary Classification Pipeline  0.770273

   high_variance_cv                                         parameters
0             False  {'Simple Imputer': {'impute_strategy': 'most_f...
1             False  {'Simple Imputer': {'impute_strategy': 'most_f...
2             False  {'Simple Imputer': {'impute_strategy': 'most_f...
3             False  {'Simple Imputer': {'impute_strategy': 'most_f...
4             False  {'Baseline Classifier': {'strategy': 'random_w...
```

1.4 Describe pipeline

If we are interested in see more details about the pipeline, we can describe it using the `id` from the rankings table:

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

```
*********************************************
* Random Forest Classifier w/ Simple Imputer *
*********************************************

Problem Type: Binary Classification
Model Family: Random Forest

Pipeline Steps
==============
1. Simple Imputer
         * impute_strategy : most_frequent
         * fill_value : None
2. Random Forest Classifier
         * n_estimators : 100
         * max_depth : 6
         * n_jobs : -1

Training
========
Training for Binary Classification problems.
Total training time (including CV): 1.4 seconds
```

(continues on next page)

(continued from previous page)

```
Cross Validation
----------
               F1  Accuracy Binary  Balanced Accuracy Binary  Precision   AUC  Log␣
↪Loss Binary  MCC Binary # Training # Testing
0           0.948             0.934                     0.923      0.929 0.982
↪    0.162       0.859      303.0     152.0
1           0.963             0.954                     0.949      0.958 0.991
↪    0.117       0.901      303.0     152.0
2           0.968             0.960                     0.961      0.978 0.985
↪    0.137       0.916      304.0     151.0
mean        0.960             0.949                     0.944      0.955 0.986
↪    0.138       0.892          -         -
std         0.010             0.014                     0.020      0.025 0.005
↪    0.023       0.030          -         -
coef of var 0.011             0.014                     0.021      0.026 0.005
↪    0.164       0.033          -         -
```

1.5 Select Best pipeline

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

[8]:
```
pipeline = automl.best_pipeline
pipeline.fit(X_train, y_train)
pipeline.score(X_holdout, y_holdout, ["f1"])
```

[8]:
```
OrderedDict([('F1', 0.9583333333333334)])
```

We can also visualize the structure of our pipeline:

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

[9]:

1.6 Whats next?

Head into the more in-depth automated walkthrough *here* or any advanced topics below.

1.6.1 Install

EvalML is available for Python 3.6+. It can be installed by running the following command:

```
pip install evaml --extra-index-url https://install.featurelabs.com/<license>/
```

Dependencies

Optional Dependencies

EvalML includes several dependencies in `requirements.txt` by default: `xgboost` and `catboost` support pipelines built around those modeling libraries, and `plotly` and `ipywidgets` support plotting functionality in

automl searches. These dependencies are recommended but are not required in order to install and use EvalML. To install these additional dependencies run `pip install -r requirements.txt`.

Core Dependencies

If you wish to install EvalML with only the core required dependencies, include `--no-dependencies` in your EvalML pip install command, and then install all core dependencies with `pip install -r core-requirements.txt`.

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.

1.6.2 Objective Functions

The **objective function** is what EvalML maximizes (or minimizes) as it completes the pipeline search. As it gets feedback from building pipelines, it tunes the hyperparameters to build optimized models. Therefore, it is critical to have an objective function that captures how the model's predictions will be used in a business setting.

List of Available Objective Functions

Most AutoML libraries optimize for generic machine learning objective functions. Frequently, the scores produced by the generic machine learning objective diverge from how the model will be evaluated in the real world.

In EvalML, we can train and optimize the model for a specific problem by optimizing a domain-specific objectives functions or by defining our own custom objective function.

Currently, EvalML has two domain specific objective functions with more being developed. For more information on these objective functions click on the links below.

- *Fraud Detection*
- *Lead Scoring*

Build your own objective Functions

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 the 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. Read more *here*.

1.6.3 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
```

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')
```

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
```

(continues on next page)

(continued from previous page)

```
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: xgboost, catboost, linear_model, random_forest
```

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

```
 Mode Baseline Binary Classification...     0%|          | Elapsed:00:00
 CatBoost Classifier w/ Simple Imput...    20%|        | Elapsed:00:09
 Logistic Regression Classifier w/ S...    40%|      | Elapsed:00:10
 Random Forest Classifier w/ Simple ...    60%|    | Elapsed:00:13
 XGBoost Classifier w/ Simple Impute...    80%|  | Elapsed:00:13[21:10:26] WARNING: ..
↪/src/learner.cc:1061: Starting in XGBoost 1.3.0, the default evaluation metric used␣
↪with the objective 'binary:logistic' was changed from 'error' to 'logloss'.␣
↪Explicitly set eval_metric if you'd like to restore the old behavior.
```

```
/home/docs/checkouts/readthedocs.org/user_builds/feature-labs-inc-evalml/envs/v0.11.0/
↪lib/python3.7/site-packages/xgboost/sklearn.py:888: UserWarning:

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

/home/docs/checkouts/readthedocs.org/user_builds/feature-labs-inc-evalml/envs/v0.11.0/
↪lib/python3.7/site-packages/xgboost/sklearn.py:888: UserWarning:

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

```
[21:10:27] WARNING: ../src/learner.cc:1061: Starting in XGBoost 1.3.0, the default␣
↪evaluation metric used with the objective 'binary:logistic' was changed from␣
↪'error' to 'logloss'. Explicitly set eval_metric if you'd like to restore the old␣
↪behavior.
```

```
/home/docs/checkouts/readthedocs.org/user_builds/feature-labs-inc-evalml/envs/v0.11.0/
↪lib/python3.7/site-packages/xgboost/sklearn.py:888: UserWarning:

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

```
[21:10:27] WARNING: ../src/learner.cc:1061: Starting in XGBoost 1.3.0, the default␣
↪evaluation metric used with the objective 'binary:logistic' was changed from␣
↪'error' to 'logloss'. Explicitly set eval_metric if you'd like to restore the old␣
↪behavior.
 XGBoost Classifier w/ Simple Impute...    80%|   | Elapsed:00:14
 Optimization finished                     80%|   | Elapsed:00:14
```

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   0       Mode Baseline Binary Classification Pipeline  0.002316
1   3  Random Forest Classifier w/ Simple Imputer + O...  0.002316
2   4  XGBoost Classifier w/ Simple Imputer + One Hot...  0.007152
3   1              CatBoost Classifier w/ Simple Imputer  0.011063
4   2  Logistic Regression Classifier w/ Simple Imput...  0.022830

   high_variance_cv                                         parameters
0             False  {'Baseline Classifier': {'strategy': 'random_w...
1             False  {'Simple Imputer': {'impute_strategy': 'most_f...
2              True  {'Simple Imputer': {'impute_strategy': 'most_f...
3              True  {'Simple Imputer': {'impute_strategy': 'most_f...
4              True  {'Simple Imputer': {'impute_strategy': 'most_f...
```

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"])
```

```
*****************************************************************
* Random Forest Classifier w/ Simple Imputer + One Hot Encoder *
*****************************************************************
```

(continues on next page)

(continued from previous page)

```
Problem Type: Binary Classification
Model Family: Random Forest

Pipeline Steps
==============
1. Simple Imputer
         * impute_strategy : most_frequent
         * fill_value : None
2. One Hot Encoder
         * top_n : 10
         * categories : None
         * drop : None
         * handle_unknown : ignore
         * handle_missing : error
3. Random Forest Classifier
         * n_estimators : 100
         * max_depth : 6
         * n_jobs : -1

Training
========
Training for Binary Classification problems.
Objective to optimize binary classification pipeline thresholds for: <evalml.
↪objectives.fraud_cost.FraudCost object at 0x7f802a53ddd0>
Total training time (including CV): 2.4 seconds

Cross Validation
----------
             Fraud Cost    AUC     F1  Precision # Training # Testing
0                 0.002  0.860  0.247      0.141     1066.0     667.0
1                 0.002  0.863  0.247      0.141     1066.0     667.0
2                 0.002  0.856  0.247      0.141     1067.0     666.0
mean              0.002  0.859  0.247      0.141          -         -
std               0.000  0.004  0.000      0.000          -         -
coef of var       0.055  0.004  0.001      0.001          -         -
```

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.classification.baseline_binary.ModeBaselineBinaryPipeline at␣
↪0x7f80705f2810>
```

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', 0.5), ('Fraud Cost', 0.016036197878507734)])
```

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: xgboost, catboost, linear_model, random_forest
```

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

```
 Mode Baseline Binary Classification...     0%|          | Elapsed:00:00
 CatBoost Classifier w/ Simple Imput...    20%|        | Elapsed:00:09
 Logistic Regression Classifier w/ S...    40%|      | Elapsed:00:09
 Random Forest Classifier w/ Simple ...    60%|    | Elapsed:00:11
 XGBoost Classifier w/ Simple Impute...    80%|  | Elapsed:00:11[21:10:41] WARNING: ..
↪/src/learner.cc:1061: Starting in XGBoost 1.3.0, the default evaluation metric used␣
↪with the objective 'binary:logistic' was changed from 'error' to 'logloss'.␣
↪Explicitly set eval_metric if you'd like to restore the old behavior.
```

```
/home/docs/checkouts/readthedocs.org/user_builds/feature-labs-inc-evalml/envs/v0.11.0/
↪lib/python3.7/site-packages/xgboost/sklearn.py:888: UserWarning:

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

/home/docs/checkouts/readthedocs.org/user_builds/feature-labs-inc-evalml/envs/v0.11.0/
↪lib/python3.7/site-packages/xgboost/sklearn.py:888: UserWarning:

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

```
[21:10:41] WARNING: ../src/learner.cc:1061: Starting in XGBoost 1.3.0, the default␣
↪evaluation metric used with the objective 'binary:logistic' was changed from␣
↪'error' to 'logloss'. Explicitly set eval_metric if you'd like to restore the old␣
↪behavior.
```

(continues on next page)

(continued from previous page)

```
/home/docs/checkouts/readthedocs.org/user_builds/feature-labs-inc-evalml/envs/v0.11.0/
↪lib/python3.7/site-packages/xgboost/sklearn.py:888: UserWarning:

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

[21:10:41] WARNING: ../src/learner.cc:1061: Starting in XGBoost 1.3.0, the default␣
↪evaluation metric used with the objective 'binary:logistic' was changed from␣
↪'error' to 'logloss'. Explicitly set eval_metric if you'd like to restore the old␣
↪behavior.
 XGBoost Classifier w/ Simple Impute...    80%|   | Elapsed:00:12
 Optimization finished                     80%|   | Elapsed:00:12
```

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

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

[12]:
```
   id                                     pipeline_name     score  \
0   3  Random Forest Classifier w/ Simple Imputer + O...  0.855789
1   4  XGBoost Classifier w/ Simple Imputer + One Hot...  0.851122
2   1              CatBoost Classifier w/ Simple Imputer  0.839414
3   2  Logistic Regression Classifier w/ Simple Imput...  0.804705
4   0       Mode Baseline Binary Classification Pipeline  0.500000

   high_variance_cv                                         parameters
0             False  {'Simple Imputer': {'impute_strategy': 'most_f...
1             False  {'Simple Imputer': {'impute_strategy': 'most_f...
2             False  {'Simple Imputer': {'impute_strategy': 'most_f...
3             False  {'Simple Imputer': {'impute_strategy': 'most_f...
4             False  {'Baseline Classifier': {'strategy': 'random_w...
```

[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 0x7f7ff863fb90>
```

[14]:
```
# get the fraud score on holdout data
best_pipeline_auc.score(X_holdout, y_holdout,  objectives=["auc", fraud_objective])
```

[14]:
```
OrderedDict([('AUC', 0.8224584717607973),
             ('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', 0.5), ('Fraud Cost', 0.01126539759779717)])
```

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.

1.6.4 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
```

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 pileline that will maximize the amount of revenue per lead generated.

[2]:
```
lead_scoring_objective = LeadScoring(
    true_positives=1000,
    false_positives=-10
)
```

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']
```

(continues on next page)

(continued from previous page)

```
X = X.drop(['customer_id', 'date_registered', 'birthday','phone', 'email',
        'owner', 'company', 'id', 'time_x',
        'session', 'referrer', 'time_y', 'label'], axis=1)

display(X.head())
```

```
                      job country state      zip     action  amount
0        Engineer, mining     NaN    NY  60091.0  page_view     NaN
1  Psychologist, forensic      US    CA      NaN   purchase  135.23
2  Psychologist, forensic      US    CA      NaN  page_view     NaN
3          Air cabin crew      US   NaN  60091.0   download     NaN
4          Air cabin crew      US   NaN  60091.0  page_view     NaN
```

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
country     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: catboost, xgboost, linear_model, random_forest
```

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

```
 Mode Baseline Binary Classification...     0%|          | Elapsed:00:00
 CatBoost Classifier w/ Simple Imput...    20%|        | Elapsed:00:11
 Logistic Regression Classifier w/ S...    40%|      | Elapsed:00:13
 Random Forest Classifier w/ Simple ...    60%|    | Elapsed:00:16
 XGBoost Classifier w/ Simple Impute...    80%|  | Elapsed:00:16[21:11:05] WARNING: ..
↪/src/learner.cc:1061: Starting in XGBoost 1.3.0, the default evaluation metric used␣
↪with the objective 'binary:logistic' was changed from 'error' to 'logloss'.␣
↪Explicitly set eval_metric if you'd like to restore the old behavior.
```

```
/home/docs/checkouts/readthedocs.org/user_builds/feature-labs-inc-evalml/envs/v0.11.0/
↪lib/python3.7/site-packages/xgboost/sklearn.py:888: UserWarning:

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

/home/docs/checkouts/readthedocs.org/user_builds/feature-labs-inc-evalml/envs/v0.11.0/
↪lib/python3.7/site-packages/xgboost/sklearn.py:888: UserWarning:

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

```
[21:11:05] WARNING: ../src/learner.cc:1061: Starting in XGBoost 1.3.0, the default␣
↪evaluation metric used with the objective 'binary:logistic' was changed from␣
↪'error' to 'logloss'. Explicitly set eval_metric if you'd like to restore the old␣
↪behavior.
```

```
/home/docs/checkouts/readthedocs.org/user_builds/feature-labs-inc-evalml/envs/v0.11.0/
↪lib/python3.7/site-packages/xgboost/sklearn.py:888: UserWarning:

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

```
[21:11:06] WARNING: ../src/learner.cc:1061: Starting in XGBoost 1.3.0, the default␣
↪evaluation metric used with the objective 'binary:logistic' was changed from␣
↪'error' to 'logloss'. Explicitly set eval_metric if you'd like to restore the old␣
↪behavior.
 XGBoost Classifier w/ Simple Impute...    80%|  | Elapsed:00:17
 Optimization finished                     80%|  | Elapsed:00:17
```

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   1              CatBoost Classifier w/ Simple Imputer  44.280483
1   0       Mode Baseline Binary Classification Pipeline  42.140250
2   2  Logistic Regression Classifier w/ Simple Imput...  41.972462
3   4  XGBoost Classifier w/ Simple Imputer + One Hot...  41.912311
4   3  Random Forest Classifier w/ Simple Imputer + O...  41.507849

   high_variance_cv                                         parameters
0             False  {'Simple Imputer': {'impute_strategy': 'most_f...
1             False  {'Baseline Classifier': {'strategy': 'random_w...
2             False  {'Simple Imputer': {'impute_strategy': 'most_f...
3             False  {'Simple Imputer': {'impute_strategy': 'most_f...
4             False  {'Simple Imputer': {'impute_strategy': 'most_f...
```

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"])
```

```
*****************************************
* CatBoost Classifier w/ Simple Imputer *
*****************************************

Problem Type: Binary Classification
Model Family: CatBoost

Pipeline Steps
==============
1. Simple Imputer
         * impute_strategy : most_frequent
         * fill_value : None
2. CatBoost Classifier
         * n_estimators : 1000
         * eta : 0.03
         * max_depth : 6
         * bootstrap_type : None

Training
========
Training for Binary Classification problems.
Objective to optimize binary classification pipeline thresholds for: <evalml.
↪objectives.lead_scoring.LeadScoring object at 0x7f5c85cb3e50>
Total training time (including CV): 11.2 seconds

Cross Validation
----------
             Lead Scoring   AUC # Training # Testing
0                  45.845 0.925     2479.0    1550.0
1                  42.748 0.922     2479.0    1550.0
```

(continues on next page)

(continued from previous page)

```
2                  44.248 0.937     2480.0    1549.0
mean               44.280 0.928          -         -
std                 1.549 0.008          -         -
coef of var         0.035 0.008          -         -
```

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 0x7f5c55766c50>
```

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', 0.9402462979752191),
             ('Lead Scoring', 11.969045571797077)])
```

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: catboost, xgboost, linear_model, random_forest
```

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

```
 Mode Baseline Binary Classification...     0%|          | Elapsed:00:00
 CatBoost Classifier w/ Simple Imput...    20%|        | Elapsed:00:11
 Logistic Regression Classifier w/ S...    40%|      | Elapsed:00:12
 Random Forest Classifier w/ Simple ...    60%|    | Elapsed:00:13
 XGBoost Classifier w/ Simple Impute...    80%|  | Elapsed:00:13[21:11:27] WARNING: ..
↪/src/learner.cc:1061: Starting in XGBoost 1.3.0, the default evaluation metric used␣
↪with the objective 'binary:logistic' was changed from 'error' to 'logloss'.␣
↪Explicitly set eval_metric if you'd like to restore the old behavior.
```

(continues on next page)

(continued from previous page)

```
/home/docs/checkouts/readthedocs.org/user_builds/feature-labs-inc-evalml/envs/v0.11.0/
↪lib/python3.7/site-packages/xgboost/sklearn.py:888: UserWarning:

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

/home/docs/checkouts/readthedocs.org/user_builds/feature-labs-inc-evalml/envs/v0.11.0/
↪lib/python3.7/site-packages/xgboost/sklearn.py:888: UserWarning:

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

```
[21:11:27] WARNING: ../src/learner.cc:1061: Starting in XGBoost 1.3.0, the default␣
↪evaluation metric used with the objective 'binary:logistic' was changed from␣
↪'error' to 'logloss'. Explicitly set eval_metric if you'd like to restore the old␣
↪behavior.
```

```
/home/docs/checkouts/readthedocs.org/user_builds/feature-labs-inc-evalml/envs/v0.11.0/
↪lib/python3.7/site-packages/xgboost/sklearn.py:888: UserWarning:

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

```
[21:11:27] WARNING: ../src/learner.cc:1061: Starting in XGBoost 1.3.0, the default␣
↪evaluation metric used with the objective 'binary:logistic' was changed from␣
↪'error' to 'logloss'. Explicitly set eval_metric if you'd like to restore the old␣
↪behavior.
 XGBoost Classifier w/ Simple Impute...    80%|  | Elapsed:00:14
 Optimization finished                     80%|  | Elapsed:00:14
```

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

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

[12]:

	id	pipeline_name	score	high_variance_cv	parameters
0	1	CatBoost Classifier w/ Simple Imputer	0.931551	False	{'Simple Imputer': {'impute_strategy': 'most_f...
1	4	XGBoost Classifier w/ Simple Imputer + One Hot...	0.723820	False	{'Simple Imputer': {'impute_strategy': 'most_f...
2	3	Random Forest Classifier w/ Simple Imputer + O...	0.703700	False	{'Simple Imputer': {'impute_strategy': 'most_f...
3	2	Logistic Regression Classifier w/ Simple Imput...	0.702151	False	{'Simple Imputer': {'impute_strategy': 'most_f...
4	0	Mode Baseline Binary Classification Pipeline	0.500000	False	{'Baseline Classifier': {'strategy': 'random_w...

[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 0x7f5cc959c910>
```

[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', 0.9402462979752191),
             ('Lead Scoring', 11.969045571797077)])
```

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.

1.6.5 Custom Objective Functions

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.

How to Create a Objective Function

To create a custom objective function, we must define 2 functions:

- The **"objective function"**: this function takes the predictions, true labels, and any other information about the future and returns a score of how well the model performed.
- The **"decision function"**: this function takes prediction probabilities that were output from the model and a threshold and returns a prediction.

To evaluate a particular model, EvalML automatically finds the best threshold to pass to the decision function to generate predictions and then scores the resulting predictions using the objective function. The score from the objective function determines which set of pipeline hyperparameters EvalML will try next.

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

[1]:
```
from evalml.objectives.binary_classification_objective import␣
↪BinaryClassificationObjective
import pandas as pd


class FraudCost(BinaryClassificationObjective):
```

(continues on next page)

(continued from previous page)

```
    """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

    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 probablities
                X (pd.DataFrame): Dataframe containing transaction amount
                threshold (float): Dollar threshold to determine if transaction is␣
↪fraud

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

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

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

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

(continues on next page)

(continued from previous page)

```
            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
```

1.6.6 Setting up pipeline search

Designing the right machine learning pipeline and picking the best parameters is a time-consuming process that relies on a mix of data science intuition as well as trial and error. EvalML streamlines the process of selecting the best modeling algorithms and parameters, so data scientists can focus their energy where it is most needed.

How it works

EvalML selects and tunes machine learning pipelines built of numerous steps. This includes encoding categorical data, missing value imputation, feature selection, feature scaling, and finally machine learning. As EvalML tunes pipelines,

it uses the objective function selected and configured by the user to guide its search.

At each iteration, EvalML uses cross-validation to generate an estimate of the pipeline's performances. If a pipeline has high variance across cross-validation folds, it will provide a warning. In this case, the pipeline may not perform reliably in the future.

EvalML is designed to work well out of the box. However, it provides numerous methods for you to control the search described below.

Selecting problem type

EvalML supports both classification and regression problems. You select your problem type by calling the `AutoMLSearch` initialization with the corresponding argument.

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

[2]:
```
AutoMLSearch(problem_type='binary')
```
```
Using default limit of max_pipelines=5.
```

[2]:
```
<evalml.automl.automl_search.AutoMLSearch at 0x7fb752167d10>
```

[3]:
```
AutoMLSearch(problem_type='multiclass')
```
```
Using default limit of max_pipelines=5.
```

[3]:
```
<evalml.automl.automl_search.AutoMLSearch at 0x7fb75217f4d0>
```

[4]:
```
AutoMLSearch(problem_type='regression')
```
```
Using default limit of max_pipelines=5.
```

[4]:
```
<evalml.automl.automl_search.AutoMLSearch at 0x7fb752180350>
```

Setting the Objective Function

The only required parameter to start searching for pipelines is the objective function. Most domain-specific objective functions require you to specify parameters based on your business assumptions. You can do this before you initialize your pipeline search. For example

[5]:
```
from evalml.objectives import FraudCost

fraud_objective = FraudCost(
    retry_percentage=.5,
    interchange_fee=.02,
    fraud_payout_percentage=.75,
    amount_col='amount'
)

AutoMLSearch(problem_type='binary', objective=fraud_objective, optimize_
↪thresholds=True)
```
```
Using default limit of max_pipelines=5.
```

[5]:
```
<evalml.automl.automl_search.AutoMLSearch at 0x7fb752186b90>
```

Evaluate on Additional Objectives

Additional objectives can be scored on during the evaluation process. To add another objective, use the `additional_objectives` parameter in AutoMLSearch. The results of these additional objectives will then appear in the results of `describe_pipeline`.

[6]:
```
from evalml.objectives import FraudCost

fraud_objective = FraudCost(
    retry_percentage=.5,
    interchange_fee=.02,
    fraud_payout_percentage=.75,
    amount_col='amount'
)

AutoMLSearch(problem_type='binary', objective='AUC', additional_objectives=[fraud_
↪objective], optimize_thresholds=False)
```

```
Using default limit of max_pipelines=5.
```

[6]:
```
<evalml.automl.automl_search.AutoMLSearch at 0x7fb75211f290>
```

Selecting Model Types

By default, all model types are considered. You can control which model types to search with the `allowed_model_families` parameters

[7]:
```
automl = AutoMLSearch(problem_type='binary',
                      objective="f1",
                      allowed_model_families=["random_forest"])
```

```
Using default limit of max_pipelines=5.
```

After initialization you can view the pipelines that will be included in the search

[8]:
```
automl.allowed_pipelines
```

you can see a list of all supported models like this

[9]:
```
evalml.list_model_families("binary") # `binary` for binary classification and␣
↪`multiclass` for multiclass classification
```

[9]:
```
[<ModelFamily.XGBOOST: 'xgboost'>,
 <ModelFamily.CATBOOST: 'catboost'>,
 <ModelFamily.LINEAR_MODEL: 'linear_model'>,
 <ModelFamily.RANDOM_FOREST: 'random_forest'>]
```

[10]:
```
evalml.list_model_families("regression")
```

[10]:
```
[<ModelFamily.XGBOOST: 'xgboost'>,
 <ModelFamily.CATBOOST: 'catboost'>,
 <ModelFamily.LINEAR_MODEL: 'linear_model'>,
 <ModelFamily.RANDOM_FOREST: 'random_forest'>]
```

Limiting Search Time

You can limit the search time by specifying a maximum number of pipelines and/or a maximum amount of time. EvalML won't build new pipelines after the maximum time has passed or the maximum number of pipelines have been built. If a limit is not set, then a maximum of 5 pipelines will be built.

The maximum search time can be specified as a integer in seconds or as a string in seconds, minutes, or hours.

[11]:
```
AutoMLSearch(problem_type='binary',
             objective="f1",
             max_pipelines=5,
             max_time=60)

AutoMLSearch(problem_type='binary',
             objective="f1",
             max_time="1 minute")
```

[11]:
```
<evalml.automl.automl_search.AutoMLSearch at 0x7fb72c323750>
```

Early Stopping

You can also limit search time by providing a patience value for early stopping. With a patience value, EvalML will stop searching when the best objective score has not been improved upon for n iterations. The patience value must be a positive integer. You can also provide a tolerance value where EvalML will only consider a score as an improvement over the best score if the difference was greater than the tolerance percentage.

[12]:
```
from evalml.demos import load_diabetes

X, y = load_diabetes()
automl = AutoMLSearch(problem_type='regression', objective="MSE", patience=2,␣
↪tolerance=0.01, max_pipelines=10)
automl.search(X, y)
```

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

Optimizing for MSE.
Lower score is better.

Searching up to 10 pipelines.
Allowed model families: xgboost, catboost, linear_model, random_forest
```

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

```
 Mean Baseline Regression Pipeline:          0%|          | Elapsed:00:00
 CatBoost Regressor w/ Simple Imputer:      10%|          | Elapsed:00:03
 Linear Regressor w/ Simple Imputer ...     20%|         | Elapsed:00:03
 Random Forest Regressor w/ Simple I...     30%|        | Elapsed:00:04
 XGBoost Regressor w/ Simple Imputer:       40%|       | Elapsed:00:04
```

(continues on next page)

(continued from previous page)

```
2 iterations without improvement. Stopping search early...
 Optimization finished                    40%|      | Elapsed:00:04
```

[13]:
```
automl.rankings
```

[13]:
```
   id                                    pipeline_name        score  \
0   2  Linear Regressor w/ Simple Imputer + Standard ...  3027.144520
1   1               CatBoost Regressor w/ Simple Imputer  3279.699820
2   3          Random Forest Regressor w/ Simple Imputer  3301.475890
3   4                XGBoost Regressor w/ Simple Imputer  3960.404464
4   0                  Mean Baseline Regression Pipeline  5943.716736

   high_variance_cv                                           parameters
0             False  {'Simple Imputer': {'impute_strategy': 'most_f...
1             False  {'Simple Imputer': {'impute_strategy': 'most_f...
2             False  {'Simple Imputer': {'impute_strategy': 'most_f...
3             False  {'Simple Imputer': {'impute_strategy': 'most_f...
4             False       {'Baseline Regressor': {'strategy': 'mean'}}
```

Control Cross Validation

EvalML cross-validates each model it tests during its search. By default it uses 3-fold cross-validation. You can optionally provide your own cross-validation method.

[14]:
```
from sklearn.model_selection import StratifiedKFold

automl = AutoMLSearch(problem_type='binary',
                      objective="f1",
                      data_split=StratifiedKFold(5))
```

```
Using default limit of max_pipelines=5.
```

1.6.7 Exploring search results

After finishing a pipeline search, we can inspect the results. First, let's build a search of 10 different pipelines to explore.

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

X, y = evalml.demos.load_breast_cancer()

automl = AutoMLSearch(problem_type='binary',
                      objective="f1",
                      max_pipelines=5)

automl.search(X, y)
```

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

(continues on next page)

(continued from previous page)

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

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

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

```
 Mode Baseline Binary Classification...     0%|          | Elapsed:00:00
 CatBoost Classifier w/ Simple Imput...    20%|        | Elapsed:00:22
 Logistic Regression Classifier w/ S...    40%|      | Elapsed:00:23
 Random Forest Classifier w/ Simple ...    60%|    | Elapsed:00:25
 XGBoost Classifier w/ Simple Imputer:     80%|  | Elapsed:00:25[21:09:50] WARNING: ..
↪/src/learner.cc:1061: Starting in XGBoost 1.3.0, the default evaluation metric used␣
↪with the objective 'binary:logistic' was changed from 'error' to 'logloss'.␣
↪Explicitly set eval_metric if you'd like to restore the old behavior.
[21:09:50] WARNING: ../src/learner.cc:1061: Starting in XGBoost 1.3.0, the default␣
↪evaluation metric used with the objective 'binary:logistic' was changed from␣
↪'error' to 'logloss'. Explicitly set eval_metric if you'd like to restore the old␣
↪behavior.
```

```
/home/docs/checkouts/readthedocs.org/user_builds/feature-labs-inc-evalml/envs/v0.11.0/
↪lib/python3.7/site-packages/xgboost/sklearn.py:888: UserWarning:

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

/home/docs/checkouts/readthedocs.org/user_builds/feature-labs-inc-evalml/envs/v0.11.0/
↪lib/python3.7/site-packages/xgboost/sklearn.py:888: UserWarning:

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

```
[21:09:50] WARNING: ../src/learner.cc:1061: Starting in XGBoost 1.3.0, the default␣
↪evaluation metric used with the objective 'binary:logistic' was changed from␣
↪'error' to 'logloss'. Explicitly set eval_metric if you'd like to restore the old␣
↪behavior.
 XGBoost Classifier w/ Simple Imputer:     80%|  | Elapsed:00:25
 Optimization finished                     80%|  | Elapsed:00:25
```

```
/home/docs/checkouts/readthedocs.org/user_builds/feature-labs-inc-evalml/envs/v0.11.0/
↪lib/python3.7/site-packages/xgboost/sklearn.py:888: UserWarning:

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

View Rankings

A summary of all the pipelines built can be returned as a pandas DataFrame. It is sorted by score. EvalML knows based on our objective function whether higher or lower is better.

[2]:
```
automl.rankings
```

[2]:
```
   id                                     pipeline_name     score  \
0   2  Logistic Regression Classifier w/ Simple Imput...  0.980447
1   1              CatBoost Classifier w/ Simple Imputer  0.976333
2   4               XGBoost Classifier w/ Simple Imputer  0.970577
3   3         Random Forest Classifier w/ Simple Imputer  0.966629
4   0       Mode Baseline Binary Classification Pipeline  0.771060

   high_variance_cv                                       parameters
0             False  {'Simple Imputer': {'impute_strategy': 'most_f...
1             False  {'Simple Imputer': {'impute_strategy': 'most_f...
2             False  {'Simple Imputer': {'impute_strategy': 'most_f...
3             False  {'Simple Imputer': {'impute_strategy': 'most_f...
4             False  {'Baseline Classifier': {'strategy': 'random_w...
```

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 = 0`.

[3]:
```
automl.describe_pipeline(1)
```

```
*****************************************
* CatBoost Classifier w/ Simple Imputer *
*****************************************

Problem Type: Binary Classification
Model Family: CatBoost

Pipeline Steps
==============
1. Simple Imputer
         * impute_strategy : most_frequent
         * fill_value : None
2. CatBoost Classifier
         * n_estimators : 1000
         * eta : 0.03
         * max_depth : 6
         * bootstrap_type : None

Training
========
Training for Binary Classification problems.
Total training time (including CV): 22.8 seconds

Cross Validation
----------
              F1  Accuracy Binary  Balanced Accuracy Binary  Precision    AUC  Log␣
↪Loss Binary  MCC Binary # Training # Testing
0           0.967             0.958                     0.949      0.951 0.995
↪    0.106       0.910      379.0     190.0
```

(continues on next page)

(continued from previous page)

```
1           0.983            0.979                      0.975      0.975 0.994
↪   0.082       0.955      379.0     190.0
2           0.979            0.974                      0.976      0.991 0.990
↪   0.093       0.944      380.0     189.0
mean        0.976            0.970                      0.967      0.973 0.993
↪   0.094       0.936          -         -
std         0.008            0.011                      0.015      0.020 0.003
↪   0.012       0.024          -         -
coef of var 0.009            0.011                      0.016      0.021 0.003
↪   0.128       0.025          -         -
```

Get Pipeline

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

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

[4]:
```
<evalml.pipelines.utils.make_pipeline.<locals>.GeneratedPipeline at 0x7f61f2fac150>
```

Get best pipeline

If we specifically want to get the best pipeline, there is a convenient access

[5]:
```
automl.best_pipeline
```

[5]:
```
<evalml.pipelines.utils.make_pipeline.<locals>.GeneratedPipeline at 0x7f626ba33490>
```

Feature Importance

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

[6]:
```
pipeline = automl.get_pipeline(1)
pipeline.fit(X, y)
pipeline.feature_importance
```

[6]:
```
                    feature  importance
0              worst texture   11.284484
1       worst concave points    8.105820
2            worst perimeter    7.989750
3               worst radius    7.872959
4                 worst area    7.851859
5        mean concave points    7.098117
6               mean texture    5.919744
7          worst smoothness    4.831183
8            worst concavity    4.665609
9                 area error    4.204315
10        compactness error    2.616379
11           worst symmetry    2.088257
12           mean concavity    2.015011
13              radius error    1.948857
14     concave points error    1.875384
15          mean compactness    1.824212
16           perimeter error    1.705293
```

(continues on next page)

(continued from previous page)

```
17          mean smoothness    1.697390
18  worst fractal dimension    1.629284
19              mean radius    1.585845
20                mean area    1.479429
21         smoothness error    1.460370
22  fractal dimension error    1.370231
23           mean perimeter    1.306400
24            texture error    1.065599
25   mean fractal dimension    1.000343
26        worst compactness    0.967447
27             mean symmetry    0.960967
28           symmetry error    0.912462
29          concavity error    0.666999
```

We can also create a bar plot of the feature importances

[7]:
```
pipeline.graph_feature_importance()
```

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

Precision-Recall Curve

For binary classification, you can view the precision-recall curve of a classifier

[8]:
```
# get the predicted probabilities associated with the "true" label
y_pred_proba = pipeline.predict_proba(X)[1]
evalml.pipelines.graph_utils.graph_precision_recall_curve(y, y_pred_proba)
```

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

ROC Curve

For binary and multiclass classification, you can view the ROC curve of a classifier

[9]:
```
# get the predicted probabilities associated with the "true" label
y_pred_proba = pipeline.predict_proba(X)[1]
evalml.pipelines.graph_utils.graph_roc_curve(y, y_pred_proba)
```

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

Confusion Matrix

For binary or multiclass classification, you can view a confusion matrix of the classifier's predictions

[10]:
```
y_pred = pipeline.predict(X)
evalml.pipelines.graph_utils.graph_confusion_matrix(y, y_pred)
```

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

Access raw results

You can also get access to all the underlying data, like this:

[11]:
```
automl.results
```

[11]:
```
{'pipeline_results': {0: {'id': 0,
   'pipeline_name': 'Mode Baseline Binary Classification Pipeline',
   'pipeline_class': evalml.pipelines.classification.baseline_binary.
↪ModeBaselineBinaryPipeline,
   'pipeline_summary': 'Baseline Classifier',
   'parameters': {'Baseline Classifier': {'strategy': 'random_weighted'}},
   'score': 0.7710601157203097,
   'high_variance_cv': False,
   'training_time': 0.03497886657714844,
   'cv_data': [{'all_objective_scores': OrderedDict([('F1',
                 0.7702265372168284),
                ('Accuracy Binary', 0.6263157894736842),
                ('Balanced Accuracy Binary', 0.5),
                ('Precision', 0.6263157894736842),
                ('AUC', 0.5),
                ('Log Loss Binary', 0.6608932451679239),
                ('MCC Binary', 0.0),
                ('# Training', 379),
                ('# Testing', 190)]),
     'score': 0.7702265372168284,
     'binary_classification_threshold': 0.5},
    {'all_objective_scores': OrderedDict([('F1', 0.7702265372168284),
                ('Accuracy Binary', 0.6263157894736842),
                ('Balanced Accuracy Binary', 0.5),
                ('Precision', 0.6263157894736842),
                ('AUC', 0.5),
                ('Log Loss Binary', 0.6608932451679239),
                ('MCC Binary', 0.0),
                ('# Training', 379),
                ('# Testing', 190)]),
     'score': 0.7702265372168284,
     'binary_classification_threshold': 0.5},
    {'all_objective_scores': OrderedDict([('F1', 0.7727272727272727),
                ('Accuracy Binary', 0.6296296296296297),
                ('Balanced Accuracy Binary', 0.5),
                ('Precision', 0.6296296296296297),
                ('AUC', 0.5),
                ('Log Loss Binary', 0.6591759924082952),
                ('MCC Binary', 0.0),
                ('# Training', 380),
                ('# Testing', 189)]),
     'score': 0.7727272727272727,
     'binary_classification_threshold': 0.5}]},
  1: {'id': 1,
   'pipeline_name': 'CatBoost Classifier w/ Simple Imputer',
   'pipeline_class': evalml.pipelines.utils.make_pipeline.<locals>.GeneratedPipeline,
```

(continues on next page)

(continued from previous page)

```
   'pipeline_summary': 'CatBoost Classifier w/ Simple Imputer',
   'parameters': {'Simple Imputer': {'impute_strategy': 'most_frequent',
     'fill_value': None},
    'CatBoost Classifier': {'n_estimators': 1000,
     'eta': 0.03,
     'max_depth': 6,
     'bootstrap_type': None}},
   'score': 0.9763329621163277,
   'high_variance_cv': False,
   'training_time': 22.760128498077393,
   'cv_data': [{'all_objective_scores': OrderedDict([('F1',
                  0.9669421487603305),
                 ('Accuracy Binary', 0.9578947368421052),
                 ('Balanced Accuracy Binary', 0.9493431175287016),
                 ('Precision', 0.9512195121951219),
                 ('AUC', 0.9945555687063559),
                 ('Log Loss Binary', 0.10583268649418161),
                 ('MCC Binary', 0.909956827190137),
                 ('# Training', 379),
                 ('# Testing', 190)]),
     'score': 0.9669421487603305,
     'binary_classification_threshold': 0.5},
    {'all_objective_scores': OrderedDict([('F1', 0.9833333333333334),
                 ('Accuracy Binary', 0.9789473684210527),
                 ('Balanced Accuracy Binary', 0.9746715587643509),
                 ('Precision', 0.9752066115702479),
                 ('AUC', 0.9943188543022844),
                 ('Log Loss Binary', 0.08186397218927995),
                 ('MCC Binary', 0.955011564828661),
                 ('# Training', 379),
                 ('# Testing', 190)]),
     'score': 0.9833333333333334,
     'binary_classification_threshold': 0.5},
    {'all_objective_scores': OrderedDict([('F1', 0.9787234042553192),
                 ('Accuracy Binary', 0.9735449735449735),
                 ('Balanced Accuracy Binary', 0.9760504201680673),
                 ('Precision', 0.9913793103448276),
                 ('AUC', 0.9899159663865547),
                 ('Log Loss Binary', 0.09296190874649334),
                 ('MCC Binary', 0.9443109474170326),
                 ('# Training', 380),
                 ('# Testing', 189)]),
     'score': 0.9787234042553192,
     'binary_classification_threshold': 0.5}]},
  2: {'id': 2,
   'pipeline_name': 'Logistic Regression Classifier w/ Simple Imputer + Standard␣
↪Scaler',
   'pipeline_class': evalml.pipelines.utils.make_pipeline.<locals>.GeneratedPipeline,
   'pipeline_summary': 'Logistic Regression Classifier w/ Simple Imputer + Standard␣
↪Scaler',
   'parameters': {'Simple Imputer': {'impute_strategy': 'most_frequent',
     'fill_value': None},
    'Logistic Regression Classifier': {'penalty': 'l2',
     'C': 1.0,
     'n_jobs': -1}},
   'score': 0.9804468499596668,
   'high_variance_cv': False,
```

(continues on next page)

(continued from previous page)

```
  'training_time': 1.0541419982910156,
  'cv_data': [{'all_objective_scores': OrderedDict([('F1',
                0.9831932773109243),
               ('Accuracy Binary', 0.9789473684210527),
               ('Balanced Accuracy Binary', 0.9775121316132087),
               ('Precision', 0.9831932773109243),
               ('AUC', 0.9936087110900698),
               ('Log Loss Binary', 0.09347817517438463),
               ('MCC Binary', 0.9550242632264173),
               ('# Training', 379),
               ('# Testing', 190)]),
    'score': 0.9831932773109243,
    'binary_classification_threshold': 0.5},
   {'all_objective_scores': OrderedDict([('F1', 0.9794238683127572),
               ('Accuracy Binary', 0.9736842105263158),
               ('Balanced Accuracy Binary', 0.9647887323943662),
               ('Precision', 0.9596774193548387),
               ('AUC', 0.9975144987572493),
               ('Log Loss Binary', 0.08320464479579018),
               ('MCC Binary', 0.9445075449666159),
               ('# Training', 379),
               ('# Testing', 190)]),
    'score': 0.9794238683127572,
    'binary_classification_threshold': 0.5},
   {'all_objective_scores': OrderedDict([('F1', 0.9787234042553192),
               ('Accuracy Binary', 0.9735449735449735),
               ('Balanced Accuracy Binary', 0.9760504201680673),
               ('Precision', 0.9913793103448276),
               ('AUC', 0.9906362545018007),
               ('Log Loss Binary', 0.09680859555948443),
               ('MCC Binary', 0.9443109474170326),
               ('# Training', 380),
               ('# Testing', 189)]),
    'score': 0.9787234042553192,
    'binary_classification_threshold': 0.5}]},
 3: {'id': 3,
  'pipeline_name': 'Random Forest Classifier w/ Simple Imputer',
  'pipeline_class': evalml.pipelines.utils.make_pipeline.<locals>.GeneratedPipeline,
  'pipeline_summary': 'Random Forest Classifier w/ Simple Imputer',
  'parameters': {'Simple Imputer': {'impute_strategy': 'most_frequent',
    'fill_value': None},
   'Random Forest Classifier': {'n_estimators': 100,
    'max_depth': 6,
    'n_jobs': -1}},
  'score': 0.9666291581686476,
  'high_variance_cv': False,
  'training_time': 1.4184563159942627,
  'cv_data': [{'all_objective_scores': OrderedDict([('F1',
                0.9543568464730291),
               ('Accuracy Binary', 0.9421052631578948),
               ('Balanced Accuracy Binary', 0.9338975026630371),
               ('Precision', 0.9426229508196722),
               ('AUC', 0.9893478518167831),
               ('Log Loss Binary', 0.13984688783161608),
               ('MCC Binary', 0.8757606542930872),
               ('# Training', 379),
               ('# Testing', 190)]),
```

(continues on next page)

(continued from previous page)

```
    'score': 0.9543568464730291,
    'binary_classification_threshold': 0.5},
   {'all_objective_scores': OrderedDict([('F1', 0.9709543568464729),
                 ('Accuracy Binary', 0.9631578947368421),
                 ('Balanced Accuracy Binary', 0.9563853710498283),
                 ('Precision', 0.9590163934426229),
                 ('AUC', 0.989347851816783),
                 ('Log Loss Binary', 0.1201072101539428),
                 ('MCC Binary', 0.9211492315750531),
                 ('# Training', 379),
                 ('# Testing', 190)]),
    'score': 0.9709543568464729,
    'binary_classification_threshold': 0.5},
   {'all_objective_scores': OrderedDict([('F1', 0.9745762711864406),
                 ('Accuracy Binary', 0.9682539682539683),
                 ('Balanced Accuracy Binary', 0.9689075630252101),
                 ('Precision', 0.9829059829059829),
                 ('AUC', 0.9927971188475391),
                 ('Log Loss Binary', 0.10765634363120971),
                 ('MCC Binary', 0.9325680982740896),
                 ('# Training', 380),
                 ('# Testing', 189)]),
    'score': 0.9745762711864406,
    'binary_classification_threshold': 0.5}]},
 4: {'id': 4,
  'pipeline_name': 'XGBoost Classifier w/ Simple Imputer',
  'pipeline_class': evalml.pipelines.utils.make_pipeline.<locals>.GeneratedPipeline,
  'pipeline_summary': 'XGBoost Classifier w/ Simple Imputer',
  'parameters': {'Simple Imputer': {'impute_strategy': 'most_frequent',
    'fill_value': None},
   'XGBoost Classifier': {'eta': 0.1,
    'max_depth': 6,
    'min_child_weight': 1,
    'n_estimators': 100}},
  'score': 0.9705772481706093,
  'high_variance_cv': False,
  'training_time': 0.42917609214782715,
  'cv_data': [{'all_objective_scores': OrderedDict([('F1',
                 0.9666666666666667),
                ('Accuracy Binary', 0.9578947368421052),
                ('Balanced Accuracy Binary', 0.9521836903775595),
                ('Precision', 0.9586776859504132),
                ('AUC', 0.9915966386554622),
                ('Log Loss Binary', 0.11449876085695762),
                ('MCC Binary', 0.9097672817424011),
                ('# Training', 379),
                ('# Testing', 190)]),
    'score': 0.9666666666666667,
    'binary_classification_threshold': 0.5},
   {'all_objective_scores': OrderedDict([('F1', 0.979253112033195),
                 ('Accuracy Binary', 0.9736842105263158),
                 ('Balanced Accuracy Binary', 0.9676293052432241),
                 ('Precision', 0.9672131147540983),
                 ('AUC', 0.9959758551307847),
                 ('Log Loss Binary', 0.07421583775339011),
                 ('MCC Binary', 0.943843520216036),
                 ('# Training', 379),
```

(continues on next page)

(continued from previous page)

```
                ('# Testing', 190)]),
    'score': 0.979253112033195,
    'binary_classification_threshold': 0.5},
   {'all_objective_scores': OrderedDict([('F1', 0.9658119658119659),
                ('Accuracy Binary', 0.9576719576719577),
                ('Balanced Accuracy Binary', 0.9605042016806722),
                ('Precision', 0.9826086956521739),
                ('AUC', 0.9885954381752701),
                ('Log Loss Binary', 0.11418110851220609),
                ('MCC Binary', 0.9112159507396058),
                ('# Training', 380),
                ('# Testing', 189)]),
    'score': 0.9658119658119659,
    'binary_classification_threshold': 0.5}]}},
 'search_order': [0, 1, 2, 3, 4]}
```

1.6.8 Regression Example

[1]:
```
import evalml
from evalml import AutoMLSearch
from evalml.demos import load_diabetes
from evalml.pipelines import PipelineBase, get_pipelines


X, y = evalml.demos.load_diabetes()

automl = AutoMLSearch(problem_type='regression', objective="R2", max_pipelines=5)

automl.search(X, y)
```

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

Optimizing for R2.
Greater score is better.

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

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

```
 Mean Baseline Regression Pipeline:          0%|          | Elapsed:00:00
 CatBoost Regressor w/ Simple Imputer:      20%|        | Elapsed:00:03
 Linear Regressor w/ Simple Imputer ...     40%|      | Elapsed:00:03
 Random Forest Regressor w/ Simple I...     60%|    | Elapsed:00:04
 XGBoost Regressor w/ Simple Imputer:       80%|   | Elapsed:00:04
 Optimization finished                      80%|   | Elapsed:00:04
```

[2]:
```
automl.rankings
```

[2]:
```
   id                                     pipeline_name     score  \
0   2  Linear Regressor w/ Simple Imputer + Standard ...  0.488703
1   1               CatBoost Regressor w/ Simple Imputer  0.446477
2   3          Random Forest Regressor w/ Simple Imputer  0.441420
3   4                XGBoost Regressor w/ Simple Imputer  0.331082
4   0                  Mean Baseline Regression Pipeline -0.004217

   high_variance_cv                                         parameters
0             False  {'Simple Imputer': {'impute_strategy': 'most_f...
1             False  {'Simple Imputer': {'impute_strategy': 'most_f...
2             False  {'Simple Imputer': {'impute_strategy': 'most_f...
3             False  {'Simple Imputer': {'impute_strategy': 'most_f...
4             False       {'Baseline Regressor': {'strategy': 'mean'}}
```

[3]:
```
automl.best_pipeline
```

[3]:
```
<evalml.pipelines.utils.make_pipeline.<locals>.GeneratedPipeline at 0x7fed1ad357d0>
```

[4]:
```
automl.get_pipeline(0)
```

[4]:
```
<evalml.pipelines.regression.baseline_regression.MeanBaselineRegressionPipeline at
↪0x7fed1bdc6950>
```

[5]:
```
automl.describe_pipeline(0)
```

```
*************************************
* Mean Baseline Regression Pipeline *
*************************************

Problem Type: Regression
Model Family: Baseline

Pipeline Steps
==============
1. Baseline Regressor
         * strategy : mean

Training
========
Training for Regression problems.
Total training time (including CV): 0.0 seconds

Cross Validation
----------
                R2  Root Mean Squared Error    MAE      MSE  MedianAE  MaxError
↪ExpVariance # Training # Testing
0           -0.007                   75.863 63.324 5755.216    57.190   186.810
↪-0.000      294.0     148.0
1           -0.000                   79.654 68.759 6344.747    67.966   193.966
↪ 0.000      295.0     147.0
2           -0.006                   75.705 65.485 5731.187    63.817   170.817
↪-0.000      295.0     147.0
mean        -0.004                   77.074 65.856 5943.717    62.991   183.864
↪-0.000           -         -
std          0.004                    2.236  2.736  347.510     5.435    11.852
↪ 0.000           -         -
coef of var -0.866                    0.029  0.042    0.058     0.086     0.064
↪-0.866           -         -
```

1.6.9 Avoiding Overfitting

The ultimate goal of machine learning is to make accurate predictions on unseen data. EvalML aims to help you build a model that will perform as you expect once it is deployed in to the real world.

One of the benefits of using EvalML to build models is that it provides data checks to ensure you are building pipelines that will perform reliably in the future. This page describes the various ways EvalML helps you avoid overfitting to your data.

[1]:
```
import evalml
```

Detecting Label Leakage

A common problem is having features that include information from your label in your training data. By default, EvalML will provide a warning when it detects this may be the case.

Let's set up a simple example to demonstrate what this looks like:

[2]:
```
import pandas as pd
from evalml.data_checks import LabelLeakageDataCheck

X = pd.DataFrame({
    "leaked_feature": [6, 6, 10, 5, 5, 11, 5, 10, 11, 4],
    "leaked_feature_2": [3, 2.5, 5, 2.5, 3, 5.5, 2, 5, 5.5, 2],
    "valid_feature": [3, 1, 3, 2, 4, 6, 1, 3, 3, 11]
})

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

label_leakage_check = LabelLeakageDataCheck()
messages = label_leakage_check.validate(X, y)
for message in messages:
    print (message)
```

```
Column 'leaked_feature' is 95.0% or more correlated with the target
Column 'leaked_feature_2' is 95.0% or more correlated with the target
```

In the example above, EvalML warned about the input features `leaked_feature` and `leak_feature_2`, which are both very closely correlated with the label we are trying to predict.

The second way to find features that may be leaking label information is to look at the top features of the model after running an AutoML search. As we can see below, the top features in our model are the 2 leaked features.

[3]:
```
automl = evalml.AutoMLSearch(
    problem_type='binary',
    max_pipelines=1,
    allowed_model_families=["linear_model"],
)

automl.search(X, y)
best_pipeline = automl.best_pipeline
best_pipeline.fit(X, y)
best_pipeline.feature_importance
```

```
Column 'leaked_feature' is 95.0% or more correlated with the target
Column 'leaked_feature_2' is 95.0% or more correlated with the target
Generating pipelines to search over...
*****************************
```

(continues on next page)

(continued from previous page)

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

Optimizing for Log Loss Binary.
Lower score is better.

Searching up to 1 pipelines.
Allowed model families: linear_model
```

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

```
 Mode Baseline Binary Classification...     0%|          | Elapsed:00:00
 Optimization finished                      0%|          | Elapsed:00:00
```

[3]:
```
            feature  importance
0    leaked_feature         0.0
1  leaked_feature_2         0.0
2     valid_feature         0.0
```

Perform cross-validation for pipeline evaluation

By default, EvalML performs 3-fold cross validation when building pipelines. This means that it evaluates each pipeline 3 times using different sets of data for training and testing. In each trial, the data used for testing has no overlap from the data used for training.

While this is a good baseline approach, you can pass your own cross validation object to be used during modeling. The cross validation object can be any of the CV methods defined in scikit-learn or use a compatible API.

For example, if we wanted to do a time series split:

[4]:
```
from sklearn.model_selection import TimeSeriesSplit

X, y = evalml.demos.load_breast_cancer()

automl = evalml.AutoMLSearch(
    problem_type='binary',
    data_split=TimeSeriesSplit(n_splits=6),
    max_pipelines=1
)

automl.search(X, y)
```

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

Optimizing for Log Loss Binary.
Lower score is better.

Searching up to 1 pipelines.
Allowed model families: catboost, random_forest, linear_model, xgboost
```

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

```
 Mode Baseline Binary Classification...     0%|          | Elapsed:00:00
 Optimization finished                      0%|          | Elapsed:00:00
```

if we describe the 1 pipeline we built, we can see the scores for each of the 6 splits as determined by the cross-validation object we provided. We can also see the number of training examples per fold increased because we were using `TimeSeriesSplit`

[5]:
```
automl.describe_pipeline(0)
```

```
*********************************************
* Mode Baseline Binary Classification Pipeline *
*********************************************

Problem Type: Binary Classification
Model Family: Baseline

Pipeline Steps
==============
1. Baseline Classifier
         * strategy : random_weighted

Training
========
Training for Binary Classification problems.
Total training time (including CV): 0.1 seconds

Cross Validation
----------
             Log Loss Binary  Accuracy Binary  Balanced Accuracy Binary    F1 ␣
↪Precision   AUC  MCC Binary # Training # Testing
0                      0.880            0.358                     0.500 0.000      0.
↪000 0.500       0.000       83.0      81.0
1                      0.697            0.469                     0.500 0.000      0.
↪000 0.500       0.000      164.0      81.0
2                      0.697            0.333                     0.500 0.000      0.
↪000 0.500       0.000      245.0      81.0
3                      0.664            0.716                     0.500 0.835      0.
↪716 0.500       0.000      326.0      81.0
4                      0.619            0.790                     0.500 0.883      0.
↪790 0.500       0.000      407.0      81.0
5                      0.611            0.741                     0.500 0.851      0.
↪741 0.500       0.000      488.0      81.0
mean                   0.695            0.568                     0.500 0.428      0.
↪374 0.500       0.000          -          -
std                    0.098            0.205                     0.000 0.469      0.
↪411 0.000       0.000          -          -
coef of var            0.141            0.361                     0.000 1.096      1.
↪097 0.000         inf          -          -
```

Detect unstable pipelines

When we perform cross validation we are trying generate an estimate of pipeline performance. EvalML does this by taking the mean of the score across the folds. If the performance across the folds varies greatly, it is indicative the the estimated value may be unreliable.

To protect the user against this, EvalML checks to see if the pipeline's performance has a variance between the different folds. EvalML triggers a warning if the "coefficient of variance" of the scores (the standard deviation divided by mean) of the pipelines scores exeeds .2.

This warning will appear in the pipeline rankings under `high_variance_cv`.

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

[6]:
```
   id                                pipeline_name     score  \
0   0  Mode Baseline Binary Classification Pipeline  0.694742

   high_variance_cv                                           parameters
0             False  {'Baseline Classifier': {'strategy': 'random_w...
```

Create holdout for model validation

EvalML offers a method to quickly create an holdout validation set. A holdout validation set is data that is not used during the process of optimizing or training the model. You should only use this validation set once you've picked the final model you'd like to use.

Below we create a holdout set of 20% of our data

[7]:
```
X, y = evalml.demos.load_breast_cancer()
X_train, X_holdout, y_train, y_holdout = evalml.preprocessing.split_data(X, y, test_
↪size=.2)
```

[8]:
```
automl = evalml.AutoMLSearch(problem_type='binary',
                             objective="f1",
                             max_pipelines=3)
automl.search(X_train, y_train)
```

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

Optimizing for F1.
Greater score is better.

Searching up to 3 pipelines.
Allowed model families: catboost, random_forest, linear_model, xgboost
```

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

```
 Mode Baseline Binary Classification...     0%|          | Elapsed:00:00
 CatBoost Classifier w/ Simple Imput...    33%|      | Elapsed:00:22
 Logistic Regression Classifier w/ S...    67%|    | Elapsed:00:23
 Optimization finished                     67%|    | Elapsed:00:23
```

then we can retrain the best pipeline on all of our training data and see how it performs compared to the estimate

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

[9]:
```
OrderedDict([('F1', 0.972972972972973)])
```

1.6.10 EvalML Components and Pipelines

EvalML searches and trains multiple machine learnining **pipelines** in order to find the best one for your data. Each pipeline is made up of various **components** that can learn from the data, transform the data and ultimately predict labels given new data. Below we'll show an example of an EvalML pipeline. You can find a more in-depth look into *components* or learn how you can construct and use your own *pipelines*.

XGBoost Pipeline

The EvalML `XGBoost Pipeline` is made up of four different components: a one-hot encoder, a missing value imputer, a feature selector and an XGBoost estimator. To initialize a pipeline you need a parameters dictionary.

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*.

[1]:
```
from evalml.demos import load_breast_cancer
from evalml.pipelines import XGBoostBinaryPipeline

X, y = load_breast_cancer()

parameters = {
        'Simple Imputer': {
            'impute_strategy': 'mean'
        },
        'RF Classifier Select From Model': {
            "percent_features": 0.5,
            "number_features": X.shape[1],
            "n_estimators": 20,
            "max_depth": 5
        },
        'XGBoost Classifier': {
            "n_estimators": 20,
            "eta": 0.5,
            "min_child_weight": 5,
            "max_depth": 10,
        }
    }

xgp = XGBoostBinaryPipeline(parameters=parameters, random_state=5)
xgp.graph()
```

[1]:

From the above graph we can see each component and its parameters. Each component takes in data and feeds it to the next. You can see more detailed information by calling `.describe()`:

[2]:
```
xgp.describe()
```

```
*****************************************
* XGBoost Binary Classification Pipeline *
*****************************************

Problem Type: Binary Classification
Model Family: XGBoost

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. XGBoost Classifier
         * eta : 0.5
         * max_depth : 10
         * min_child_weight : 5
         * n_estimators : 20
```

You can then fit and score an individual pipeline with an objective. An objective can either be a string representation of an EvalML objective or an EvalML objective class. You can find more objectives *here*.

[3]:
```
xgp.fit(X, y)
xgp.score(X, y, objectives=['f1'])
```

```
[21:12:35] WARNING: ../src/learner.cc:1061: Starting in XGBoost 1.3.0, the default␣
↪evaluation metric used with the objective 'binary:logistic' was changed from␣
↪'error' to 'logloss'. Explicitly set eval_metric if you'd like to restore the old␣
↪behavior.
```

```
/home/docs/checkouts/readthedocs.org/user_builds/feature-labs-inc-evalml/envs/v0.11.0/
↪lib/python3.7/site-packages/xgboost/sklearn.py:888: UserWarning: The use of label␣
↪encoder in XGBClassifier is deprecated and will be removed in a future release. To␣
↪remove this warning, do the following: 1) Pass option use_label_encoder=False when␣
↪constructing XGBClassifier object; and 2) Encode your labels (y) as integers␣
↪starting with 0, i.e. 0, 1, 2, ..., [num_class - 1].
  warnings.warn(label_encoder_deprecation_msg, UserWarning)
```

[3]:
```
OrderedDict([('F1', 0.9916434540389972)])
```

1.6.11 EvalML Components

From the *overview*, we see how each machine learning pipeline consists of individual components that process data before the data is ultimately sent to an estimator. Below we will describe each type of component in an EvalML pipeline.

Component Classes

Components can be split into two distinct classes: **transformers** and **estimators**.

[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  2.0  3
1  1  NaN  3
```

Transformers take in data as input and output altered data. For example, an *imputer* takes in data and outputs filled in missing data with the mean, median, or most frequent value of each column.

A transformer can fit on data and then transform it in two steps by calling `.fit()` and `.transform()` or in one step by calling `fit_transform()`.

[2]:
```
imp = SimpleImputer(impute_strategy="mean")
X = imp.fit_transform(X)

display(X)
```

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

On the other hand, an estimator fits on data (X) and labels (y) in order to take in new data as input and return the predicted label as output. Therefore, an estimator can fit on data and labels by calling `.fit()` and then predict by calling `.predict()` on new data. An example of this would be the *LogisticRegressionClassifier*. We can now see how a transformer alters data to make it easier for an estimator to learn and predict.

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

clf = LogisticRegressionClassifier()

X = X
y = [1, 0]

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

[3]:
```
0    0
1    0
dtype: int64
```

Component Types

Components can further separate into different types that serve different functionality. Below we will go over the different types of transformers and estimators.

Transformer Types

- Imputer: fills missing data

  - Ex: *SimpleImputer*
- Scaler: alters numerical data into different scales
  - Ex: *StandardScaler*
- Encoder: translates different data types
  - Ex: *OneHotEncoder*
- Feature Selection: selects most useful columns of data
  - Ex: *RFClassifierSelectFromModel*

Estimator Types

- Regressor: predicts numerical or continuous labels
  - Ex: *LinearRegressor*
- Classifier: predicts categorical or discrete labels
  - Ex: *XGBoostClassifier*

1.6.12 Custom Pipelines in EvalML

EvalML pipelines consist of modular components combining any number of transformers and an estimator. This allows you to create pipelines that fit the needs of your data to achieve the best results.

Requirements

A custom pipeline must adhere to the following requirements:

1. Inherit from the proper pipeline base class
   - Binary classification - `BinaryClassificationPipeline`
   - Multiclass classification - `MulticlassClassificationPipeline`
   - Regression - `RegressionPipeline`
2. Have a `component_graph` list as a class variable detailing the structure of the pipeline. Each component in the graph can be provided as either a string name or an instance.

Pipeline Configuration

There are a few other options to configure your custom pipeline.

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:

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

class CustomPipeline(BinaryClassificationPipeline):
    component_graph = ['Simple Imputer', 'Logistic Regression Classifier']
    custom_name = 'A custom pipeline name'

print(CustomPipeline.name)
```

```
A custom pipeline name
```

Custom Hyperparameters

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.

[2]:
```
class CustomPipeline(BinaryClassificationPipeline):
    component_graph = ['Simple Imputer', 'Logistic Regression Classifier']

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

class CustomPipeline(BinaryClassificationPipeline):
    component_graph = ['Simple Imputer', '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']},␣
↪'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']}, 'Logistic Regression␣
↪Classifier': {'penalty': ['l2'], 'C': Real(low=0.01, high=10, prior='uniform',␣
↪transform='identity')}}
```

1.6.13 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.

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.message)
```

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

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 exists. 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
```

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
```

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]
```

[1]:
```
import evalml
```

1.6.14 Default DataChecks in AutoML

By default, AutoML will run the series of data checks in `DefaultDataChecks` when `automl.search()` is called to check that inputs are valid before running the search and fitting pipelines. Currently, `DefaultDataChecks` contains `HighlyNullDataCheck()`, `IDColumnsDataCheck()`, `LabelLeakageDataCheck()`. You can see the other available data checks under `evalml/data_checks`.

If the data checks return any warnings, those warnings will be logged, but the search will continue. However, if the data checks returns any error messages, `automl.search()` will raise a `ValueError` and quit before searching. This allows users to address any issues before running the potentially time-intensive search process.

Below, we have some data that contain a lot of null values, causing `DefaultDataChecks` to log a warning "Column 'D' is 95.0% or more null" and "Column 'id' is 100.0% or more likely to be an ID column" when try to run the search

below.

[2]:
```
import pandas as pd
import numpy as np

X = pd.DataFrame(np.random.random((100, 5)), columns=["A", "B", "C", "D", "id"])
X.loc[:11, 'A'] = np.nan
X.loc[:9, 'B'] = np.nan
X.loc[:30, 'C'] = np.nan
X.loc[:95, 'D'] = np.nan
X.loc[:, 'id'] = range(100)
y = pd.Series([0,1]*50)

automl = evalml.AutoMLSearch(problem_type='binary', max_pipelines=1)
automl.search(X, y)
```

```
Column 'D' is 95.0% or more null
Column 'id' is 100.0% or more likely to be an ID column
Generating pipelines to search over...
*****************************
* Beginning pipeline search *
*****************************

Optimizing for Log Loss Binary.
Lower score is better.

Searching up to 1 pipelines.
Allowed model families: xgboost, catboost, random_forest, linear_model
```

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

```
 Mode Baseline Binary Classification...     0%|          | Elapsed:00:00
 Optimization finished                      0%|          | Elapsed:00:00
```

To access the exact warning and/or error messages our data checks returned, we can access `automl.data_check_results`.

[3]:
```
for message in automl.data_check_results:
    print (message.message)
```

```
Column 'D' is 95.0% or more null
Column 'id' is 100.0% or more likely to be an ID column
```

Using your own DataCheck with AutoML

If you'd prefer to pass in your own data check, you can do so by passing in a `DataChecks` object as the value for the `data_checks` parameter. Here, we've implemented our own custom data check which returns a list of `DataCheckError` objects if there are any columns that have zero variance.

[4]:
```
from evalml.data_checks import DataCheck, DataChecks
from evalml.data_checks.data_check_message import DataCheckError

class ZeroVarianceDataCheck(DataCheck):
    def validate(self, X, y):
```

(continues on next page)

(continued from previous page)

```
        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]
```

If we now call `search()`, our error message will be logged and a `ValueError` will be raised:

[5]:
```
data_checks = DataChecks(data_checks=[ZeroVarianceDataCheck()])

X = pd.DataFrame({'no_var': [1, 1, 1, 1, 1, 1, 1, 1, 1, 1],
                  'any_average_col': [2, 0, 1, 2, 1, 2, 0, 1, 2, 1],
                  'another_average_col': [0, 1, 2, 3, 4, 5, 6, 7, 8, 9]})
y = pd.Series([0,1,1,0,0,0,1,1,0,0])

automl = evalml.AutoMLSearch(problem_type='binary', max_pipelines=1)
automl.search(X, y, data_checks=data_checks)
```

```
Column 'no_var' has zero variance
```

```
---------------------------------------------------
ValueError                                Traceback (most recent call last)
<ipython-input-5-5f0233162825> in <module>
      7
      8 automl = evalml.AutoMLSearch(problem_type='binary', max_pipelines=1)
---> 9 automl.search(X, y, data_checks=data_checks)

~/checkouts/readthedocs.org/user_builds/feature-labs-inc-evalml/envs/v0.11.0/lib/
↪python3.7/site-packages/evalml/automl/automl_search.py in search(self, X, y,␣
↪data_checks, feature_types, raise_errors, show_iteration_plot)
    318                     logger.error(message)
    319             if any([message.message_type == DataCheckMessageType.ERROR for␣
↪message in self._data_check_results]):
-> 320                 raise ValueError("Data checks raised some warnings and/or␣
↪errors. Please see `self.data_check_results` for more information or pass␣
↪data_checks=EmptyDataChecks() to search() to disable data checking.")
    321
    322         if self.allowed_pipelines is None:

ValueError: Data checks raised some warnings and/or errors. Please see `self.
↪data_check_results` for more information or pass data_checks=EmptyDataChecks() to␣
↪search() to disable data checking.
```

Again, we can access `self.data_check_results` to help us begin to address the issues raised by data checks.

[6]:
```
for message in automl.data_check_results:
    print (message.message)
```

```
Column 'no_var' has zero variance
```

Disabling DataChecks

If you'd prefer not to run any data checks before running search, you can provide an `EmptyDataChecks` instance to `search()` instead.

[7]:
```
from evalml.data_checks import EmptyDataChecks

X = pd.DataFrame({'no_var': [1, 1, 1, 1, 1, 1, 1, 1, 1, 1],
                  'any_average_col': [2, 0, 1, 2, 1, 2, 0, 1, 2, 1],
                  'another_average_col': [0, 1, 2, 3, 4, 5, 6, 7, 8, 9]})
y = pd.Series([0,1,1,0,0,0,1,1,0,0])
automl = evalml.AutoMLSearch(problem_type='binary', max_pipelines=1)
automl.search(X, y, data_checks=EmptyDataChecks())
```

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

Optimizing for Log Loss Binary.
Lower score is better.

Searching up to 1 pipelines.
Allowed model families: xgboost, catboost, random_forest, linear_model
```

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

```
 Mode Baseline Binary Classification...     0%|          | Elapsed:00:00
 Optimization finished                      0%|          | Elapsed:00:00
```

Even though we are using the same data as above, no data checks will be run and hence, the same input data we used above will not raise an error and continue with the search process.

1.6.15 Changelog

Future Releases

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

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
- **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 precicion-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 an 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 seperate 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 *evamlm.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 hyperparemeters 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 objecitve #1
  - Added basic classifier #1
- **Documentation Changes**
  - Initialized Sphinx for docs #1

1.6.16 API Reference

Demo Datasets

load_fraud	Load credit card fraud dataset.
load_wine	Load wine dataset.
load_breast_cancer	Load breast cancer dataset.
load_diabetes	Load diabetes dataset.

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

evalml.demos.load_wine

evalml.demos.**load_wine**()

Load wine dataset. Multiclass problem

Returns X, y

Return type pd.DataFrame, pd.Series

evalml.demos.load_breast_cancer

evalml.demos.**load_breast_cancer**()

Load breast cancer dataset. Multiclass problem

Returns X, y

Return type pd.DataFrame, pd.Series

evalml.demos.load_diabetes

evalml.demos.**load_diabetes**()

Load diabetes dataset. Regression problem

Returns X, y

Return type pd.DataFrame, pd.Series

Preprocessing

Utilities to preprocess data before using evalml.

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

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

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

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

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

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

AutoML

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*, *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*)

Automated Pipeline search.

Methods

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

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*)

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.list_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

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*)

Saves AutoML object at file path

Parameters **file_path** (*str*) – location to save file

Returns None

evalml.automl.AutoMLSearch.search

AutoMLSearch.**search**(*X*, *y*, *data_checks='auto'*, *feature_types=None*, *raise_errors=True*, *show_iteration_plot=True*)

Find best classifier

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
- **raise_errors** (*boolean*) – If True, raise errors and exit search if a pipeline errors during fitting. If False, set scores for the errored pipeline to NaN and continue search. Defaults to True.
- **show_iteration_plot** (*boolean, True*) – Shows an iteration vs. score plot in Jupyter notebook. Disabled by default in non-Jupyter enviroments.
- **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

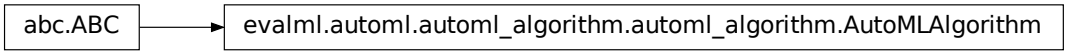
Attributes

best_pipeline	Returns an untrained instance of the best pipeline and parameters found during automl search.
data_check_results	
full_rankings	Returns a pandas.DataFrame with scoring results from all pipelines searched
has_searched	Returns *True* if search has been ran and *False* if not
rankings	Returns a pandas.DataFrame with scoring results from the highest-scoring set of parameters used with each pipeline.

AutoML Algorithm Classes

AutoMLAlgorithm	Base class for the automl algorithms which power evalml.
IterativeAlgorithm	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*, *random_state=0*)

Base class for the automl algorithms which power evalml.

Methods

__init__	This class represents an automated machine learning (AutoML) algorithm.
add_result	Register results from evaluating a pipeline
next_batch	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

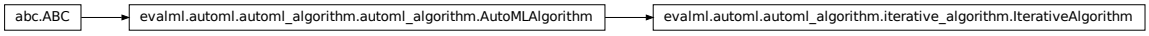
Returns a list of instances of PipelineBase subclasses, ready to be trained and evaluated.

Return type list(*PipelineBase*)

Attributes

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

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

__init__	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.
add_result	Register results from evaluating a pipeline
next_batch	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

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

Pipelines

Pipeline Base Classes

PipelineBase	Base class for all pipelines.
ClassificationPipeline	Pipeline subclass for all classification pipelines.
BinaryClassificationPipeline	Pipeline subclass for all binary classification pipelines.
MulticlassClassificationPipeline	Pipeline subclass for all multiclass classification pipelines.
RegressionPipeline	Pipeline subclass for all regression pipelines.

evalml.pipelines.PipelineBase

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

Base class for all pipelines.

Methods

__init__	Machine learning pipeline made out of transformers and a estimator.
clone	Constructs a new pipeline with the same parameters and components.
describe	Outputs pipeline details including component parameters
fit	Build a model
get_component	Returns component by name
graph	Generate an image representing the pipeline graph

Continued on next page

Table 12 – continued from previous page

graph_feature_importance	Generate a bar graph of the pipeline's feature importance
load	Loads pipeline at file path
predict	Make predictions using selected features.
save	Saves pipeline at file path
score	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**(*show_all_features=False*)

Generate a bar graph of the pipeline's feature importance

Parameters **show_all_features** (*bool, optional*) – If true, graph features with an importance value of zero. Defaults to false.

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*)

Saves pipeline at file path

Parameters **file_path** (*str*) – location to save file

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

__init__	Machine learning pipeline made out of transformers and a estimator.
clone	Constructs a new pipeline with the same parameters and components.
describe	Outputs pipeline details including component parameters

Continued on next page

Table 13 – continued from previous page

fit	Build a model
get_component	Returns component by name
graph	Generate an image representing the pipeline graph
graph_feature_importance	Generate a bar graph of the pipeline's feature importance
load	Loads pipeline at file path
predict	Make predictions using selected features.
predict_proba	Make probability estimates for labels.
save	Saves pipeline at file path
score	Evaluate model performance on objectives

evalml.pipelines.ClassificationPipeline.__init__

ClassificationPipeline.**__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.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 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.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**(*show_all_features=False*)
Generate a bar graph of the pipeline's feature importance

Parameters **show_all_features** (*bool, optional*) – If true, graph features with an importance value of zero. Defaults to false.

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*)
Saves pipeline at file path

Parameters **file_path** (*str*) – location to save file

Returns None

evalml.pipelines.ClassificationPipeline.score

ClassificationPipeline.**score**(*X*, *y*, *objectives*)
Evaluate model performance on objectives

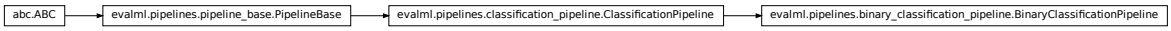
Parameters

- **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

__init__	Machine learning pipeline made out of transformers and a estimator.
clone	Constructs a new pipeline with the same parameters and components.
describe	Outputs pipeline details including component parameters
fit	Build a model
get_component	Returns component by name
graph	Generate an image representing the pipeline graph
graph_feature_importance	Generate a bar graph of the pipeline's feature importance
load	Loads pipeline at file path
predict	Make predictions using selected features.
predict_proba	Make probability estimates for labels.
save	Saves pipeline at file path
score	Evaluate model performance on objectives

evalml.pipelines.BinaryClassificationPipeline.__init__

BinaryClassificationPipeline.**__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.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 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.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**(*show_all_features=False*)
Generate a bar graph of the pipeline's feature importance

Parameters **show_all_features** (*bool, optional*) – If true, graph features with an importance value of zero. Defaults to false.

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.

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*)
Saves pipeline at file path

Parameters **file_path** (*str*) – location to save file

Returns None

evalml.pipelines.BinaryClassificationPipeline.score

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

Evaluate model performance on objectives

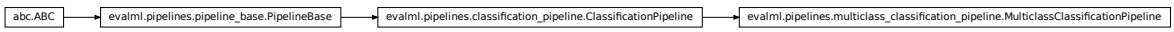
Parameters

- **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

__init__	Machine learning pipeline made out of transformers and a estimator.
clone	Constructs a new pipeline with the same parameters and components.
describe	Outputs pipeline details including component parameters
fit	Build a model
get_component	Returns component by name
graph	Generate an image representing the pipeline graph
graph_feature_importance	Generate a bar graph of the pipeline's feature importance
load	Loads pipeline at file path
predict	Make predictions using selected features.
predict_proba	Make probability estimates for labels.
save	Saves pipeline at file path
score	Evaluate model performance on objectives

evalml.pipelines.MulticlassClassificationPipeline.__init__

MulticlassClassificationPipeline.**__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.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 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.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**(*show_all_features=False*)
Generate a bar graph of the pipeline's feature importance

Parameters **show_all_features** (*bool, optional*) – If true, graph features with an importance value of zero. Defaults to false.

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*)
Saves pipeline at file path

Parameters **file_path** (*str*) – location to save file

Returns None

evalml.pipelines.MulticlassClassificationPipeline.score

MulticlassClassificationPipeline.**score**(*X*, *y*, *objectives*)
Evaluate model performance on objectives

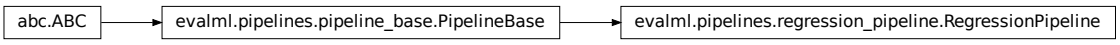
Parameters

- **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

__init__	Machine learning pipeline made out of transformers and a estimator.
clone	Constructs a new pipeline with the same parameters and components.
describe	Outputs pipeline details including component parameters
fit	Build a model
get_component	Returns component by name
graph	Generate an image representing the pipeline graph
graph_feature_importance	Generate a bar graph of the pipeline's feature importance
load	Loads pipeline at file path
predict	Make predictions using selected features.

Continued on next page

Table 16 – continued from previous page

save	Saves pipeline at file path
score	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 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**(*show_all_features=False*)
Generate a bar graph of the pipeline's feature importance

Parameters **show_all_features** (*bool, optional*) – If true, graph features with an importance value of zero. Defaults to false.

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*)

Saves pipeline at file path

Parameters **file_path** (*str*) – location to save file

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

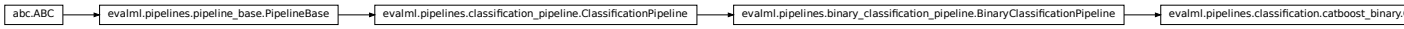
Returns ordered dictionary of objective scores

Return type dict

Classification Pipelines

CatBoostBinaryClassificationPipeline	CatBoost Pipeline for binary classification.
CatBoostMulticlassClassificationPipelin	CatBoost Pipeline for multiclass classification.
ENBinaryPipeline	Elastic Net Pipeline for binary classification problems.
ENMulticlassPipeline	Elastic Net Pipeline for multiclass classification problems.
ETBinaryClassificationPipeline	Extra Trees Pipeline for binary classification.
ETMulticlassClassificationPipeline	Extra Trees Pipeline for multiclass classification.
LogisticRegressionBinaryPipeline	Logistic Regression Pipeline for binary classification.
LogisticRegressionMulticlassPipeline	Logistic Regression Pipeline for multiclass classification.
RFBinaryClassificationPipeline	Random Forest Pipeline for binary classification.
RFMulticlassClassificationPipeline	Random Forest Pipeline for multiclass classification.
XGBoostBinaryPipeline	XGBoost Pipeline for binary classification.
XGBoostMulticlassPipeline	XGBoost Pipeline for multiclass classification.
BaselineBinaryPipeline	Baseline Pipeline for binary classification.
BaselineMulticlassPipeline	Baseline Pipeline for multiclass classification.
ModeBaselineBinaryPipeline	Mode Baseline Pipeline for binary classification.
ModeBaselineMulticlassPipeline	Mode Baseline Pipeline for multiclass classification.

evalml.pipelines.CatBoostBinaryClassificationPipeline



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

CatBoost Pipeline for binary classification. CatBoost is an open-source library and natively supports categorical features.

For more information, check out https://catboost.ai/ Note: impute_strategy must support both string and numeric data

name = 'Cat Boost Binary Classification Pipeline'

custom_name = None

summary = 'CatBoost Classifier w/ Simple Imputer'

component_graph = ['Simple Imputer', 'CatBoost Classifier']

problem_type = 'binary'

model_family = 'catboost'

hyperparameters = {'CatBoost Classifier': {'eta': Real(low=1e-06, high=1, prior='uni

custom_hyperparameters = {'Simple Imputer': {'impute_strategy': ['most_frequent']}}

default_parameters = {'CatBoost Classifier': {'bootstrap_type': None, 'eta': 0.03,

Instance attributes

feature_importance	Return importance associated with each feature.
parameters	Returns parameter dictionary for this pipeline
threshold	

Methods:

__init__	Machine learning pipeline made out of transformers and a estimator.
clone	Constructs a new pipeline with the same parameters and components.
describe	Outputs pipeline details including component parameters
fit	Build a model
get_component	Returns component by name
graph	Generate an image representing the pipeline graph
graph_feature_importance	Generate a bar graph of the pipeline's feature importance
load	Loads pipeline at file path
predict	Make predictions using selected features.
predict_proba	Make probability estimates for labels.
save	Saves pipeline at file path
score	Evaluate model performance on objectives

evalml.pipelines.CatBoostBinaryClassificationPipeline.__init__

CatBoostBinaryClassificationPipeline.**__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.CatBoostBinaryClassificationPipeline.clone

CatBoostBinaryClassificationPipeline.**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.CatBoostBinaryClassificationPipeline.describe

CatBoostBinaryClassificationPipeline.**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.CatBoostBinaryClassificationPipeline.fit

CatBoostBinaryClassificationPipeline.**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.CatBoostBinaryClassificationPipeline.get_component

CatBoostBinaryClassificationPipeline.**get_component**(*name*)

Returns component by name

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

Returns component to return

Return type Component

evalml.pipelines.CatBoostBinaryClassificationPipeline.graph

CatBoostBinaryClassificationPipeline.**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.CatBoostBinaryClassificationPipeline.graph_feature_importance

CatBoostBinaryClassificationPipeline.**graph_feature_importance**(*show_all_features=False*)
Generate a bar graph of the pipeline's feature importance

Parameters **show_all_features** (*bool, optional*) – If true, graph features with an importance value of zero. Defaults to false.

Returns plotly.Figure, a bar graph showing features and their corresponding importance

evalml.pipelines.CatBoostBinaryClassificationPipeline.load

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

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

Returns PipelineBase object

evalml.pipelines.CatBoostBinaryClassificationPipeline.predict

CatBoostBinaryClassificationPipeline.**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.CatBoostBinaryClassificationPipeline.predict_proba

CatBoostBinaryClassificationPipeline.**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.CatBoostBinaryClassificationPipeline.save

CatBoostBinaryClassificationPipeline.**save**(*file_path*)

Saves pipeline at file path

Parameters **file_path** (*str*) – location to save file

Returns None

evalml.pipelines.CatBoostBinaryClassificationPipeline.score

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

Evaluate model performance on objectives

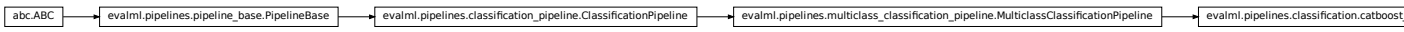
Parameters

- **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.CatBoostMulticlassClassificationPipeline



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

CatBoost Pipeline for multiclass classification. CatBoost is an open-source library and natively supports categorical features.

For more information, check out https://catboost.ai/ Note: impute_strategy must support both string and numeric data

```
name = 'Cat Boost Multiclass Classification Pipeline'

custom_name = None

summary = 'CatBoost Classifier w/ Simple Imputer'

component_graph = ['Simple Imputer', 'CatBoost Classifier']

problem_type = 'multiclass'

model_family = 'catboost'

hyperparameters = {'CatBoost Classifier': {'eta': Real(low=1e-06, high=1, prior='uni

custom_hyperparameters = {'Simple Imputer': {'impute_strategy': ['most_frequent']}}
```

```
default_parameters = {'CatBoost Classifier':  {'bootstrap_type':  None, 'eta':  0.03,
```

Instance attributes

feature_importance	Return importance associated with each feature.
parameters	Returns parameter dictionary for this pipeline

Methods:

__init__	Machine learning pipeline made out of transformers and a estimator.
clone	Constructs a new pipeline with the same parameters and components.
describe	Outputs pipeline details including component parameters
fit	Build a model
get_component	Returns component by name
graph	Generate an image representing the pipeline graph
graph_feature_importance	Generate a bar graph of the pipeline's feature importance
load	Loads pipeline at file path
predict	Make predictions using selected features.
predict_proba	Make probability estimates for labels.
save	Saves pipeline at file path
score	Evaluate model performance on objectives

evalml.pipelines.CatBoostMulticlassClassificationPipeline.__init__

CatBoostMulticlassClassificationPipeline.**__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.CatBoostMulticlassClassificationPipeline.clone

CatBoostMulticlassClassificationPipeline.**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.CatBoostMulticlassClassificationPipeline.describe

CatBoostMulticlassClassificationPipeline.**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.CatBoostMulticlassClassificationPipeline.fit

CatBoostMulticlassClassificationPipeline.**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.CatBoostMulticlassClassificationPipeline.get_component

CatBoostMulticlassClassificationPipeline.**get_component**(*name*)
Returns component by name

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

Returns component to return

Return type Component

evalml.pipelines.CatBoostMulticlassClassificationPipeline.graph

CatBoostMulticlassClassificationPipeline.**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.CatBoostMulticlassClassificationPipeline.graph_feature_importance

CatBoostMulticlassClassificationPipeline.**graph_feature_importance**(*show_all_features=False*)
Generate a bar graph of the pipeline's feature importance

Parameters **show_all_features** (*bool, optional*) – If true, graph features with an importance value of zero. Defaults to false.

Returns plotly.Figure, a bar graph showing features and their corresponding importance

evalml.pipelines.CatBoostMulticlassClassificationPipeline.load

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

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

Returns PipelineBase object

evalml.pipelines.CatBoostMulticlassClassificationPipeline.predict

CatBoostMulticlassClassificationPipeline.**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.CatBoostMulticlassClassificationPipeline.predict_proba

CatBoostMulticlassClassificationPipeline.**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.CatBoostMulticlassClassificationPipeline.save

CatBoostMulticlassClassificationPipeline.**save**(*file_path*)
Saves pipeline at file path

Parameters **file_path** (*str*) – location to save file

Returns None

evalml.pipelines.CatBoostMulticlassClassificationPipeline.score

CatBoostMulticlassClassificationPipeline.**score**(*X*, *y*, *objectives*)
Evaluate model performance on objectives

Parameters

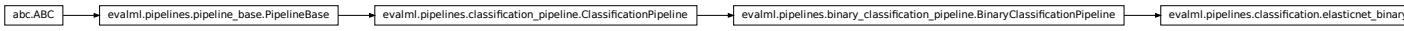
- **X** (*pd.DataFrame or np.array*) – data of shape [n_samples, n_features]
- **y** (*pd.Series*) – true labels of length [n_samples]

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

Returns ordered dictionary of objective scores

Return type dict

evalml.pipelines.ENBinaryPipeline



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

Elastic Net Pipeline for binary classification problems.

name = 'ENBinary Pipeline'

custom_name = None

summary = 'Elastic Net Classifier w/ One Hot Encoder + Simple Imputer'

component_graph = ['One Hot Encoder', 'Simple Imputer', 'Elastic Net Classifier']

problem_type = 'binary'

model_family = 'linear_model'

hyperparameters = {'Elastic Net Classifier': {'alpha': Real(low=0, high=1, prior='un

custom_hyperparameters = None

default_parameters = {'Elastic Net Classifier': {'alpha': 0.5, 'l1_ratio': 0.5, 'ma

Instance attributes

feature_importance	Return importance associated with each feature.
parameters	Returns parameter dictionary for this pipeline
threshold	

Methods:

__init__	Machine learning pipeline made out of transformers and a estimator.
clone	Constructs a new pipeline with the same parameters and components.
describe	Outputs pipeline details including component parameters
fit	Build a model
get_component	Returns component by name
graph	Generate an image representing the pipeline graph
graph_feature_importance	Generate a bar graph of the pipeline's feature importance
load	Loads pipeline at file path
predict	Make predictions using selected features.

Continued on next page

Table 23 – continued from previous page

predict_proba	Make probability estimates for labels.
save	Saves pipeline at file path
score	Evaluate model performance on objectives

evalml.pipelines.ENBinaryPipeline.__init__

ENBinaryPipeline.**__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.ENBinaryPipeline.clone

ENBinaryPipeline.**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.ENBinaryPipeline.describe

ENBinaryPipeline.**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.ENBinaryPipeline.fit

ENBinaryPipeline.**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.ENBinaryPipeline.get_component

ENBinaryPipeline.**get_component**(*name*)
Returns component by name

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

Returns component to return

Return type Component

evalml.pipelines.ENBinaryPipeline.graph

ENBinaryPipeline.**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.ENBinaryPipeline.graph_feature_importance

ENBinaryPipeline.**graph_feature_importance**(*show_all_features=False*)
Generate a bar graph of the pipeline's feature importance

Parameters **show_all_features** (*bool, optional*) – If true, graph features with an importance value of zero. Defaults to false.

Returns plotly.Figure, a bar graph showing features and their corresponding importance

evalml.pipelines.ENBinaryPipeline.load

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

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

Returns PipelineBase object

evalml.pipelines.ENBinaryPipeline.predict

ENBinaryPipeline.**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.ENBinaryPipeline.predict_proba

ENBinaryPipeline.**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.ENBinaryPipeline.save

ENBinaryPipeline.**save**(*file_path*)
Saves pipeline at file path

Parameters **file_path** (*str*) – location to save file

Returns None

evalml.pipelines.ENBinaryPipeline.score

ENBinaryPipeline.**score**(*X*, *y*, *objectives*)
Evaluate model performance on objectives

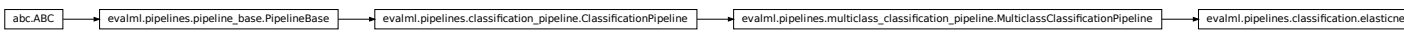
Parameters

- **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.ENMulticlassPipeline



class evalml.pipelines.**ENMulticlassPipeline**(*parameters*, *random_state=0*)
Elastic Net Pipeline for multiclass classification problems.

name = 'ENMulticlass Pipeline'

custom_name = None

summary = 'Elastic Net Classifier w/ One Hot Encoder + Simple Imputer'

component_graph = ['One Hot Encoder', 'Simple Imputer', 'Elastic Net Classifier']

problem_type = 'multiclass'

model_family = 'linear_model'

hyperparameters = {'Elastic Net Classifier': {'alpha': Real(low=0, high=1, prior='un

custom_hyperparameters = None

default_parameters = {'Elastic Net Classifier': {'alpha': 0.5, 'l1_ratio': 0.5, 'ma

Instance attributes

feature_importance	Return importance associated with each feature.
parameters	Returns parameter dictionary for this pipeline

Methods:

__init__	Machine learning pipeline made out of transformers and a estimator.
clone	Constructs a new pipeline with the same parameters and components.
describe	Outputs pipeline details including component parameters
fit	Build a model
get_component	Returns component by name
graph	Generate an image representing the pipeline graph
graph_feature_importance	Generate a bar graph of the pipeline's feature importance
load	Loads pipeline at file path
predict	Make predictions using selected features.
predict_proba	Make probability estimates for labels.
save	Saves pipeline at file path
score	Evaluate model performance on objectives

evalml.pipelines.ENMulticlassPipeline.__init__

ENMulticlassPipeline.**__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.ENMulticlassPipeline.clone

ENMulticlassPipeline.**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.ENMulticlassPipeline.describe

ENMulticlassPipeline.**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.ENMulticlassPipeline.fit

ENMulticlassPipeline.**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.ENMulticlassPipeline.get_component

ENMulticlassPipeline.**get_component**(*name*)

Returns component by name

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

Returns component to return

Return type Component

evalml.pipelines.ENMulticlassPipeline.graph

ENMulticlassPipeline.**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.ENMulticlassPipeline.graph_feature_importance

ENMulticlassPipeline.**graph_feature_importance**(*show_all_features=False*)
Generate a bar graph of the pipeline's feature importance

Parameters **show_all_features** (*bool, optional*) – If true, graph features with an importance value of zero. Defaults to false.

Returns plotly.Figure, a bar graph showing features and their corresponding importance

evalml.pipelines.ENMulticlassPipeline.load

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

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

Returns PipelineBase object

evalml.pipelines.ENMulticlassPipeline.predict

ENMulticlassPipeline.**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.ENMulticlassPipeline.predict_proba

ENMulticlassPipeline.**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.ENMulticlassPipeline.save

ENMulticlassPipeline.**save**(*file_path*)
Saves pipeline at file path

Parameters **file_path** (*str*) – location to save file

Returns None

evalml.pipelines.ENMulticlassPipeline.score

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

Evaluate model performance on objectives

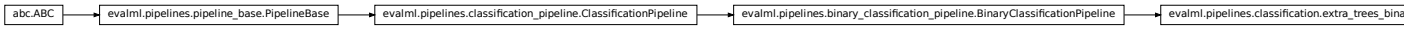
Parameters

- **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.ETBinaryClassificationPipeline



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

Extra Trees Pipeline for binary classification.

name = 'Extra Trees Binary Classification Pipeline'

custom_name = 'Extra Trees Binary Classification Pipeline'

summary = 'Extra Trees Classifier w/ One Hot Encoder + Simple Imputer'

component_graph = ['One Hot Encoder', 'Simple Imputer', 'Extra Trees Classifier']

problem_type = 'binary'

model_family = 'extra_trees'

hyperparameters = {'Extra Trees Classifier': {'max_depth': Integer(low=4, high=10, p

custom_hyperparameters = None

default_parameters = {'Extra Trees Classifier': {'max_depth': 6, 'max_features': 'a

Instance attributes

feature_importance	Return importance associated with each feature.
parameters	Returns parameter dictionary for this pipeline
threshold	

Methods:

__init__	Machine learning pipeline made out of transformers and a estimator.

Continued on next page

Table 27 – continued from previous page

clone	Constructs a new pipeline with the same parameters and components.
describe	Outputs pipeline details including component parameters
fit	Build a model
get_component	Returns component by name
graph	Generate an image representing the pipeline graph
graph_feature_importance	Generate a bar graph of the pipeline's feature importance
load	Loads pipeline at file path
predict	Make predictions using selected features.
predict_proba	Make probability estimates for labels.
save	Saves pipeline at file path
score	Evaluate model performance on objectives

evalml.pipelines.ETBinaryClassificationPipeline.__init__

ETBinaryClassificationPipeline.**__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.ETBinaryClassificationPipeline.clone

ETBinaryClassificationPipeline.**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.ETBinaryClassificationPipeline.describe

ETBinaryClassificationPipeline.**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.ETBinaryClassificationPipeline.fit

ETBinaryClassificationPipeline.**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.ETBinaryClassificationPipeline.get_component

ETBinaryClassificationPipeline.**get_component**(*name*)
Returns component by name

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

Returns component to return

Return type Component

evalml.pipelines.ETBinaryClassificationPipeline.graph

ETBinaryClassificationPipeline.**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.ETBinaryClassificationPipeline.graph_feature_importance

ETBinaryClassificationPipeline.**graph_feature_importance**(*show_all_features=False*)
Generate a bar graph of the pipeline's feature importance

Parameters **show_all_features** (*bool, optional*) – If true, graph features with an importance value of zero. Defaults to false.

Returns plotly.Figure, a bar graph showing features and their corresponding importance

evalml.pipelines.ETBinaryClassificationPipeline.load

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

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

Returns PipelineBase object

evalml.pipelines.ETBinaryClassificationPipeline.predict

ETBinaryClassificationPipeline.**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.ETBinaryClassificationPipeline.predict_proba

ETBinaryClassificationPipeline.**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.ETBinaryClassificationPipeline.save

ETBinaryClassificationPipeline.**save**(*file_path*)

Saves pipeline at file path

Parameters **file_path** (*str*) – location to save file

Returns None

evalml.pipelines.ETBinaryClassificationPipeline.score

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

Evaluate model performance on objectives

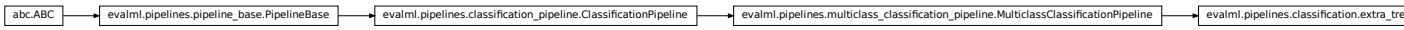
Parameters

- **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.ETMulticlassClassificationPipeline



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

Extra Trees Pipeline for multiclass classification.

name = 'Extra Trees Multiclass Classification Pipeline'

custom_name = 'Extra Trees Multiclass Classification Pipeline'

summary = 'Extra Trees Classifier w/ One Hot Encoder + Simple Imputer'

component_graph = ['One Hot Encoder', 'Simple Imputer', 'Extra Trees Classifier']

problem_type = 'multiclass'

model_family = 'extra_trees'

hyperparameters = {'Extra Trees Classifier': {'max_depth': Integer(low=4, high=10, p

custom_hyperparameters = None

default_parameters = {'Extra Trees Classifier': {'max_depth': 6, 'max_features': 'a

Instance attributes

feature_importance	Return importance associated with each feature.
parameters	Returns parameter dictionary for this pipeline

Methods:

__init__	Machine learning pipeline made out of transformers and a estimator.
clone	Constructs a new pipeline with the same parameters and components.
describe	Outputs pipeline details including component parameters
fit	Build a model
get_component	Returns component by name
graph	Generate an image representing the pipeline graph
graph_feature_importance	Generate a bar graph of the pipeline's feature importance
load	Loads pipeline at file path
predict	Make predictions using selected features.
predict_proba	Make probability estimates for labels.
save	Saves pipeline at file path
score	Evaluate model performance on objectives

evalml.pipelines.ETMulticlassClassificationPipeline.__init__

ETMulticlassClassificationPipeline.**__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.ETMulticlassClassificationPipeline.clone

ETMulticlassClassificationPipeline.**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.ETMulticlassClassificationPipeline.describe

ETMulticlassClassificationPipeline.**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.ETMulticlassClassificationPipeline.fit

ETMulticlassClassificationPipeline.**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.ETMulticlassClassificationPipeline.get_component

ETMulticlassClassificationPipeline.**get_component**(*name*)

Returns component by name

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

Returns component to return

Return type Component

evalml.pipelines.ETMulticlassClassificationPipeline.graph

ETMulticlassClassificationPipeline.**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.ETMulticlassClassificationPipeline.graph_feature_importance

ETMulticlassClassificationPipeline.**graph_feature_importance**(*show_all_features=False*)

Generate a bar graph of the pipeline's feature importance

Parameters **show_all_features** (*bool, optional*) – If true, graph features with an importance value of zero. Defaults to false.

Returns plotly.Figure, a bar graph showing features and their corresponding importance

evalml.pipelines.ETMulticlassClassificationPipeline.load

static ETMulticlassClassificationPipeline.**load**(*file_path*)

Loads pipeline at file path

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

Returns PipelineBase object

evalml.pipelines.ETMulticlassClassificationPipeline.predict

ETMulticlassClassificationPipeline.**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.ETMulticlassClassificationPipeline.predict_proba

ETMulticlassClassificationPipeline.**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.ETMulticlassClassificationPipeline.save

ETMulticlassClassificationPipeline.**save**(*file_path*)

Saves pipeline at file path

Parameters **file_path** (*str*) – location to save file

Returns None

evalml.pipelines.ETMulticlassClassificationPipeline.score

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

Evaluate model performance on objectives

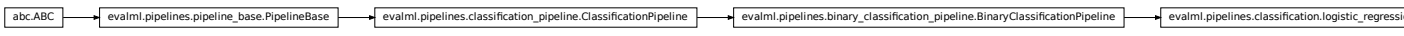
Parameters

- **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.LogisticRegressionBinaryPipeline



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

Logistic Regression Pipeline for binary classification.

```
name = 'Logistic Regression Binary Pipeline'
custom_name = None
summary = 'Logistic Regression Classifier w/ Simple Imputer + One Hot Encoder + Standa
component_graph = ['Simple Imputer', 'One Hot Encoder', 'Standard Scaler', 'Logistic R
problem_type = 'binary'
model_family = 'linear_model'
```

```
hyperparameters = {'Logistic Regression Classifier':  {'C': Real(low=0.01, high=10, pr
custom_hyperparameters = None
default_parameters = {'Logistic Regression Classifier':  {'C': 1.0, 'n_jobs':  -1, 'pe
```

Instance attributes

feature_importance	Return importance associated with each feature.
parameters	Returns parameter dictionary for this pipeline
threshold	

Methods:

__init__	Machine learning pipeline made out of transformers and a estimator.
clone	Constructs a new pipeline with the same parameters and components.
describe	Outputs pipeline details including component parameters
fit	Build a model
get_component	Returns component by name
graph	Generate an image representing the pipeline graph
graph_feature_importance	Generate a bar graph of the pipeline's feature importance
load	Loads pipeline at file path
predict	Make predictions using selected features.
predict_proba	Make probability estimates for labels.
save	Saves pipeline at file path
score	Evaluate model performance on objectives

evalml.pipelines.LogisticRegressionBinaryPipeline.__init__

LogisticRegressionBinaryPipeline.**__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.LogisticRegressionBinaryPipeline.clone

LogisticRegressionBinaryPipeline.**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.LogisticRegressionBinaryPipeline.describe

LogisticRegressionBinaryPipeline.**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.LogisticRegressionBinaryPipeline.fit

LogisticRegressionBinaryPipeline.**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.LogisticRegressionBinaryPipeline.get_component

LogisticRegressionBinaryPipeline.**get_component**(*name*)

Returns component by name

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

Returns component to return

Return type Component

evalml.pipelines.LogisticRegressionBinaryPipeline.graph

LogisticRegressionBinaryPipeline.**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.LogisticRegressionBinaryPipeline.graph_feature_importance

LogisticRegressionBinaryPipeline.**graph_feature_importance**(*show_all_features=False*)
Generate a bar graph of the pipeline's feature importance

Parameters **show_all_features** (*bool, optional*) – If true, graph features with an importance value of zero. Defaults to false.

Returns plotly.Figure, a bar graph showing features and their corresponding importance

evalml.pipelines.LogisticRegressionBinaryPipeline.load

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

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

Returns PipelineBase object

evalml.pipelines.LogisticRegressionBinaryPipeline.predict

LogisticRegressionBinaryPipeline.**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.LogisticRegressionBinaryPipeline.predict_proba

LogisticRegressionBinaryPipeline.**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.LogisticRegressionBinaryPipeline.save

LogisticRegressionBinaryPipeline.**save**(*file_path*)
Saves pipeline at file path

Parameters **file_path** (*str*) – location to save file

Returns None

evalml.pipelines.LogisticRegressionBinaryPipeline.score

LogisticRegressionBinaryPipeline.**score**(*X*, *y*, *objectives*)
Evaluate model performance on objectives

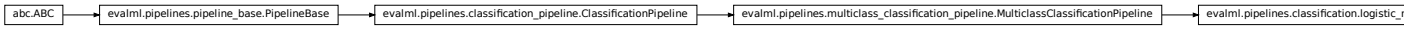
Parameters

- **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.LogisticRegressionMulticlassPipeline



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

Logistic Regression Pipeline for multiclass classification.

name = 'Logistic Regression Multiclass Pipeline'

custom_name = None

summary = 'Logistic Regression Classifier w/ One Hot Encoder + Simple Imputer + Standa

component_graph = ['One Hot Encoder', 'Simple Imputer', 'Standard Scaler', 'Logistic R

problem_type = 'multiclass'

model_family = 'linear_model'

hyperparameters = {'Logistic Regression Classifier': {'C': Real(low=0.01, high=10, pr

custom_hyperparameters = None

default_parameters = {'Logistic Regression Classifier': {'C': 1.0, 'n_jobs': -1, 'pe

Instance attributes

feature_importance	Return importance associated with each feature.
parameters	Returns parameter dictionary for this pipeline

Methods:

__init__	Machine learning pipeline made out of transformers and a estimator.
clone	Constructs a new pipeline with the same parameters and components.

Continued on next page

Table 33 – continued from previous page

describe	Outputs pipeline details including component parameters
fit	Build a model
get_component	Returns component by name
graph	Generate an image representing the pipeline graph
graph_feature_importance	Generate a bar graph of the pipeline's feature importance
load	Loads pipeline at file path
predict	Make predictions using selected features.
predict_proba	Make probability estimates for labels.
save	Saves pipeline at file path
score	Evaluate model performance on objectives

evalml.pipelines.LogisticRegressionMulticlassPipeline.__init__

LogisticRegressionMulticlassPipeline.**__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.LogisticRegressionMulticlassPipeline.clone

LogisticRegressionMulticlassPipeline.**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.LogisticRegressionMulticlassPipeline.describe

LogisticRegressionMulticlassPipeline.**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.LogisticRegressionMulticlassPipeline.fit

LogisticRegressionMulticlassPipeline.**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.LogisticRegressionMulticlassPipeline.get_component

LogisticRegressionMulticlassPipeline.**get_component**(*name*)
Returns component by name

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

Returns component to return

Return type Component

evalml.pipelines.LogisticRegressionMulticlassPipeline.graph

LogisticRegressionMulticlassPipeline.**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.LogisticRegressionMulticlassPipeline.graph_feature_importance

LogisticRegressionMulticlassPipeline.**graph_feature_importance**(*show_all_features=False*)
Generate a bar graph of the pipeline's feature importance

Parameters **show_all_features** (*bool, optional*) – If true, graph features with an importance value of zero. Defaults to false.

Returns plotly.Figure, a bar graph showing features and their corresponding importance

evalml.pipelines.LogisticRegressionMulticlassPipeline.load

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

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

Returns PipelineBase object

evalml.pipelines.LogisticRegressionMulticlassPipeline.predict

LogisticRegressionMulticlassPipeline.**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.LogisticRegressionMulticlassPipeline.predict_proba

LogisticRegressionMulticlassPipeline.**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.LogisticRegressionMulticlassPipeline.save

LogisticRegressionMulticlassPipeline.**save**(*file_path*)

Saves pipeline at file path

Parameters **file_path** (*str*) – location to save file

Returns None

evalml.pipelines.LogisticRegressionMulticlassPipeline.score

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

Evaluate model performance on objectives

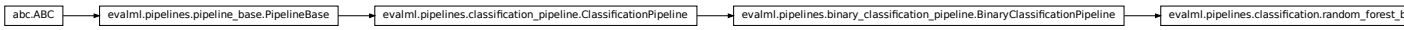
Parameters

- **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.RFBinaryClassificationPipeline



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

Random Forest Pipeline for binary classification.

name = 'Random Forest Binary Classification Pipeline'

custom_name = 'Random Forest Binary Classification Pipeline'

summary = 'Random Forest Classifier w/ Simple Imputer + One Hot Encoder'

component_graph = ['Simple Imputer', 'One Hot Encoder', 'Random Forest Classifier']

problem_type = 'binary'

model_family = 'random_forest'

hyperparameters = {'One Hot Encoder': {}, 'Random Forest Classifier': {'max_depth':

custom_hyperparameters = None

default_parameters = {'One Hot Encoder': {'categories': None, 'drop': None, 'handle_

Instance attributes

feature_importance	Return importance associated with each feature.
parameters	Returns parameter dictionary for this pipeline
threshold	

Methods:

__init__	Machine learning pipeline made out of transformers and a estimator.
clone	Constructs a new pipeline with the same parameters and components.
describe	Outputs pipeline details including component parameters
fit	Build a model
get_component	Returns component by name
graph	Generate an image representing the pipeline graph
graph_feature_importance	Generate a bar graph of the pipeline's feature importance
load	Loads pipeline at file path
predict	Make predictions using selected features.
predict_proba	Make probability estimates for labels.
save	Saves pipeline at file path
score	Evaluate model performance on objectives

evalml.pipelines.RFBinaryClassificationPipeline.__init__

RFBinaryClassificationPipeline.**__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.RFBinaryClassificationPipeline.clone

RFBinaryClassificationPipeline.**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.RFBinaryClassificationPipeline.describe

RFBinaryClassificationPipeline.**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.RFBinaryClassificationPipeline.fit

RFBinaryClassificationPipeline.**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.RFBinaryClassificationPipeline.get_component

RFBinaryClassificationPipeline.**get_component**(*name*)
Returns component by name

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

Returns component to return

Return type Component

evalml.pipelines.RFBinaryClassificationPipeline.graph

RFBinaryClassificationPipeline.**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.RFBinaryClassificationPipeline.graph_feature_importance

RFBinaryClassificationPipeline.**graph_feature_importance**(*show_all_features=False*)
Generate a bar graph of the pipeline's feature importance

Parameters **show_all_features** (*bool, optional*) – If true, graph features with an importance value of zero. Defaults to false.

Returns plotly.Figure, a bar graph showing features and their corresponding importance

evalml.pipelines.RFBinaryClassificationPipeline.load

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

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

Returns PipelineBase object

evalml.pipelines.RFBinaryClassificationPipeline.predict

RFBinaryClassificationPipeline.**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.RFBinaryClassificationPipeline.predict_proba

RFBinaryClassificationPipeline.**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.RFBinaryClassificationPipeline.save

RFBinaryClassificationPipeline.**save**(*file_path*)

Saves pipeline at file path

Parameters **file_path** (*str*) – location to save file

Returns None

evalml.pipelines.RFBinaryClassificationPipeline.score

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

Evaluate model performance on objectives

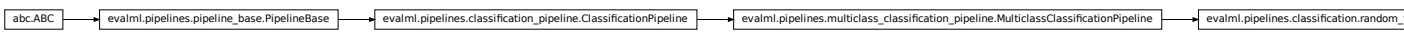
Parameters

- **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.RFMulticlassClassificationPipeline



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

Random Forest Pipeline for multiclass classification.

```
name = 'Random Forest Multiclass Classification Pipeline'
custom_name = 'Random Forest Multiclass Classification Pipeline'
summary = 'Random Forest Classifier w/ One Hot Encoder + Simple Imputer'
component_graph = ['One Hot Encoder', 'Simple Imputer', 'Random Forest Classifier']
problem_type = 'multiclass'
model_family = 'random_forest'
```

```
hyperparameters = {'One Hot Encoder': {}, 'Random Forest Classifier': {'max_depth':
custom_hyperparameters = None
default_parameters = {'One Hot Encoder': {'categories': None, 'drop': None, 'handle
```

Instance attributes

feature_importance	Return importance associated with each feature.
parameters	Returns parameter dictionary for this pipeline

Methods:

__init__	Machine learning pipeline made out of transformers and a estimator.
clone	Constructs a new pipeline with the same parameters and components.
describe	Outputs pipeline details including component parameters
fit	Build a model
get_component	Returns component by name
graph	Generate an image representing the pipeline graph
graph_feature_importance	Generate a bar graph of the pipeline's feature importance
load	Loads pipeline at file path
predict	Make predictions using selected features.
predict_proba	Make probability estimates for labels.
save	Saves pipeline at file path
score	Evaluate model performance on objectives

evalml.pipelines.RFMulticlassClassificationPipeline.__init__

RFMulticlassClassificationPipeline.**__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.RFMulticlassClassificationPipeline.clone

RFMulticlassClassificationPipeline.**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.RFMulticlassClassificationPipeline.describe

RFMulticlassClassificationPipeline.**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.RFMulticlassClassificationPipeline.fit

RFMulticlassClassificationPipeline.**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.RFMulticlassClassificationPipeline.get_component

RFMulticlassClassificationPipeline.**get_component**(*name*)

Returns component by name

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

Returns component to return

Return type Component

evalml.pipelines.RFMulticlassClassificationPipeline.graph

RFMulticlassClassificationPipeline.**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.RFMulticlassClassificationPipeline.graph_feature_importance

RFMulticlassClassificationPipeline.**graph_feature_importance**(*show_all_features=False*)
Generate a bar graph of the pipeline's feature importance

Parameters **show_all_features** (*bool, optional*) – If true, graph features with an importance value of zero. Defaults to false.

Returns plotly.Figure, a bar graph showing features and their corresponding importance

evalml.pipelines.RFMulticlassClassificationPipeline.load

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

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

Returns PipelineBase object

evalml.pipelines.RFMulticlassClassificationPipeline.predict

RFMulticlassClassificationPipeline.**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.RFMulticlassClassificationPipeline.predict_proba

RFMulticlassClassificationPipeline.**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.RFMulticlassClassificationPipeline.save

RFMulticlassClassificationPipeline.**save**(*file_path*)
Saves pipeline at file path

Parameters **file_path** (*str*) – location to save file

Returns None

evalml.pipelines.RFMulticlassClassificationPipeline.score

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

Evaluate model performance on objectives

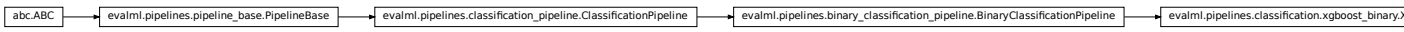
Parameters

- **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.XGBoostBinaryPipeline



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

XGBoost Pipeline for binary classification.

```
name = 'XGBoost Binary Classification Pipeline'
custom_name = 'XGBoost Binary Classification Pipeline'
summary = 'XGBoost Classifier w/ Simple Imputer + One Hot Encoder'
component_graph = ['Simple Imputer', 'One Hot Encoder', 'XGBoost Classifier']
problem_type = 'binary'
model_family = 'xgboost'
hyperparameters = {'One Hot Encoder': {}, 'Simple Imputer': {'impute_strategy': ['m
custom_hyperparameters = None
default_parameters = {'One Hot Encoder': {'categories': None, 'drop': None, 'handle
```

Instance attributes

feature_importance	Return importance associated with each feature.
parameters	Returns parameter dictionary for this pipeline
threshold	

Methods:

__init__	Machine learning pipeline made out of transformers and a estimator.

Continued on next page

Table 39 – continued from previous page

clone	Constructs a new pipeline with the same parameters and components.
describe	Outputs pipeline details including component parameters
fit	Build a model
get_component	Returns component by name
graph	Generate an image representing the pipeline graph
graph_feature_importance	Generate a bar graph of the pipeline's feature importance
load	Loads pipeline at file path
predict	Make predictions using selected features.
predict_proba	Make probability estimates for labels.
save	Saves pipeline at file path
score	Evaluate model performance on objectives

evalml.pipelines.XGBoostBinaryPipeline.__init__

XGBoostBinaryPipeline.**__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.XGBoostBinaryPipeline.clone

XGBoostBinaryPipeline.**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.XGBoostBinaryPipeline.describe

XGBoostBinaryPipeline.**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.XGBoostBinaryPipeline.fit

XGBoostBinaryPipeline.**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.XGBoostBinaryPipeline.get_component

XGBoostBinaryPipeline.**get_component**(*name*)
Returns component by name

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

Returns component to return

Return type Component

evalml.pipelines.XGBoostBinaryPipeline.graph

XGBoostBinaryPipeline.**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.XGBoostBinaryPipeline.graph_feature_importance

XGBoostBinaryPipeline.**graph_feature_importance**(*show_all_features=False*)
Generate a bar graph of the pipeline's feature importance

Parameters **show_all_features** (*bool, optional*) – If true, graph features with an importance value of zero. Defaults to false.

Returns plotly.Figure, a bar graph showing features and their corresponding importance

evalml.pipelines.XGBoostBinaryPipeline.load

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

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

Returns PipelineBase object

evalml.pipelines.XGBoostBinaryPipeline.predict

XGBoostBinaryPipeline.**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.XGBoostBinaryPipeline.predict_proba

XGBoostBinaryPipeline.**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.XGBoostBinaryPipeline.save

XGBoostBinaryPipeline.**save**(*file_path*)
Saves pipeline at file path

Parameters **file_path** (*str*) – location to save file

Returns None

evalml.pipelines.XGBoostBinaryPipeline.score

XGBoostBinaryPipeline.**score**(*X*, *y*, *objectives*)
Evaluate model performance on objectives

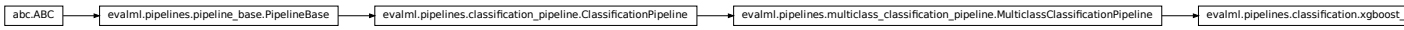
Parameters

- **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.XGBoostMulticlassPipeline



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

XGBoost Pipeline for multiclass classification.

name = 'XGBoost Multiclass Classification Pipeline'

custom_name = 'XGBoost Multiclass Classification Pipeline'

summary = 'XGBoost Classifier w/ One Hot Encoder + Simple Imputer'

component_graph = ['One Hot Encoder', 'Simple Imputer', 'XGBoost Classifier']

problem_type = 'multiclass'

model_family = 'xgboost'

hyperparameters = {'One Hot Encoder': {}, 'Simple Imputer': {'impute_strategy': ['m

custom_hyperparameters = None

default_parameters = {'One Hot Encoder': {'categories': None, 'drop': None, 'handle_

Instance attributes

feature_importance	Return importance associated with each feature.
parameters	Returns parameter dictionary for this pipeline

Methods:

__init__	Machine learning pipeline made out of transformers and a estimator.
clone	Constructs a new pipeline with the same parameters and components.
describe	Outputs pipeline details including component parameters
fit	Build a model
get_component	Returns component by name
graph	Generate an image representing the pipeline graph
graph_feature_importance	Generate a bar graph of the pipeline's feature importance
load	Loads pipeline at file path
predict	Make predictions using selected features.
predict_proba	Make probability estimates for labels.
save	Saves pipeline at file path
score	Evaluate model performance on objectives

evalml.pipelines.XGBoostMulticlassPipeline.__init__

XGBoostMulticlassPipeline.**__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.XGBoostMulticlassPipeline.clone

XGBoostMulticlassPipeline.**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.XGBoostMulticlassPipeline.describe

XGBoostMulticlassPipeline.**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.XGBoostMulticlassPipeline.fit

XGBoostMulticlassPipeline.**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.XGBoostMulticlassPipeline.get_component

XGBoostMulticlassPipeline.**get_component**(*name*)
Returns component by name

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

Returns component to return

Return type Component

evalml.pipelines.XGBoostMulticlassPipeline.graph

XGBoostMulticlassPipeline.**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.XGBoostMulticlassPipeline.graph_feature_importance

XGBoostMulticlassPipeline.**graph_feature_importance**(*show_all_features=False*)
Generate a bar graph of the pipeline's feature importance

Parameters **show_all_features** (*bool, optional*) – If true, graph features with an importance value of zero. Defaults to false.

Returns plotly.Figure, a bar graph showing features and their corresponding importance

evalml.pipelines.XGBoostMulticlassPipeline.load

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

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

Returns PipelineBase object

evalml.pipelines.XGBoostMulticlassPipeline.predict

XGBoostMulticlassPipeline.**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.XGBoostMulticlassPipeline.predict_proba

XGBoostMulticlassPipeline.**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.XGBoostMulticlassPipeline.save

XGBoostMulticlassPipeline.**save**(*file_path*)
Saves pipeline at file path

Parameters **file_path** (*str*) – location to save file

Returns None

evalml.pipelines.XGBoostMulticlassPipeline.score

XGBoostMulticlassPipeline.**score**(*X*, *y*, *objectives*)
Evaluate model performance on objectives

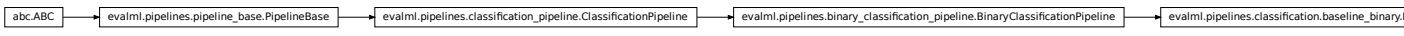
Parameters

- **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.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 importance associated with each feature.
parameters	Returns parameter dictionary for this pipeline
threshold	

Methods:

__init__	Machine learning pipeline made out of transformers and a estimator.
clone	Constructs a new pipeline with the same parameters and components.
describe	Outputs pipeline details including component parameters
fit	Build a model
get_component	Returns component by name
graph	Generate an image representing the pipeline graph
graph_feature_importance	Generate a bar graph of the pipeline's feature importance
load	Loads pipeline at file path
predict	Make predictions using selected features.
predict_proba	Make probability estimates for labels.
save	Saves pipeline at file path
score	Evaluate model performance on objectives

evalml.pipelines.BaselineBinaryPipeline.__init__

BaselineBinaryPipeline.**__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.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 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.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**(*show_all_features=False*)
Generate a bar graph of the pipeline's feature importance

Parameters **show_all_features** (*bool, optional*) – If true, graph features with an importance value of zero. Defaults to false.

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.

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*)
Saves pipeline at file path

Parameters **file_path** (*str*) – location to save file

Returns None

evalml.pipelines.BaselineBinaryPipeline.score

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

Evaluate model performance on objectives

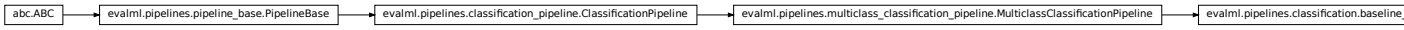
Parameters

- **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

feature_importance	Return importance associated with each feature.
parameters	Returns parameter dictionary for this pipeline

Methods:

__init__	Machine learning pipeline made out of transformers and a estimator.
clone	Constructs a new pipeline with the same parameters and components.

Continued on next page

Table 45 – continued from previous page

describe	Outputs pipeline details including component parameters
fit	Build a model
get_component	Returns component by name
graph	Generate an image representing the pipeline graph
graph_feature_importance	Generate a bar graph of the pipeline's feature importance
load	Loads pipeline at file path
predict	Make predictions using selected features.
predict_proba	Make probability estimates for labels.
save	Saves pipeline at file path
score	Evaluate model performance on objectives

evalml.pipelines.BaselineMulticlassPipeline.__init__

BaselineMulticlassPipeline.**__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.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 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.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**(*show_all_features=False*)
Generate a bar graph of the pipeline's feature importance

Parameters **show_all_features** (*bool, optional*) – If true, graph features with an importance value of zero. Defaults to false.

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*)

Saves pipeline at file path

Parameters **file_path** (*str*) – location to save file

Returns None

evalml.pipelines.BaselineMulticlassPipeline.score

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

Evaluate model performance on objectives

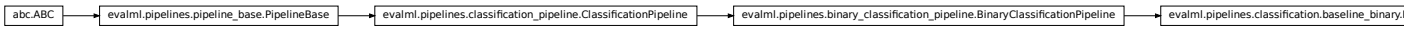
Parameters

- **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

feature_importance	Return importance associated with each feature.
parameters	Returns parameter dictionary for this pipeline
threshold	

Methods:

__init__	Machine learning pipeline made out of transformers and a estimator.
clone	Constructs a new pipeline with the same parameters and components.
describe	Outputs pipeline details including component parameters
fit	Build a model
get_component	Returns component by name
graph	Generate an image representing the pipeline graph
graph_feature_importance	Generate a bar graph of the pipeline's feature importance
load	Loads pipeline at file path
predict	Make predictions using selected features.
predict_proba	Make probability estimates for labels.
save	Saves pipeline at file path
score	Evaluate model performance on objectives

evalml.pipelines.ModeBaselineBinaryPipeline.__init__

ModeBaselineBinaryPipeline.**__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.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 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.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**(*show_all_features=False*)

Generate a bar graph of the pipeline's feature importance

Parameters **show_all_features** (*bool, optional*) – If true, graph features with an importance value of zero. Defaults to false.

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.

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*)

Saves pipeline at file path

Parameters **file_path** (*str*) – location to save file

Returns None

evalml.pipelines.ModeBaselineBinaryPipeline.score

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

Evaluate model performance on objectives

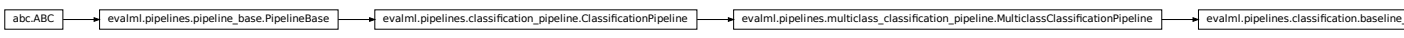
Parameters

- **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

feature_importance	Return importance associated with each feature.
parameters	Returns parameter dictionary for this pipeline

Methods:

__init__	Machine learning pipeline made out of transformers and a estimator.
clone	Constructs a new pipeline with the same parameters and components.
describe	Outputs pipeline details including component parameters
fit	Build a model
get_component	Returns component by name
graph	Generate an image representing the pipeline graph
graph_feature_importance	Generate a bar graph of the pipeline's feature importance
load	Loads pipeline at file path
predict	Make predictions using selected features.
predict_proba	Make probability estimates for labels.
save	Saves pipeline at file path
score	Evaluate model performance on objectives

evalml.pipelines.ModeBaselineMulticlassPipeline.__init__

ModeBaselineMulticlassPipeline.**__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.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 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.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**(*show_all_features=False*)
Generate a bar graph of the pipeline's feature importance

Parameters **show_all_features** (*bool, optional*) – If true, graph features with an importance value of zero. Defaults to false.

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*)
Saves pipeline at file path

Parameters **file_path** (*str*) – location to save file

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

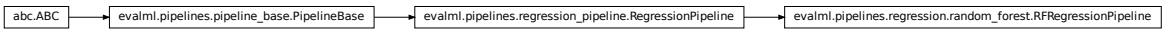
Returns ordered dictionary of objective scores

Return type dict

Regression Pipelines

RFRegressionPipeline	Random Forest Pipeline for regression problems.
CatBoostRegressionPipeline	CatBoost Pipeline for regression problems.
ENRegressionPipeline	Elastic Net Pipeline for regression problems.
ETRegressionPipeline	Extra Trees Pipeline for regression problems.
LinearRegressionPipeline	Linear Regression Pipeline for regression problems.
XGBoostRegressionPipeline	XGBoost Pipeline for regression problems.
BaselineRegressionPipeline	Baseline Pipeline for regression problems.
MeanBaselineRegressionPipeline	Baseline Pipeline for regression problems.

evalml.pipelines.RFRegressionPipeline



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

Random Forest Pipeline for regression problems.

```
name = 'Random Forest Regression Pipeline'

custom_name = 'Random Forest Regression Pipeline'

summary = 'Random Forest Regressor w/ One Hot Encoder + Simple Imputer'

component_graph = ['One Hot Encoder', 'Simple Imputer', 'Random Forest Regressor']

problem_type = 'regression'

model_family = 'random_forest'

hyperparameters = {'One Hot Encoder':  {}, 'Random Forest Regressor':  {'max_depth':

custom_hyperparameters = None

default_parameters = {'One Hot Encoder':  {'categories':  None, 'drop':  None, 'handle_
```

Instance attributes

feature_importance	Return importance associated with each feature.
parameters	Returns parameter dictionary for this pipeline

Methods:

__init__	Machine learning pipeline made out of transformers and a estimator.
clone	Constructs a new pipeline with the same parameters and components.
describe	Outputs pipeline details including component parameters
fit	Build a model
get_component	Returns component by name
graph	Generate an image representing the pipeline graph
graph_feature_importance	Generate a bar graph of the pipeline's feature importance
load	Loads pipeline at file path
predict	Make predictions using selected features.
save	Saves pipeline at file path
score	Evaluate model performance on current and additional objectives

evalml.pipelines.RFRegressionPipeline.__init__

RFRegressionPipeline.**__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.RFRegressionPipeline.clone

RFRegressionPipeline.**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.RFRegressionPipeline.describe

RFRegressionPipeline.**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.RFRegressionPipeline.fit

RFRegressionPipeline.**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.RFRegressionPipeline.get_component

RFRegressionPipeline.**get_component**(*name*)
Returns component by name

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

Returns component to return

Return type Component

evalml.pipelines.RFRegressionPipeline.graph

RFRegressionPipeline.**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.RFRegressionPipeline.graph_feature_importance

RFRegressionPipeline.**graph_feature_importance**(*show_all_features=False*)
Generate a bar graph of the pipeline's feature importance

Parameters **show_all_features** (*bool, optional*) – If true, graph features with an importance value of zero. Defaults to false.

Returns plotly.Figure, a bar graph showing features and their corresponding importance

evalml.pipelines.RFRegressionPipeline.load

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

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

Returns PipelineBase object

evalml.pipelines.RFRegressionPipeline.predict

RFRegressionPipeline.**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.RFRegressionPipeline.save

RFRegressionPipeline.**save**(*file_path*)
Saves pipeline at file path

Parameters **file_path** (*str*) – location to save file

Returns None

evalml.pipelines.RFRegressionPipeline.score

RFRegressionPipeline.**score**(*X*, *y*, *objectives*)
Evaluate model performance on current and additional objectives

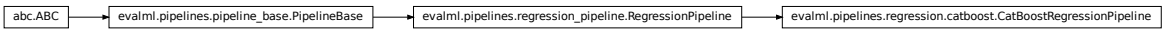
Parameters

- **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.CatBoostRegressionPipeline



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

CatBoost Pipeline for regression problems. CatBoost is an open-source library and natively supports categorical features.

For more information, check out https://catboost.ai/

Note: impute_strategy must support both string and numeric data

name = 'Cat Boost Regression Pipeline'

custom_name = None

summary = 'CatBoost Regressor w/ Simple Imputer'

component_graph = ['Simple Imputer', 'CatBoost Regressor']

problem_type = 'regression'

model_family = 'catboost'

hyperparameters = {'CatBoost Regressor': {'eta': Real(low=1e-06, high=1, prior='unif

custom_hyperparameters = {'Simple Imputer': {'impute_strategy': ['most_frequent']}}

default_parameters = {'CatBoost Regressor': {'bootstrap_type': None, 'eta': 0.03, '

Instance attributes

feature_importance	Return importance associated with each feature.
parameters	Returns parameter dictionary for this pipeline

Methods:

__init__	Machine learning pipeline made out of transformers and a estimator.
clone	Constructs a new pipeline with the same parameters and components.
describe	Outputs pipeline details including component parameters
fit	Build a model
get_component	Returns component by name
graph	Generate an image representing the pipeline graph
graph_feature_importance	Generate a bar graph of the pipeline's feature importance
load	Loads pipeline at file path
predict	Make predictions using selected features.
save	Saves pipeline at file path

Continued on next page

Table 54 – continued from previous page

score	Evaluate model performance on current and additional objectives

evalml.pipelines.CatBoostRegressionPipeline.__init__

CatBoostRegressionPipeline.**__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.CatBoostRegressionPipeline.clone

CatBoostRegressionPipeline.**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.CatBoostRegressionPipeline.describe

CatBoostRegressionPipeline.**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.CatBoostRegressionPipeline.fit

CatBoostRegressionPipeline.**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.CatBoostRegressionPipeline.get_component

CatBoostRegressionPipeline.**get_component**(*name*)

Returns component by name

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

Returns component to return

Return type Component

evalml.pipelines.CatBoostRegressionPipeline.graph

CatBoostRegressionPipeline.**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.CatBoostRegressionPipeline.graph_feature_importance

CatBoostRegressionPipeline.**graph_feature_importance**(*show_all_features=False*)

Generate a bar graph of the pipeline's feature importance

Parameters **show_all_features** (*bool, optional*) – If true, graph features with an importance value of zero. Defaults to false.

Returns plotly.Figure, a bar graph showing features and their corresponding importance

evalml.pipelines.CatBoostRegressionPipeline.load

static CatBoostRegressionPipeline.**load**(*file_path*)

Loads pipeline at file path

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

Returns PipelineBase object

evalml.pipelines.CatBoostRegressionPipeline.predict

CatBoostRegressionPipeline.**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.CatBoostRegressionPipeline.save

CatBoostRegressionPipeline.**save**(*file_path*)

Saves pipeline at file path

Parameters **file_path** (*str*) – location to save file

Returns None

evalml.pipelines.CatBoostRegressionPipeline.score

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

Evaluate model performance on current and additional objectives

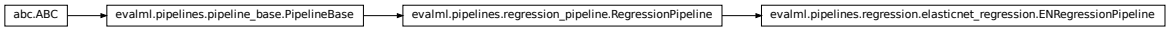
Parameters

- **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.ENRegressionPipeline



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

Elastic Net Pipeline for regression problems.

name = 'ENRegression Pipeline'

custom_name = None

summary = 'Elastic Net Regressor w/ One Hot Encoder + Simple Imputer'

component_graph = ['One Hot Encoder', 'Simple Imputer', 'Elastic Net Regressor']

problem_type = 'regression'

model_family = 'linear_model'

hyperparameters = {'Elastic Net Regressor': {'alpha': Real(low=0, high=1, prior='uni

custom_hyperparameters = None

default_parameters = {'Elastic Net Regressor': {'alpha': 0.5, 'l1_ratio': 0.5, 'max_

Instance attributes

feature_importance	Return importance associated with each feature.

Continued on next page

Table 55 – continued from previous page

parameters	Returns parameter dictionary for this pipeline

Methods:

__init__	Machine learning pipeline made out of transformers and a estimator.
clone	Constructs a new pipeline with the same parameters and components.
describe	Outputs pipeline details including component parameters
fit	Build a model
get_component	Returns component by name
graph	Generate an image representing the pipeline graph
graph_feature_importance	Generate a bar graph of the pipeline's feature importance
load	Loads pipeline at file path
predict	Make predictions using selected features.
save	Saves pipeline at file path
score	Evaluate model performance on current and additional objectives

evalml.pipelines.ENRegressionPipeline.__init__

ENRegressionPipeline.**__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.ENRegressionPipeline.clone

ENRegressionPipeline.**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.ENRegressionPipeline.describe

ENRegressionPipeline.**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.ENRegressionPipeline.fit

ENRegressionPipeline.**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.ENRegressionPipeline.get_component

ENRegressionPipeline.**get_component**(*name*)
Returns component by name

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

Returns component to return

Return type Component

evalml.pipelines.ENRegressionPipeline.graph

ENRegressionPipeline.**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.ENRegressionPipeline.graph_feature_importance

ENRegressionPipeline.**graph_feature_importance**(*show_all_features=False*)
Generate a bar graph of the pipeline's feature importance

Parameters **show_all_features** (*bool, optional*) – If true, graph features with an importance value of zero. Defaults to false.

Returns plotly.Figure, a bar graph showing features and their corresponding importance

evalml.pipelines.ENRegressionPipeline.load

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

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

Returns PipelineBase object

evalml.pipelines.ENRegressionPipeline.predict

ENRegressionPipeline.**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.ENRegressionPipeline.save

ENRegressionPipeline.**save**(*file_path*)
Saves pipeline at file path

Parameters **file_path** (*str*) – location to save file

Returns None

evalml.pipelines.ENRegressionPipeline.score

ENRegressionPipeline.**score**(*X*, *y*, *objectives*)
Evaluate model performance on current and additional objectives

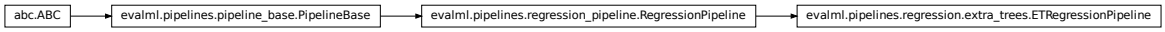
Parameters

- **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.ETRegressionPipeline



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

Extra Trees Pipeline for regression problems.

name = 'Extra Trees Regression Pipeline'

custom_name = 'Extra Trees Regression Pipeline'

summary = 'Extra Trees Regressor w/ One Hot Encoder + Simple Imputer'

component_graph = ['One Hot Encoder', 'Simple Imputer', 'Extra Trees Regressor']

problem_type = 'regression'

model_family = 'extra_trees'

hyperparameters = {'Extra Trees Regressor': {'max_depth': Integer(low=4, high=10, pr

custom_hyperparameters = None

default_parameters = {'Extra Trees Regressor': {'max_depth': 6, 'max_features': 'au

Instance attributes

feature_importance	Return importance associated with each feature.
parameters	Returns parameter dictionary for this pipeline

Methods:

__init__	Machine learning pipeline made out of transformers and a estimator.
clone	Constructs a new pipeline with the same parameters and components.
describe	Outputs pipeline details including component parameters
fit	Build a model
get_component	Returns component by name
graph	Generate an image representing the pipeline graph
graph_feature_importance	Generate a bar graph of the pipeline's feature importance
load	Loads pipeline at file path
predict	Make predictions using selected features.
save	Saves pipeline at file path
score	Evaluate model performance on current and additional objectives

evalml.pipelines.ETRegressionPipeline.__init__

ETRegressionPipeline.**__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.ETRegressionPipeline.clone

ETRegressionPipeline.**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.ETRegressionPipeline.describe

ETRegressionPipeline.**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.ETRegressionPipeline.fit

ETRegressionPipeline.**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.ETRegressionPipeline.get_component

ETRegressionPipeline.**get_component**(*name*)
Returns component by name

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

Returns component to return

Return type Component

evalml.pipelines.ETRegressionPipeline.graph

ETRegressionPipeline.**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.ETRegressionPipeline.graph_feature_importance

ETRegressionPipeline.**graph_feature_importance**(*show_all_features=False*)
Generate a bar graph of the pipeline's feature importance

Parameters **show_all_features** (*bool, optional*) – If true, graph features with an importance value of zero. Defaults to false.

Returns plotly.Figure, a bar graph showing features and their corresponding importance

evalml.pipelines.ETRegressionPipeline.load

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

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

Returns PipelineBase object

evalml.pipelines.ETRegressionPipeline.predict

ETRegressionPipeline.**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.ETRegressionPipeline.save

ETRegressionPipeline.**save**(*file_path*)

Saves pipeline at file path

Parameters **file_path** (*str*) – location to save file

Returns None

evalml.pipelines.ETRegressionPipeline.score

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

Evaluate model performance on current and additional objectives

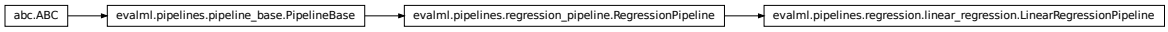
Parameters

- **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.LinearRegressionPipeline



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

Linear Regression Pipeline for regression problems.

name = 'Linear Regression Pipeline'

custom_name = None

summary = 'Linear Regressor w/ One Hot Encoder + Simple Imputer + Standard Scaler'

component_graph = ['One Hot Encoder', 'Simple Imputer', 'Standard Scaler', 'Linear Reg

problem_type = 'regression'

model_family = 'linear_model'

hyperparameters = {'Linear Regressor': {'fit_intercept': [True, False], 'normalize':

custom_hyperparameters = None

default_parameters = {'Linear Regressor': {'fit_intercept': True, 'n_jobs': -1, 'no

Instance attributes

feature_importance	Return importance associated with each feature.

Continued on next page

Table 59 – continued from previous page

parameters	Returns parameter dictionary for this pipeline

Methods:

__init__	Machine learning pipeline made out of transformers and a estimator.
clone	Constructs a new pipeline with the same parameters and components.
describe	Outputs pipeline details including component parameters
fit	Build a model
get_component	Returns component by name
graph	Generate an image representing the pipeline graph
graph_feature_importance	Generate a bar graph of the pipeline's feature importance
load	Loads pipeline at file path
predict	Make predictions using selected features.
save	Saves pipeline at file path
score	Evaluate model performance on current and additional objectives

evalml.pipelines.LinearRegressionPipeline.__init__

LinearRegressionPipeline.**__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.LinearRegressionPipeline.clone

LinearRegressionPipeline.**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.LinearRegressionPipeline.describe

LinearRegressionPipeline.**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.LinearRegressionPipeline.fit

LinearRegressionPipeline.**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.LinearRegressionPipeline.get_component

LinearRegressionPipeline.**get_component**(*name*)

Returns component by name

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

Returns component to return

Return type Component

evalml.pipelines.LinearRegressionPipeline.graph

LinearRegressionPipeline.**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.LinearRegressionPipeline.graph_feature_importance

LinearRegressionPipeline.**graph_feature_importance**(*show_all_features=False*)

Generate a bar graph of the pipeline's feature importance

Parameters **show_all_features** (*bool, optional*) – If true, graph features with an importance value of zero. Defaults to false.

Returns plotly.Figure, a bar graph showing features and their corresponding importance

evalml.pipelines.LinearRegressionPipeline.load

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

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

Returns PipelineBase object

evalml.pipelines.LinearRegressionPipeline.predict

LinearRegressionPipeline.**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.LinearRegressionPipeline.save

LinearRegressionPipeline.**save**(*file_path*)
Saves pipeline at file path

Parameters **file_path** (*str*) – location to save file

Returns None

evalml.pipelines.LinearRegressionPipeline.score

LinearRegressionPipeline.**score**(*X*, *y*, *objectives*)
Evaluate model performance on current and additional objectives

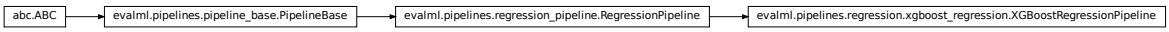
Parameters

- **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.XGBoostRegressionPipeline



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

XGBoost Pipeline for regression problems.

name = 'XGBoost Regression Pipeline'

custom_name = None

summary = 'XGBoost Regressor w/ One Hot Encoder + Simple Imputer'

component_graph = ['One Hot Encoder', 'Simple Imputer', 'XGBoost Regressor']

problem_type = 'regression'

model_family = 'xgboost'

hyperparameters = {'One Hot Encoder': {}, 'Simple Imputer': {'impute_strategy': ['m

custom_hyperparameters = None

default_parameters = {'One Hot Encoder': {'categories': None, 'drop': None, 'handle_

Instance attributes

feature_importance	Return importance associated with each feature.
parameters	Returns parameter dictionary for this pipeline

Methods:

__init__	Machine learning pipeline made out of transformers and a estimator.
clone	Constructs a new pipeline with the same parameters and components.
describe	Outputs pipeline details including component parameters
fit	Build a model
get_component	Returns component by name
graph	Generate an image representing the pipeline graph
graph_feature_importance	Generate a bar graph of the pipeline's feature importance
load	Loads pipeline at file path
predict	Make predictions using selected features.
save	Saves pipeline at file path
score	Evaluate model performance on current and additional objectives

evalml.pipelines.XGBoostRegressionPipeline.__init__

XGBoostRegressionPipeline.**__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.XGBoostRegressionPipeline.clone

XGBoostRegressionPipeline.**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.XGBoostRegressionPipeline.describe

XGBoostRegressionPipeline.**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.XGBoostRegressionPipeline.fit

XGBoostRegressionPipeline.**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.XGBoostRegressionPipeline.get_component

XGBoostRegressionPipeline.**get_component**(*name*)
Returns component by name

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

Returns component to return

Return type Component

evalml.pipelines.XGBoostRegressionPipeline.graph

XGBoostRegressionPipeline.**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.XGBoostRegressionPipeline.graph_feature_importance

XGBoostRegressionPipeline.**graph_feature_importance**(*show_all_features=False*)
Generate a bar graph of the pipeline's feature importance

Parameters **show_all_features** (*bool, optional*) – If true, graph features with an importance value of zero. Defaults to false.

Returns plotly.Figure, a bar graph showing features and their corresponding importance

evalml.pipelines.XGBoostRegressionPipeline.load

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

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

Returns PipelineBase object

evalml.pipelines.XGBoostRegressionPipeline.predict

XGBoostRegressionPipeline.**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.XGBoostRegressionPipeline.save

XGBoostRegressionPipeline.**save**(*file_path*)

Saves pipeline at file path

Parameters **file_path** (*str*) – location to save file

Returns None

evalml.pipelines.XGBoostRegressionPipeline.score

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

Evaluate model performance on current and additional objectives

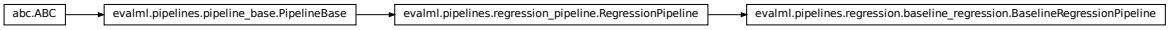
Parameters

- **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.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

feature_importance	Return importance associated with each feature.

Continued on next page

Table 63 – continued from previous page

parameters	Returns parameter dictionary for this pipeline

Methods:

__init__	Machine learning pipeline made out of transformers and a estimator.
clone	Constructs a new pipeline with the same parameters and components.
describe	Outputs pipeline details including component parameters
fit	Build a model
get_component	Returns component by name
graph	Generate an image representing the pipeline graph
graph_feature_importance	Generate a bar graph of the pipeline's feature importance
load	Loads pipeline at file path
predict	Make predictions using selected features.
save	Saves pipeline at file path
score	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 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**(*show_all_features=False*)
Generate a bar graph of the pipeline's feature importance

Parameters **show_all_features** (*bool, optional*) – If true, graph features with an importance value of zero. Defaults to false.

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*)
Saves pipeline at file path

Parameters **file_path** (*str*) – location to save file

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

feature_importance	Return importance associated with each feature.
parameters	Returns parameter dictionary for this pipeline

Methods:

__init__	Machine learning pipeline made out of transformers and a estimator.
clone	Constructs a new pipeline with the same parameters and components.
describe	Outputs pipeline details including component parameters
fit	Build a model
get_component	Returns component by name
graph	Generate an image representing the pipeline graph
graph_feature_importance	Generate a bar graph of the pipeline's feature importance
load	Loads pipeline at file path
predict	Make predictions using selected features.
save	Saves pipeline at file path
score	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 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**(*show_all_features=False*)
Generate a bar graph of the pipeline's feature importance

Parameters **show_all_features** (*bool, optional*) – If true, graph features with an importance value of zero. Defaults to false.

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*)

Saves pipeline at file path

Parameters **file_path** (*str*) – location to save file

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

Pipeline Utils

all_pipelines	Returns a complete list of all supported pipeline classes.
get_pipelines	Returns the pipelines allowed for a particular problem type.
get_estimators	Returns the estimators allowed for a particular problem type.
make_pipeline	Given input data, target data, an estimator class and the problem type,
list_model_families	List model type for a particular problem type.

evalml.pipelines.all_pipelines

evalml.pipelines.**all_pipelines**()

Returns a complete list of all supported pipeline classes.

Returns a list of pipeline classes

Return type list[*PipelineBase*]

evalml.pipelines.get_pipelines

evalml.pipelines.**get_pipelines**(*problem_type*, *model_families=None*)

Returns the pipelines 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 pipeline classes

Return type list[*PipelineBase*]

evalml.pipelines.get_estimators

evalml.pipelines.**get_estimators**(*problem_type*, *model_families=None*)

Returns the estimators allowed for a particular problem type.

Can also optionally filter by a list of model types.

Parameters

- **problem_type** (*ProblemTypes or str*) – problem type to filter for
- **model_families** (*list[ModelFamily] or list[str]*) – model families to filter for

Returns a list of estimator subclasses

Return type list[class]

evalml.pipelines.make_pipeline

evalml.pipelines.**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

evalml.pipelines.list_model_families

evalml.pipelines.**list_model_families**(*problem_type*)

List model type 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*]

Pipeline Graph Utils

precision_recall_curve	Given labels and binary classifier predicted probabilities, compute and return the data representing a precision-recall curve.
graph_precision_recall_curve	Generate and display a precision-recall plot.
roc_curve	Given labels and classifier predicted probabilities, compute and return the data representing a Receiver Operating Characteristic (ROC) curve.
graph_roc_curve	Generate and display a Receiver Operating Characteristic (ROC) plot.
confusion_matrix	Confusion matrix for binary and multiclass classification.
normalize_confusion_matrix	Normalizes a confusion matrix.
graph_confusion_matrix	Generate and display a confusion matrix plot.
calculate_permutation_importance	Calculates permutation importance for features.
graph_permutation_importance	Generate a bar graph of the pipeline's permutation importance.

evalml.pipelines.precision_recall_curve

evalml.pipelines.**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.pipelines.graph_precision_recall_curve

evalml.pipelines.**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.pipelines.roc_curve

evalml.pipelines.**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.pipelines.graph_roc_curve

evalml.pipelines.**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 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.pipelines.confusion_matrix

evalml.pipelines.**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 np.array

evalml.pipelines.normalize_confusion_matrix

evalml.pipelines.**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 A normalized version of the input confusion matrix.

evalml.pipelines.graph_confusion_matrix

evalml.pipelines.**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.pipelines.calculate_permutation_importance

evalml.pipelines.**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.pipelines.graph_permutation_importance

evalml.pipelines.**graph_permutation_importance**(*pipeline*, *X*, *y*, *objective*, *show_all_features=False*)

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
- **show_all_features** (*bool, optional*) – If True, graph features with a permutation importance value of zero. Defaults to False.

Returns plotly.Figure, a bar graph showing features and their respective permutation importance.

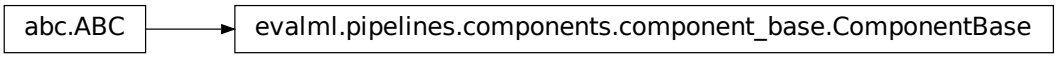
Components

Component Base Classes

Components represent a step in a pipeline.

ComponentBase	Base class for all components.
Transformer	A component that may or may not need fitting that transforms data.
Estimator	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

__init__	Initialize self.
clone	Constructs a new component with the same parameters
describe	Describe a component and its parameters
fit	Fits component to data

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*)

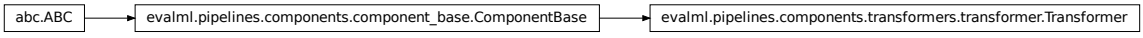
Fits component to data

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.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

__init__	Initialize self.
clone	Constructs a new component with the same parameters
describe	Describe a component and its parameters
fit	Fits component to data
fit_transform	Fits on X and transforms X
transform	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.transform

Transformer.**transform**(*X*, *y=None*)
Transforms data X

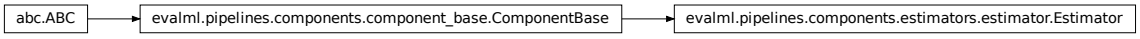
Parameters

- **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

__init__	Initialize self.
clone	Constructs a new component with the same parameters
describe	Describe a component and its parameters
fit	Fits component to data
predict	Make predictions using selected features.
predict_proba	Make probability estimates for labels.

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.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

Transformers

Transformers are components that take in data as input and output transformed data.

DropColumns	Drops specified columns in input data.
SelectColumns	Selects specified columns in input data.
OneHotEncoder	One-hot encoder to encode non-numeric data.
PerColumnImputer	Imputes missing data according to a specified imputation strategy per column
SimpleImputer	Imputes missing data according to a specified imputation strategy.
StandardScaler	Standardize features: removes mean and scales to unit variance.
RFRegressorSelectFromModel	Selects top features based on importance weights using a Random Forest regressor.
RFClassifierSelectFromModel	Selects top features based on importance weights using a Random Forest classifier.

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

parameters	Returns the parameters which were used to initialize the component

Methods:

__init__	Initalizes an transformer that drops specified columns in input data.

Continued on next page

Table 75 – continued from previous page

clone	Constructs a new component with the same parameters
describe	Describe a component and its parameters
fit	'Fits' the transformer by checking if the column names are present in the dataset.
fit_transform	Fit transformer to data, then transform data.
transform	Transforms data X by dropping columns.

evalml.pipelines.components.DropColumns.__init__

DropColumns.**__init__**(*columns=None*, *random_state=0*, ***kwargs*)

Initalizes 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.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

parameters	Returns the parameters which were used to initialize the component

Methods:

__init__	Initalizes an transformer that drops specified columns in input data.
clone	Constructs a new component with the same parameters
describe	Describe a component and its parameters
fit	'Fits' the transformer by checking if the column names are present in the dataset.
fit_transform	Fit transformer to data, then transform data.
transform	Transforms data X by selecting columns.

evalml.pipelines.components.SelectColumns.__init__

SelectColumns.**__init__**(*columns=None*, *random_state=0*, ***kwargs*)

Initalizes 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.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_unknown='ignore'*, *handle_missing='error'*, *random_state=0*, ***kwargs*)

One-hot encoder to encode non-numeric data.

name = 'One Hot Encoder'

model_family = 'none'

hyperparameter_ranges = {}

default_parameters = {'categories': None, 'drop': None, 'handle_missing': 'error',

Instance attributes

parameters	Returns the parameters which were used to initialize the component

Methods:

__init__	Initalizes an transformer that encodes categorical features in a one-hot numeric array."
clone	Constructs a new component with the same parameters
describe	Describe a component and its parameters
fit	Fits component to data
fit_transform	Fits on X and transforms X
get_feature_names	Returns names of transformed and added columns
transform	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*)

Initalizes 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.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*, *default_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

parameters	Returns the parameters which were used to initialize the component

Methods:

__init__	Initializes a transformer that imputes missing data according to the specified imputation strategy per column."
clone	Constructs a new component with the same parameters
describe	Describe a component and its parameters
fit	Fits imputers on data X
fit_transform	Fits imputer on data X then imputes missing values in X
transform	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.transform

PerColumnImputer.**transform**(*X*, *y=None*)

Transforms data X by imputing missing values

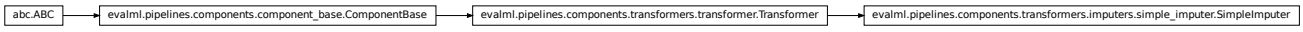
Parameters

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

Returns Transformed X

Return type pd.DataFrame

evalml.pipelines.components.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

parameters	Returns the parameters which were used to initialize the component

Methods:

__init__	Initalizes an transformer that imputes missing data according to the specified imputation strategy."
clone	Constructs a new component with the same parameters

Continued on next page

Table 83 – continued from previous page

describe	Describe a component and its parameters
fit	Fits imputer to data
fit_transform	Fits on X and transforms X
transform	Transforms data X by imputing missing values

evalml.pipelines.components.SimpleImputer.__init__

SimpleImputer.**__init__**(*impute_strategy='most_frequent'*, *fill_value=None*, *random_state=0*, ***kwargs*)

Initalizes 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

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.SimpleImputer.transform

SimpleImputer.**transform**(*X*, *y=None*)

Transforms data X by imputing missing values

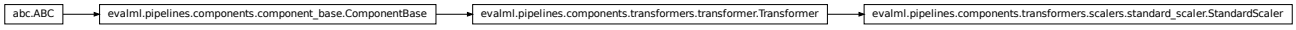
Parameters

- **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

parameters	Returns the parameters which were used to initialize the component

Methods:

__init__	Initialize self.
clone	Constructs a new component with the same parameters
describe	Describe a component and its parameters
fit	Fits component to data
fit_transform	Fits on X and transforms X
transform	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.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*, *percent_features=0.5*, *threshold=-inf*, *n_jobs=-1*, *random_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

`parameters`	Returns the parameters which were used to initialize the component

Methods:

`__init__`	Initialize self.
`clone`	Constructs a new component with the same parameters
`describe`	Describe a component and its parameters
`fit`	Fits component to data
`fit_transform`	Fits feature selector on data X then transforms X by selecting features
`get_indices`	Get integer index of features selected
`get_names`	Get names of selected features.
`transform`	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_indices

RFRegressorSelectFromModel.**get_indices**()

Get integer index of features selected

Returns list of indices

Return type list

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.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*, *percent_features=0.5*, *threshold=-inf*, *n_jobs=-1*, *random_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

parameters	Returns the parameters which were used to initialize the component

Methods:

__init__	Initialize self.
clone	Constructs a new component with the same parameters
describe	Describe a component and its parameters
fit	Fits component to data
fit_transform	Fits feature selector on data X then transforms X by selecting features
get_indices	Get integer index of features selected
get_names	Get names of selected features.
transform	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_indices

RFClassifierSelectFromModel.**get_indices**()
Get integer index of features selected

Returns list of indices

Return type list

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.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

Estimators

Classifiers

Classifiers are components that output a predicted class label.

CatBoostClassifier	CatBoost Classifier, a classifier that uses gradient-boosting on decision trees.
ElasticNetClassifier	Elastic Net Classifier.
ExtraTreesClassifier	Extra Trees Classifier.
RandomForestClassifier	Random Forest Classifier.
LogisticRegressionClassifier	Logistic Regression Classifier.
XGBoostClassifier	XGBoost Classifier.
BaselineClassifier	Classifier that predicts using the specified strategy.

evalml.pipelines.components.CatBoostClassifier

class evalml.pipelines.components.**CatBoostClassifier**(*n_estimators=1000*, *eta=0.03*, *max_depth=6*, *bootstrap_type=None*, *random_state=0*, ***kwargs*)

CatBoost Classifier, a classifier that uses gradient-boosting on decision trees. CatBoost is an open-source library and natively supports categorical features.

For more information, check out https://catboost.ai/

name = 'CatBoost Classifier'

model_family = 'catboost'

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

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

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

Instance attributes

SEED_MAX	
SEED_MIN	
feature_importance	Returns importance associated with each feature.
parameters	Returns the parameters which were used to initialize the component

Methods:

__init__	Initialize self.
clone	Constructs a new component with the same parameters
describe	Describe a component and its parameters
fit	Fits component to data
predict	Make predictions using selected features.
predict_proba	Make probability estimates for labels.

evalml.pipelines.components.CatBoostClassifier.__init__

CatBoostClassifier.**__init__**(*n_estimators=1000*, *eta=0.03*, *max_depth=6*, *bootstrap_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.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*)

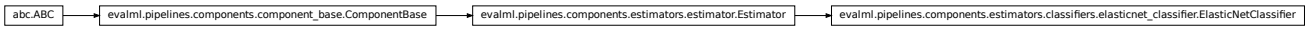
Make probability estimates for labels.

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

Returns probability estimates

Return type pd.DataFrame

evalml.pipelines.components.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='ide

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

Instance attributes

feature_importance	Returns importance associated with each feature.
parameters	Returns the parameters which were used to initialize the component

Methods:

__init__	Initialize self.
clone	Constructs a new component with the same parameters
describe	Describe a component and its parameters
fit	Fits component to data
predict	Make predictions using selected features.
predict_proba	Make probability estimates for labels.

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.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.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

feature_importance	Returns importance associated with each feature.
parameters	Returns the parameters which were used to initialize the component

Methods:

__init__	Initialize self.
clone	Constructs a new component with the same parameters
describe	Describe a component and its parameters
fit	Fits component to data
predict	Make predictions using selected features.
predict_proba	Make probability estimates for labels.

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.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.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

feature_importance	Returns importance associated with each feature.
parameters	Returns the parameters which were used to initialize the component

Methods:

__init__	Initialize self.
clone	Constructs a new component with the same parameters
describe	Describe a component and its parameters
fit	Fits component to data
predict	Make predictions using selected features.
predict_proba	Make probability estimates for labels.

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.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.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

feature_importance	Returns importance associated with each feature.
parameters	Returns the parameters which were used to initialize the component

Methods:

__init__	Initialize self.
clone	Constructs a new component with the same parameters
describe	Describe a component and its parameters
fit	Fits component to data
predict	Make predictions using selected features.
predict_proba	Make probability estimates for labels.

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.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.XGBoostClassifier

class evalml.pipelines.components.**XGBoostClassifier**(*eta=0.1*, *max_depth=6*, *min_child_weight=1*, *n_estimators=100*, *random_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	Returns importance associated with each feature.
parameters	Returns the parameters which were used to initialize the component

Methods:

__init__	Initialize self.
clone	Constructs a new component with the same parameters
describe	Describe a component and its parameters
fit	Fits component to data
predict	Make predictions using selected features.
predict_proba	Make probability estimates for labels.

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.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.BaselineClassifier

class evalml.pipelines.components.**BaselineClassifier**(*strategy='mode'*, *random_state=0*, ***kwargs*)

Classifier that predicts using the specified strategy.

This is useful as a simple baseline classifier to compare with other classifiers.

name = 'Baseline Classifier'

model_family = 'baseline'

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

hyperparameter_ranges = {}

default_parameters = {'strategy': 'mode'}

Instance attributes

feature_importance	Returns importance associated with each feature.
parameters	Returns the parameters which were used to initialize the component

Methods:

__init__	Baseline classifier that uses a simple strategy to make predictions.
clone	Constructs a new component with the same parameters
describe	Describe a component and its parameters
fit	Fits component to data
predict	Make predictions using selected features.
predict_proba	Make probability estimates for labels.

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.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

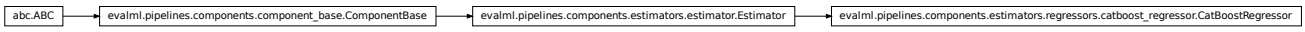
Return type pd.DataFrame

Regressors

Regressors are components that output a predicted target value.

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

evalml.pipelines.components.CatBoostRegressor



class evalml.pipelines.components.**CatBoostRegressor**(*n_estimators=1000*, *eta=0.03*, *max_depth=6*, *bootstrap_type=None*, *random_state=0*, ***kwargs*)

CatBoost Regressor, a regressor that uses gradient-boosting on decision trees. CatBoost is an open-source library and natively supports categorical features.

For more information, check out https://catboost.ai/

name = 'CatBoost Regressor'

model_family = 'catboost'

supported_problem_types = [<ProblemTypes.REGRESSION: 'regression'>]

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_estima

Instance attributes

SEED_MAX

Continued on next page

Table 106 – continued from previous page

SEED_MIN	
feature_importance	Returns importance associated with each feature.
parameters	Returns the parameters which were used to initialize the component

Methods:

__init__	Initialize self.
clone	Constructs a new component with the same parameters
describe	Describe a component and its parameters
fit	Build a model
predict	Make predictions using selected features.
predict_proba	Make probability estimates for labels.

evalml.pipelines.components.CatBoostRegressor.__init__

CatBoostRegressor.**__init__**(*n_estimators=1000*, *eta=0.03*, *max_depth=6*, *bootstrap_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.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.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='ide

default_parameters = {'alpha': 0.5, 'l1_ratio': 0.5, 'max_iter': 1000, 'normalize':

Instance attributes

feature_importance	Returns importance associated with each feature.
parameters	Returns the parameters which were used to initialize the component

Methods:

__init__	Initialize self.
clone	Constructs a new component with the same parameters
describe	Describe a component and its parameters
fit	Fits component to data
predict	Make predictions using selected features.
predict_proba	Make probability estimates for labels.

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.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.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

feature_importance	Returns importance associated with each feature.

Continued on next page

Table 110 – continued from previous page

parameters	Returns the parameters which were used to initialize the component

Methods:

__init__	Initialize self.
clone	Constructs a new component with the same parameters
describe	Describe a component and its parameters
fit	Fits component to data
predict	Make predictions using selected features.
predict_proba	Make probability estimates for labels.

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.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*)

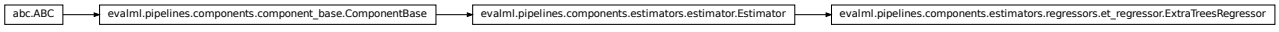
Make probability estimates for labels.

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

Returns probability estimates

Return type pd.DataFrame

evalml.pipelines.components.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', transf

default_parameters = {'max_depth': 6, 'max_features': 'auto', 'min_samples_split':

Instance attributes

`feature_importance`	Returns importance associated with each feature.
`parameters`	Returns the parameters which were used to initialize the component

Methods:

`__init__`	Initialize self.
`clone`	Constructs a new component with the same parameters
`describe`	Describe a component and its parameters
`fit`	Fits component to data
`predict`	Make predictions using selected features.
`predict_proba`	Make probability estimates for labels.

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.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.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', transf

default_parameters = {'max_depth': 6, 'n_estimators': 100, 'n_jobs': -1}

Instance attributes

feature_importance	Returns importance associated with each feature.
parameters	Returns the parameters which were used to initialize the component

Methods:

__init__	Initialize self.
clone	Constructs a new component with the same parameters
describe	Describe a component and its parameters
fit	Fits component to data
predict	Make predictions using selected features.
predict_proba	Make probability estimates for labels.

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.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.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	Returns importance associated with each feature.
parameters	Returns the parameters which were used to initialize the component

Methods:

__init__	Initialize self.
clone	Constructs a new component with the same parameters
describe	Describe a component and its parameters
fit	Fits component to data
predict	Make predictions using selected features.
predict_proba	Make probability estimates for labels.

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.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*)

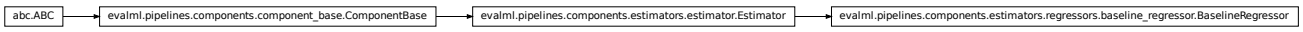
Make probability estimates for labels.

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

Returns probability estimates

Return type pd.DataFrame

evalml.pipelines.components.BaselineRegressor



class evalml.pipelines.components.**BaselineRegressor**(*strategy='mean'*, *random_state=0*, ***kwargs*)

Regressor that predicts using the specified strategy.

This is useful as a simple baseline regressor to compare with other regressors.

name = 'Baseline Regressor'

model_family = 'baseline'

supported_problem_types = [<ProblemTypes.REGRESSION: 'regression'>]

hyperparameter_ranges = {}

default_parameters = {'strategy': 'mean'}

Instance attributes

feature_importance	Returns importance associated with each feature.
parameters	Returns the parameters which were used to initialize the component

Methods:

__init__	Baseline regressor that uses a simple strategy to make predictions.
clone	Constructs a new component with the same parameters
describe	Describe a component and its parameters
fit	Fits component to data
predict	Make predictions using selected features.
predict_proba	Make probability estimates for labels.

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.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

Objective Functions

Objective Base Classes

ObjectiveBase	Base class for all objectives.
BinaryClassificationObjective	Base class for all binary classification objectives.
MulticlassClassificationObjective	Base class for all multiclass classification objectives.
RegressionObjective	Base class for all regression objectives.

evalml.objectives.ObjectiveBase

class evalml.objectives.**ObjectiveBase**
Base class for all objectives.

Methods

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

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*)

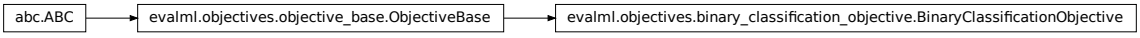
Validates the input based on a few simple checks.

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

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

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

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

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*)

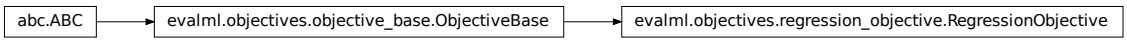
Validates the input based on a few simple checks.

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

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

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

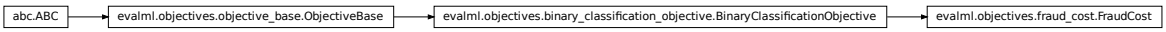
- **y_predicted** (*pd.Series*) – predicted values of length [n_samples]
- **y_true** (*pd.Series*) – actual class labels of length [n_samples]

Returns None

Domain-Specific Objectives

FraudCost	Score the percentage of money lost of the total transaction amount process due to fraud.
LeadScoring	Lead scoring.

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

__init__	Create instance of FraudCost
decision_function	Determine if a transaction is fraud given predicted probabilities, threshold, and dataframe with transaction amount.
objective_function	Calculate amount lost to fraud per transaction given predictions, true values, and dataframe with transaction amount.
optimize_threshold	Learn a binary classification threshold which optimizes the current objective.
score	Returns a numerical score indicating performance based on the differences between the predicted and actual values.
validate_inputs	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.decision_function

FraudCost.**decision_function**(*ypred_proba*, *threshold=0.0*, *X=None*)

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

Parameters

- **ypred_proba** (*pd.Series*) – Predicted probablities
- **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**(*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.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

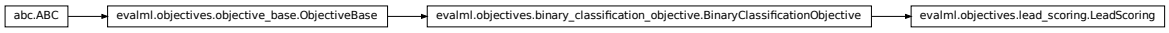
FraudCost.**validate_inputs**(*y_true*, *y_predicted*)
Validates the input based on a few simple checks.

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

__init__	Create instance.
decision_function	Apply a learned threshold to predicted probabilities to get predicted classes.
objective_function	Calculate the profit per lead.
optimize_threshold	Learn a binary classification threshold which optimizes the current objective.
score	Returns a numerical score indicating performance based on the differences between the predicted and actual values.
validate_inputs	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.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

Classification Objectives

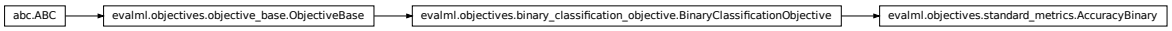
AccuracyBinary	Accuracy score for binary classification.
AccuracyMulticlass	Accuracy score for multiclass classification.
AUC	AUC score for binary classification.
AUCMacro	AUC score for multiclass classification using macro averaging.
AUCMicro	AUC score for multiclass classification using micro averaging.
AUCWeighted	AUC Score for multiclass classification using weighted averaging.
BalancedAccuracyBinary	Balanced accuracy score for binary classification.
BalancedAccuracyMulticlass	Balanced accuracy score for multiclass classification.
F1	F1 score for binary classification.
F1Micro	F1 score for multiclass classification using micro averaging.
F1Macro	F1 score for multiclass classification using macro averaging.
F1Weighted	F1 score for multiclass classification using weighted averaging.
LogLossBinary	Log Loss for binary classification.
LogLossMulticlass	Log Loss for multiclass classification.
MCCBinary	Matthews correlation coefficient for binary classification.
MCCMulticlass	Matthews correlation coefficient for multiclass classification.
Precision	Precision score for binary classification.
PrecisionMicro	Precision score for multiclass classification using micro averaging.
PrecisionMacro	Precision score for multiclass classification using macro averaging.
PrecisionWeighted	Precision score for multiclass classification using weighted averaging.
Recall	Recall score for binary classification.
RecallMicro	Recall score for multiclass classification using micro averaging.
RecallMacro	Recall score for multiclass classification using macro averaging.

Continued on next page

Table 128 – continued from previous page

RecallWeighted	Recall score for multiclass classification using weighted averaging.

evalml.objectives.AccuracyBinary



class evalml.objectives.**AccuracyBinary**

Accuracy score for binary classification.

Methods

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

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*)

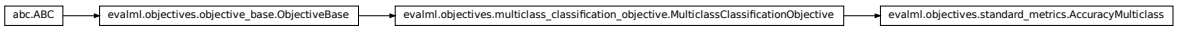
Validates the input based on a few simple checks.

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

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

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

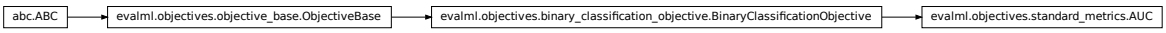
AccuracyMulticlass.**validate_inputs**(*y_true*, *y_predicted*)
Validates the input based on a few simple checks.

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

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

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*)

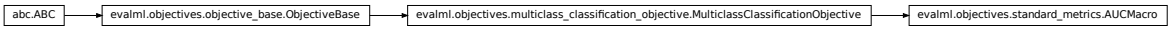
Validates the input based on a few simple checks.

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

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

evalml.objectives.AUCMacro.objective_function

AUCMacro.**objective_function**(*y_true*, *y_predicted*, *X=None*)

Computes the relative value of the provided predictions compared to the actual labels, according a specified metric

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

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

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*)

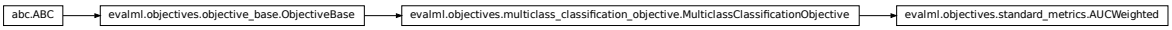
Validates the input based on a few simple checks.

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

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

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*)

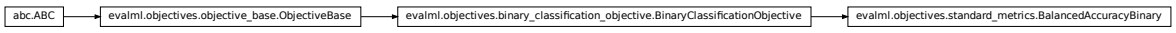
Validates the input based on a few simple checks.

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

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

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**(*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.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*)

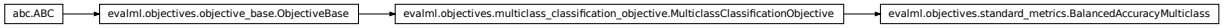
Validates the input based on a few simple checks.

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

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

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*)

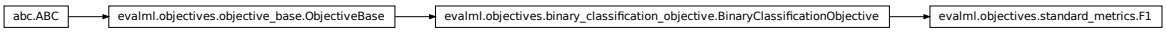
Validates the input based on a few simple checks.

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

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

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

- **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.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*)

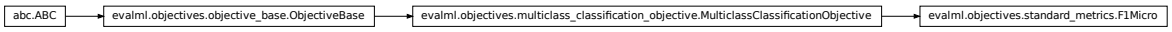
Validates the input based on a few simple checks.

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

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

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*)

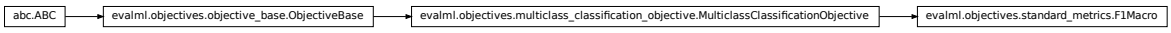
Validates the input based on a few simple checks.

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

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

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

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

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

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

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*)

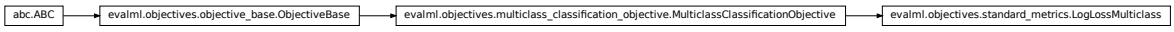
Validates the input based on a few simple checks.

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

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

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*)

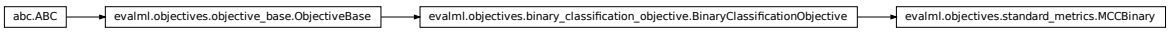
Validates the input based on a few simple checks.

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

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

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

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

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

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

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

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

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

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

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

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

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*)

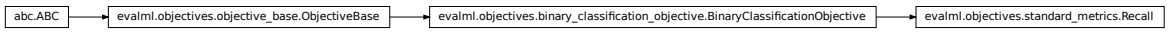
Validates the input based on a few simple checks.

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

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

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

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

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

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

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

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

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

Regression Objectives

R2	Coefficient of determination for regression.
MAE	Mean absolute error for regression.
MSE	Mean squared error for regression.
MeanSquaredLogError	Mean squared log error for regression.
MedianAE	Median absolute error for regression.
MaxError	Maximum residual error for regression.
ExpVariance	Explained variance score for regression.
RootMeanSquaredError	Root mean squared error for regression.
RootMeanSquaredLogError	Root mean squared log error for regression.

evalml.objectives.R2

class evalml.objectives.**R2**

Coefficient of determination for regression.

Methods

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

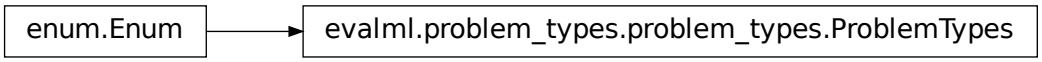
- **y_predicted** (*pd.Series*) – predicted values of length [n_samples]
- **y_true** (*pd.Series*) – actual class labels of length [n_samples]

Returns None

Problem Types

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

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.

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.

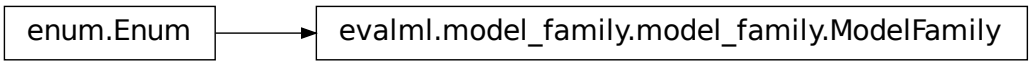
Parameters **problem_type** (*str or ProblemTypes*) – problem type that needs to be handled

Returns ProblemTypes

Model Family

ModelFamily	Enum for family of machine learning models.

evalml.model_family.ModelFamily



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

Tuners

Tuner	Defines API for Tuners.
SKOptTuner	Bayesian Optimizer.
GridSearchTuner	Grid Search Optimizer.
RandomSearchTuner	Random Search Optimizer.

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

__init__	Base Tuner class
add	Register a set of hyperparameters with the score obtained from training a pipeline with those hyperparameters.
is_search_space_exhausted	Optional.
propose	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

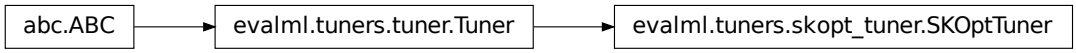
Tuner.**propose**()

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

evalml.tuners.SKOptTuner



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

Bayesian Optimizer.

Methods

__init__	Init SkOptTuner

Continued on next page

Table 168 – continued from previous page

add	Add score to sample
is_search_space_exhausted	Optional.
propose	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

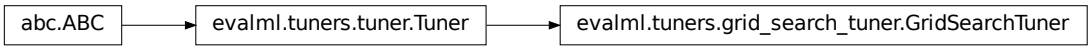
SKOptTuner.**propose**()

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

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

__init__	Generate all of the possible points to search for in the grid
add	Not applicable to grid search tuner as generated parameters are not dependent on scores of previous parameters.
is_search_space_exhausted	Checks if it is possible to generate a set of valid parameters.
propose	Returns parameters from _grid_points 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**()

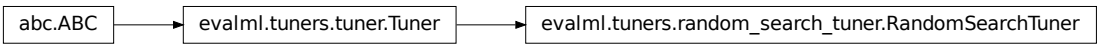
Returns parameters from _grid_points iterations

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

Returns proposed pipeline parameters

Return type dict

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

__init__	Sets up check for duplication if needed.
add	Not applicable to random search tuner as generated parameters are not dependent on scores of previous parameters.
is_search_space_exhausted	Checks if it is possible to generate a set of valid parameters.
propose	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 initalized 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

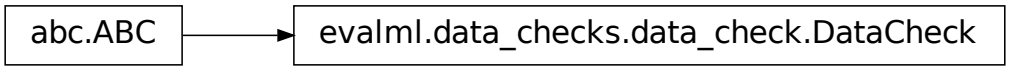
Return type dict

Data Checks

Data Check Classes

DataCheck	Base class for all data checks.
InvalidTargetDataCheck	Checks if the target labels contain missing or invalid data.
HighlyNullDataCheck	Checks if there are any highly-null columns in the input.
IDColumnsDataCheck	Check if any of the features are likely to be ID columns.
LabelLeakageDataCheck	Check if any of the features are highly correlated with the target.
OutliersDataCheck	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.

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:

validate	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.

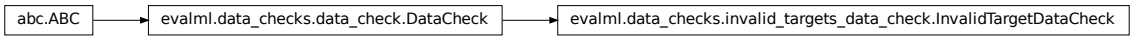
Parameters

- **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:

validate	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.

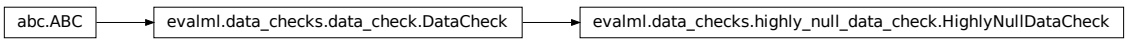
Returns list with DataCheckErrors if any invalid data is found in target labels.

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:

__init__	Checks if there are any highly-null columns in the input.
validate	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.

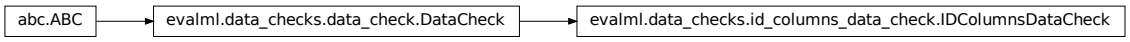
Returns list with a DataCheckWarning if there are any highly-null columns.

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:

__init__	Check if any of the features are likely to be ID columns.
validate	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

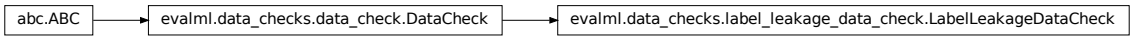
- **X** (*pd.DataFrame*) – The input features to check
- **threshold** (*float*) – the probability threshold to be considered an ID column. Defaults to 1.0

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:

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

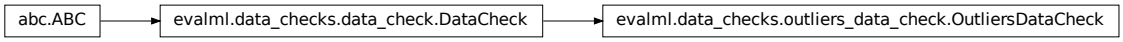
Returns list with a DataCheckWarning if there is label leakage detected.

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:

__init__	Checks if there are any outliers in the input data.
validate	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")]
```

DataChecks	A collection of data checks.
DefaultDataChecks	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

__init__	A collection of data checks.
validate	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.

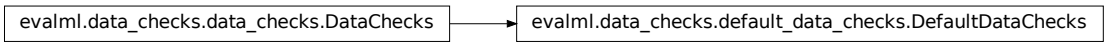
Parameters

- **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, and LabelLeakageDataCheck.

Methods

__init__	A collection of basic data checks.
validate	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*)

Data Check Messages

DataCheckMessage	Base class for all DataCheckMessages.
DataCheckError	DataCheckMessage subclass for errors returned by data checks.
DataCheckWarning	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:

__init__	Message returned by a DataCheck, tagged by name."
__str__	String representation of data check message, equivalent to self.message attribute.
__eq__	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

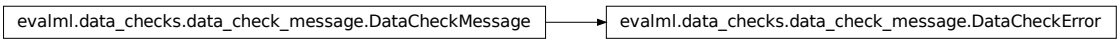
evalml.data_checks.DataCheckMessage.__str__

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:

__init__	Message returned by a DataCheck, tagged by name.”
__str__	String representation of data check message, equivalent to self.message attribute.
__eq__	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__

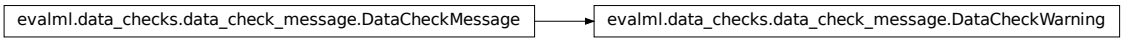
DataCheckError.**__str__**()
String representation of data check message, equivalent to self.message attribute.

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:

__init__	Message returned by a DataCheck, tagged by name."
__str__	String representation of data check message, equivalent to self.message attribute.
__eq__	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__

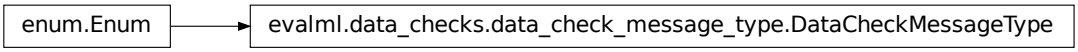
DataCheckWarning.**__eq__**(*other*)

Checks for equality. Two DataCheckMessage objs are considered equivalent if their message type and message are equivalent.

Data Check Message Types

DataCheckMessageType	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.

Utils

import_or_raise	Attempts to import the requested library by name.
convert_to_seconds	Converts a string describing a length of time to its length in seconds.
get_random_state	Generates a numpy.random.RandomState instance using seed.
get_random_seed	Given a numpy.random.RandomState 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

1.6.17 FAQ

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.

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*.

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.

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*.

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).

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.

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.

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.

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.

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.

[ ]:

INDEX

Symbols

__eq__() (*evalml.data_checks.DataCheckError method*), 309
__eq__() (*evalml.data_checks.DataCheckMessage method*), 308
__eq__() (*evalml.data_checks.DataCheckWarning method*), 309
__init__() (*evalml.automl.AutoMLSearch method*), 66
__init__() (*evalml.automl.automl_algorithm.AutoMLAlgorithm method*), 70
__init__() (*evalml.automl.automl_algorithm.IterativeAlgorithm method*), 72
__init__() (*evalml.data_checks.DataCheckError method*), 308
__init__() (*evalml.data_checks.DataCheckMessage method*), 307
__init__() (*evalml.data_checks.DataCheckWarning method*), 309
__init__() (*evalml.data_checks.DataChecks method*), 305
__init__() (*evalml.data_checks.DefaultDataChecks method*), 306
__init__() (*evalml.data_checks.HighlyNullDataCheck method*), 300
__init__() (*evalml.data_checks.IDColumnsDataCheck method*), 302
__init__() (*evalml.data_checks.LabelLeakageDataCheck method*), 303
__init__() (*evalml.data_checks.OutliersDataCheck method*), 304
__init__() (*evalml.objectives.FraudCost method*), 241
__init__() (*evalml.objectives.LeadScoring method*), 243
__init__() (*evalml.pipelines.BaselineBinaryPipeline method*), 134
__init__() (*evalml.pipelines.BaselineMulticlassPipeline method*), 138
__init__() (*evalml.pipelines.BaselineRegressionPipeline method*), 170
__init__() (*evalml.pipelines.BinaryClassificationPipeline method*), 80
__init__() (*evalml.pipelines.CatBoostBinaryClassificationPipeline method*), 90
__init__() (*evalml.pipelines.CatBoostMulticlassClassificationPipeline method*), 94
__init__() (*evalml.pipelines.CatBoostRegressionPipeline method*), 153
__init__() (*evalml.pipelines.ClassificationPipeline method*), 77
__init__() (*evalml.pipelines.ENBinaryPipeline method*), 98
__init__() (*evalml.pipelines.ENMulticlassPipeline method*), 101
__init__() (*evalml.pipelines.ENRegressionPipeline method*), 156
__init__() (*evalml.pipelines.ETBinaryClassificationPipeline method*), 105
__init__() (*evalml.pipelines.ETMulticlassClassificationPipeline method*), 109
__init__() (*evalml.pipelines.ETRegressionPipeline method*), 160
__init__() (*evalml.pipelines.LinearRegressionPipeline method*), 163
__init__() (*evalml.pipelines.LogisticRegressionBinaryPipeline method*), 112
__init__() (*evalml.pipelines.LogisticRegressionMulticlassPipeline method*), 116
__init__() (*evalml.pipelines.MeanBaselineRegressionPipeline method*), 174
__init__() (*evalml.pipelines.ModeBaselineBinaryPipeline method*), 142
__init__() (*evalml.pipelines.ModeBaselineMulticlassPipeline method*), 145
__init__() (*evalml.pipelines.MulticlassClassificationPipeline method*), 83
__init__() (*evalml.pipelines.PipelineBase method*), 74
__init__() (*evalml.pipelines.RFBinaryClassificationPipeline method*), 120
__init__() (*evalml.pipelines.RFMulticlassClassificationPipeline method*), 123
__init__() (*evalml.pipelines.RFRegressionPipeline*

method), 149
__init__() (*evalml.pipelines.RegressionPipeline method*), 87
__init__() (*evalml.pipelines.XGBoostBinaryPipeline method*), 127
__init__() (*evalml.pipelines.XGBoostMulticlassPipeline method*), 131
__init__() (*evalml.pipelines.XGBoostRegressionPipeline method*), 167
__init__() (*evalml.pipelines.components.BaselineClassifier method*), 218
__init__() (*evalml.pipelines.components.BaselineRegressor method*), 233
__init__() (*evalml.pipelines.components.CatBoostClassifier method*), 206
__init__() (*evalml.pipelines.components.CatBoostRegressor method*), 221
__init__() (*evalml.pipelines.components.ComponentBase method*), 182
__init__() (*evalml.pipelines.components.DropColumns method*), 188
__init__() (*evalml.pipelines.components.ElasticNetClassifier method*), 208
__init__() (*evalml.pipelines.components.ElasticNetRegressor method*), 223
__init__() (*evalml.pipelines.components.Estimator method*), 185
__init__() (*evalml.pipelines.components.ExtraTreesClassifier method*), 210
__init__() (*evalml.pipelines.components.ExtraTreesRegressor method*), 227
__init__() (*evalml.pipelines.components.LinearRegressor method*), 225
__init__() (*evalml.pipelines.components.LogisticRegressionClassifier method*), 214
__init__() (*evalml.pipelines.components.OneHotEncoder method*), 192
__init__() (*evalml.pipelines.components.PerColumnImputer method*), 194
__init__() (*evalml.pipelines.components.RFClassifierSelectFromModel method*), 204
__init__() (*evalml.pipelines.components.RFRegressorSelectFromModel method*), 201
__init__() (*evalml.pipelines.components.RandomForestClassifier method*), 212
__init__() (*evalml.pipelines.components.RandomForestRegressor method*), 229
__init__() (*evalml.pipelines.components.SelectColumns method*), 190
__init__() (*evalml.pipelines.components.SimpleImputer method*), 197
__init__() (*evalml.pipelines.components.StandardScaler method*), 199
__init__() (*evalml.pipelines.components.Transformer method*), 184
__init__() (*evalml.pipelines.components.XGBoostClassifier method*), 216
__init__() (*evalml.pipelines.components.XGBoostRegressor method*), 231
__init__() (*evalml.tuners.GridSearchTuner method*), 295
__init__() (*evalml.tuners.RandomSearchTuner method*), 297
__init__() (*evalml.tuners.SKOptTuner method*), 294
__init__() (*evalml.tuners.Tuner method*), 292
__str__() (*evalml.data_checks.DataCheckError method*), 308
__str__() (*evalml.data_checks.DataCheckMessage method*), 308
__str__() (*evalml.data_checks.DataCheckWarning method*), 309

A

AccuracyBinary (*class in evalml.objectives*), 246
AccuracyMulticlass (*class in evalml.objectives*), 248
add() (*evalml.tuners.GridSearchTuner method*), 296
add() (*evalml.tuners.RandomSearchTuner method*), 297
add() (*evalml.tuners.SKOptTuner method*), 294
add() (*evalml.tuners.Tuner method*), 293
add_result() (*evalml.automl.automl_algorithm.AutoMLAlgorithm method*), 71
add_result() (*evalml.automl.automl_algorithm.IterativeAlgorithm method*), 72
add_to_rankings() (*evalml.automl.AutoMLSearch method*), 67
all_pipelines() (*in module evalml.pipelines*), 176
AUC (*class in evalml.objectives*), 249
AUCMacro (*class in evalml.objectives*), 251
AUCMicro (*class in evalml.objectives*), 252
AUCWeighted (*class in evalml.objectives*), 253
AutoMLAlgorithm (*class in evalml.automl.automl_algorithm*), 70
AutoMLSearch (*class in evalml.automl*), 66

B

BalancedAccuracyBinary (*class in evalml.objectives*), 254
BalancedAccuracyMulticlass (*class in evalml.objectives*), 256
BaselineBinaryPipeline (*class in evalml.pipelines*), 133
BaselineClassifier (*class in evalml.pipelines.components*), 218
BaselineMulticlassPipeline (*class in evalml.pipelines*), 137

BaselineRegressionPipeline (*class in evalml.pipelines*), 169
BaselineRegressor (*class in evalml.pipelines.components*), 232
BinaryClassificationObjective (*class in evalml.objectives*), 236
BinaryClassificationPipeline (*class in evalml.pipelines*), 80

C

calculate_permutation_importance() (*in module evalml.pipelines*), 180
CatBoostBinaryClassificationPipeline (*class in evalml.pipelines*), 89
CatBoostClassifier (*class in evalml.pipelines.components*), 206
CatBoostMulticlassClassificationPipeline (*class in evalml.pipelines*), 93
CatBoostRegressionPipeline (*class in evalml.pipelines*), 152
CatBoostRegressor (*class in evalml.pipelines.components*), 220
ClassificationPipeline (*class in evalml.pipelines*), 76
clone() (*evalml.pipelines.BaselineBinaryPipeline method*), 134
clone() (*evalml.pipelines.BaselineMulticlassPipeline method*), 138
clone() (*evalml.pipelines.BaselineRegressionPipeline method*), 170
clone() (*evalml.pipelines.BinaryClassificationPipeline method*), 80
clone() (*evalml.pipelines.CatBoostBinaryClassificationPipeline method*), 91
clone() (*evalml.pipelines.CatBoostMulticlassClassificationPipeline method*), 94
clone() (*evalml.pipelines.CatBoostRegressionPipeline method*), 153
clone() (*evalml.pipelines.ClassificationPipeline method*), 77
clone() (*evalml.pipelines.components.BaselineClassifier method*), 219
clone() (*evalml.pipelines.components.BaselineRegressor method*), 233
clone() (*evalml.pipelines.components.CatBoostClassifier method*), 207
clone() (*evalml.pipelines.components.CatBoostRegressor method*), 221
clone() (*evalml.pipelines.components.ComponentBase method*), 182
clone() (*evalml.pipelines.components.DropColumns method*), 188
clone() (*evalml.pipelines.components.ElasticNetClassifier method*), 209
clone() (*evalml.pipelines.components.ElasticNetRegressor method*), 223
clone() (*evalml.pipelines.components.Estimator method*), 186
clone() (*evalml.pipelines.components.ExtraTreesClassifier method*), 211
clone() (*evalml.pipelines.components.ExtraTreesRegressor method*), 227
clone() (*evalml.pipelines.components.LinearRegressor method*), 225
clone() (*evalml.pipelines.components.LogisticRegressionClassifier method*), 215
clone() (*evalml.pipelines.components.OneHotEncoder method*), 192
clone() (*evalml.pipelines.components.PerColumnImputer method*), 195
clone() (*evalml.pipelines.components.RandomForestClassifier method*), 213
clone() (*evalml.pipelines.components.RandomForestRegressor method*), 229
clone() (*evalml.pipelines.components.RFClassifierSelectFromModel method*), 204
clone() (*evalml.pipelines.components.RFRegressorSelectFromModel method*), 201
clone() (*evalml.pipelines.components.SelectColumns method*), 190
clone() (*evalml.pipelines.components.SimpleImputer method*), 197
clone() (*evalml.pipelines.components.StandardScaler method*), 199
clone() (*evalml.pipelines.components.Transformer method*), 184
clone() (*evalml.pipelines.components.XGBoostClassifier method*), 217
clone() (*evalml.pipelines.components.XGBoostRegressor method*), 231
clone() (*evalml.pipelines.ENBinaryPipeline method*), 98
clone() (*evalml.pipelines.ENMulticlassPipeline method*), 101
clone() (*evalml.pipelines.ENRegressionPipeline method*), 156
clone() (*evalml.pipelines.ETBinaryClassificationPipeline method*), 105
clone() (*evalml.pipelines.ETMulticlassClassificationPipeline method*), 109
clone() (*evalml.pipelines.ETRegressionPipeline method*), 160
clone() (*evalml.pipelines.LinearRegressionPipeline method*), 163
clone() (*evalml.pipelines.LogisticRegressionBinaryPipeline method*), 112
clone() (*evalml.pipelines.LogisticRegressionMulticlassPipeline method*), 116

clone() (*evalml.pipelines.MeanBaselineRegressionPipeline method*), 174
clone() (*evalml.pipelines.ModeBaselineBinaryPipeline method*), 142
clone() (*evalml.pipelines.ModeBaselineMulticlassPipeline method*), 145
clone() (*evalml.pipelines.MulticlassClassificationPipeline method*), 84
clone() (*evalml.pipelines.PipelineBase method*), 74
clone() (*evalml.pipelines.RegressionPipeline method*), 87
clone() (*evalml.pipelines.RFBinaryClassificationPipeline method*), 120
clone() (*evalml.pipelines.RFMulticlassClassificationPipeline method*), 123
clone() (*evalml.pipelines.RFRegressionPipeline method*), 149
clone() (*evalml.pipelines.XGBoostBinaryPipeline method*), 127
clone() (*evalml.pipelines.XGBoostMulticlassPipeline method*), 131
clone() (*evalml.pipelines.XGBoostRegressionPipeline method*), 167
component_graph (*evalml.pipelines.BaselineBinaryPipeline attribute*), 133
component_graph (*evalml.pipelines.BaselineMulticlassPipeline attribute*), 137
component_graph (*evalml.pipelines.BaselineRegressionPipeline attribute*), 169
component_graph (*evalml.pipelines.CatBoostBinaryClassificationPipeline attribute*), 90
component_graph (*evalml.pipelines.CatBoostMulticlassClassificationPipeline attribute*), 93
component_graph (*evalml.pipelines.CatBoostRegressionPipeline attribute*), 152
component_graph (*evalml.pipelines.ENBinaryPipeline attribute*), 97
component_graph (*evalml.pipelines.ENMulticlassPipeline attribute*), 100
component_graph (*evalml.pipelines.ENRegressionPipeline attribute*), 155
component_graph (*evalml.pipelines.ETBinaryClassificationPipeline attribute*), 104
component_graph (*evalml.pipelines.ETMulticlassClassificationPipeline attribute*), 108
component_graph (*evalml.pipelines.ETRegressionPipeline attribute*), 159
component_graph (*evalml.pipelines.LinearRegressionPipeline attribute*), 162
component_graph (*evalml.pipelines.LogisticRegressionBinaryPipeline attribute*), 111
component_graph (*evalml.pipelines.LogisticRegressionMulticlassPipeline attribute*), 115
component_graph (*evalml.pipelines.MeanBaselineRegressionPipeline attribute*), 173
component_graph (*evalml.pipelines.ModeBaselineBinaryPipeline attribute*), 141
component_graph (*evalml.pipelines.ModeBaselineMulticlassPipeline attribute*), 144
component_graph (*evalml.pipelines.RFBinaryClassificationPipeline attribute*), 119
component_graph (*evalml.pipelines.RFMulticlassClassificationPipeline attribute*), 122
component_graph (*evalml.pipelines.RFRegressionPipeline attribute*), 148
component_graph (*evalml.pipelines.XGBoostBinaryPipeline attribute*), 126
component_graph (*evalml.pipelines.XGBoostMulticlassPipeline attribute*), 130
component_graph (*evalml.pipelines.XGBoostRegressionPipeline attribute*), 166
ComponentBase (*class in evalml.pipelines.components*), 182
confusion_matrix() (*in module evalml.pipelines*), 179
convert_to_seconds() (*in module evalml.utils*), 310
custom_hyperparameters (*evalml.pipelines.BaselineBinaryPipeline attribute*), 134
custom_hyperparameters (*evalml.pipelines.BaselineMulticlassPipeline attribute*), 137
custom_hyperparameters (*evalml.pipelines.BaselineRegressionPipeline attribute*), 169
custom_hyperparameters (*evalml.pipelines.CatBoostBinaryClassificationPipeline attribute*), 90
custom_hyperparameters (*evalml.pipelines.CatBoostMulticlassClassificationPipeline attribute*), 93
custom_hyperparameters (*evalml.pipelines.CatBoostRegressionPipeline attribute*), 152
custom_hyperparameters (*evalml.pipelines.ENBinaryPipeline attribute*), 97
custom_hyperparameters (*evalml.pipelines.ENMulticlassPipeline attribute*), 101
custom_hyperparameters (*evalml.pipelines.ENRegressionPipeline attribute*), 155
custom_hyperparameters (*evalml.pipelines.ETBinaryClassificationPipeline attribute*), 104
custom_hyperparameters

(*evalml.pipelines.ETMulticlassClassificationPipeline attribute*), 108
custom_hyperparameters (*evalml.pipelines.ETRegressionPipeline attribute*), 159
custom_hyperparameters (*evalml.pipelines.LinearRegressionPipeline attribute*), 162
custom_hyperparameters (*evalml.pipelines.LogisticRegressionBinaryPipeline attribute*), 112
custom_hyperparameters (*evalml.pipelines.LogisticRegressionMulticlassPipeline attribute*), 115
custom_hyperparameters (*evalml.pipelines.MeanBaselineRegressionPipeline attribute*), 173
custom_hyperparameters (*evalml.pipelines.ModeBaselineBinaryPipeline attribute*), 141
custom_hyperparameters (*evalml.pipelines.ModeBaselineMulticlassPipeline attribute*), 145
custom_hyperparameters (*evalml.pipelines.RFBinaryClassificationPipeline attribute*), 119
custom_hyperparameters (*evalml.pipelines.RFMulticlassClassificationPipeline attribute*), 123
custom_hyperparameters (*evalml.pipelines.RFRegressionPipeline attribute*), 148
custom_hyperparameters (*evalml.pipelines.XGBoostBinaryPipeline attribute*), 126
custom_hyperparameters (*evalml.pipelines.XGBoostMulticlassPipeline attribute*), 130
custom_hyperparameters (*evalml.pipelines.XGBoostRegressionPipeline attribute*), 166
custom_name (*evalml.pipelines.BaselineBinaryPipeline attribute*), 133
custom_name (*evalml.pipelines.BaselineMulticlassPipeline attribute*), 137
custom_name (*evalml.pipelines.BaselineRegressionPipeline attribute*), 169
custom_name (*evalml.pipelines.CatBoostBinaryClassificationPipeline attribute*), 90
custom_name (*evalml.pipelines.CatBoostMulticlassClassificationPipeline attribute*), 93
custom_name (*evalml.pipelines.CatBoostRegressionPipeline attribute*), 152
custom_name (*evalml.pipelines.ENBinaryPipeline attribute*), 97
custom_name (*evalml.pipelines.ENMulticlassPipeline attribute*), 100
custom_name (*evalml.pipelines.ENRegressionPipeline attribute*), 155
custom_name (*evalml.pipelines.ETBinaryClassificationPipeline attribute*), 104
custom_name (*evalml.pipelines.ETMulticlassClassificationPipeline attribute*), 108
custom_name (*evalml.pipelines.ETRegressionPipeline attribute*), 159
custom_name (*evalml.pipelines.LinearRegressionPipeline attribute*), 162
custom_name (*evalml.pipelines.LogisticRegressionBinaryPipeline attribute*), 111
custom_name (*evalml.pipelines.LogisticRegressionMulticlassPipeline attribute*), 115
custom_name (*evalml.pipelines.MeanBaselineRegressionPipeline attribute*), 173
custom_name (*evalml.pipelines.ModeBaselineBinaryPipeline attribute*), 141
custom_name (*evalml.pipelines.ModeBaselineMulticlassPipeline attribute*), 144
custom_name (*evalml.pipelines.RFBinaryClassificationPipeline attribute*), 119
custom_name (*evalml.pipelines.RFMulticlassClassificationPipeline attribute*), 122
custom_name (*evalml.pipelines.RFRegressionPipeline attribute*), 148
custom_name (*evalml.pipelines.XGBoostBinaryPipeline attribute*), 126
custom_name (*evalml.pipelines.XGBoostMulticlassPipeline attribute*), 130
custom_name (*evalml.pipelines.XGBoostRegressionPipeline attribute*), 166

D

DataCheck (*class in evalml.data_checks*), 298
DataCheckError (*class in evalml.data_checks*), 308
DataCheckMessage (*class in evalml.data_checks*), 307
DataCheckMessageType (*class in evalml.data_checks*), 310
DataChecks (*class in evalml.data_checks*), 305
DataCheckWarning (*class in evalml.data_checks*), 309
decision_function() (*evalml.objectives.AccuracyBinary method*), 246
decision_function() (*evalml.objectives.AUC method*), 249
decision_function() (*evalml.objectives.BalancedAccuracyBinary method*), 255

decision_function() (evalml.objectives.BinaryClassificationObjective method), 236
decision_function() (evalml.objectives.F1 method), 258
decision_function() (evalml.objectives.FraudCost method), 241
decision_function() (evalml.objectives.LeadScoring method), 243
decision_function() (evalml.objectives.LogLossBinary method), 263
decision_function() (evalml.objectives.MCCBinary method), 266
decision_function() (evalml.objectives.Precision method), 269
decision_function() (evalml.objectives.Recall method), 274
default_parameters (evalml.pipelines.BaselineBinaryPipeline attribute), 134
default_parameters (evalml.pipelines.BaselineMulticlassPipeline attribute), 137
default_parameters (evalml.pipelines.BaselineRegressionPipeline attribute), 169
default_parameters (evalml.pipelines.CatBoostBinaryClassificationPipeline attribute), 90
default_parameters (evalml.pipelines.CatBoostMulticlassClassificationPipeline attribute), 93
default_parameters (evalml.pipelines.CatBoostRegressionPipeline attribute), 152
default_parameters (evalml.pipelines.components.BaselineClassifier attribute), 218
default_parameters (evalml.pipelines.components.BaselineRegressor attribute), 232
default_parameters (evalml.pipelines.components.CatBoostClassifier attribute), 206
default_parameters (evalml.pipelines.components.CatBoostRegressor attribute), 220
default_parameters (evalml.pipelines.components.DropColumns attribute), 187
default_parameters (evalml.pipelines.components.ElasticNetClassifier attribute), 208
default_parameters (evalml.pipelines.components.ElasticNetRegressor attribute), 222
default_parameters (evalml.pipelines.components.ExtraTreesClassifier attribute), 210
default_parameters (evalml.pipelines.components.ExtraTreesRegressor attribute), 226
default_parameters (evalml.pipelines.components.LinearRegressor attribute), 224
default_parameters (evalml.pipelines.components.LogisticRegressionClassifier attribute), 214
default_parameters (evalml.pipelines.components.OneHotEncoder attribute), 191
default_parameters (evalml.pipelines.components.PerColumnImputer attribute), 194
default_parameters (evalml.pipelines.components.RandomForestClassifier attribute), 212
default_parameters (evalml.pipelines.components.RandomForestRegressor attribute), 228
default_parameters (evalml.pipelines.components.RFClassifierSelectFromModel attribute), 203
default_parameters (evalml.pipelines.components.RFRegressorSelectFromModel attribute), 200
default_parameters (evalml.pipelines.components.SelectColumns attribute), 189
default_parameters (evalml.pipelines.components.SimpleImputer attribute), 196
default_parameters (evalml.pipelines.components.StandardScaler attribute), 198
default_parameters (evalml.pipelines.components.XGBoostClassifier attribute), 216
default_parameters (evalml.pipelines.components.XGBoostRegressor attribute), 230
default_parameters (evalml.pipelines.ENBinaryPipeline attribute), 97
default_parameters

(*evalml.pipelines.ENMulticlassPipeline attribute*), 101
default_parameters (*evalml.pipelines.ENRegressionPipeline attribute*), 155
default_parameters (*evalml.pipelines.ETBinaryClassificationPipeline attribute*), 104
default_parameters (*evalml.pipelines.ETMulticlassClassificationPipeline attribute*), 108
default_parameters (*evalml.pipelines.ETRegressionPipeline attribute*), 159
default_parameters (*evalml.pipelines.LinearRegressionPipeline attribute*), 162
default_parameters (*evalml.pipelines.LogisticRegressionBinaryPipeline attribute*), 112
default_parameters (*evalml.pipelines.LogisticRegressionMulticlassPipeline attribute*), 115
default_parameters (*evalml.pipelines.MeanBaselineRegressionPipeline attribute*), 173
default_parameters (*evalml.pipelines.ModeBaselineBinaryPipeline attribute*), 141
default_parameters (*evalml.pipelines.ModeBaselineMulticlassPipeline attribute*), 145
default_parameters (*evalml.pipelines.RFBinaryClassificationPipeline attribute*), 119
default_parameters (*evalml.pipelines.RFMulticlassClassificationPipeline attribute*), 123
default_parameters (*evalml.pipelines.RFRegressionPipeline attribute*), 148
default_parameters (*evalml.pipelines.XGBoostBinaryPipeline attribute*), 126
default_parameters (*evalml.pipelines.XGBoostMulticlassPipeline attribute*), 130
default_parameters (*evalml.pipelines.XGBoostRegressionPipeline attribute*), 166
DefaultDataChecks (*class in evalml.data_checks*), 306
describe() (*evalml.pipelines.BaselineBinaryPipeline method*), 135
describe() (*evalml.pipelines.BaselineMulticlassPipeline method*), 138
describe() (*evalml.pipelines.BaselineRegressionPipeline method*), 171
describe() (*evalml.pipelines.BinaryClassificationPipeline method*), 81
describe() (*evalml.pipelines.CatBoostBinaryClassificationPipeline method*), 91
describe() (*evalml.pipelines.CatBoostMulticlassClassificationPipeline method*), 95
describe() (*evalml.pipelines.CatBoostRegressionPipeline method*), 153
describe() (*evalml.pipelines.ClassificationPipeline method*), 77
describe() (*evalml.pipelines.components.BaselineClassifier method*), 219
describe() (*evalml.pipelines.components.BaselineRegressor method*), 233
describe() (*evalml.pipelines.components.CatBoostClassifier method*), 207
describe() (*evalml.pipelines.components.CatBoostRegressor method*), 221
describe() (*evalml.pipelines.components.ComponentBase method*), 182
describe() (*evalml.pipelines.components.DropColumns method*), 188
describe() (*evalml.pipelines.components.ElasticNetClassifier method*), 209
describe() (*evalml.pipelines.components.ElasticNetRegressor method*), 223
describe() (*evalml.pipelines.components.Estimator method*), 186
describe() (*evalml.pipelines.components.ExtraTreesClassifier method*), 211
describe() (*evalml.pipelines.components.ExtraTreesRegressor method*), 227
describe() (*evalml.pipelines.components.LinearRegressor method*), 225
describe() (*evalml.pipelines.components.LogisticRegressionClassifier method*), 215
describe() (*evalml.pipelines.components.OneHotEncoder method*), 192
describe() (*evalml.pipelines.components.PerColumnImputer method*), 195
describe() (*evalml.pipelines.components.RandomForestClassifier method*), 213
describe() (*evalml.pipelines.components.RandomForestRegressor method*), 229
describe() (*evalml.pipelines.components.RFClassifierSelectFromMode method*), 204
describe() (*evalml.pipelines.components.RFRegressorSelectFromMode method*), 201
describe() (*evalml.pipelines.components.SelectColumns method*), 190

describe() (*evalml.pipelines.components.SimpleImputer method*), 197
describe() (*evalml.pipelines.components.StandardScaler method*), 199
describe() (*evalml.pipelines.components.Transformer method*), 184
describe() (*evalml.pipelines.components.XGBoostClassifier method*), 217
describe() (*evalml.pipelines.components.XGBoostRegressor method*), 231
describe() (*evalml.pipelines.ENBinaryPipeline method*), 98
describe() (*evalml.pipelines.ENMulticlassPipeline method*), 102
describe() (*evalml.pipelines.ENRegressionPipeline method*), 157
describe() (*evalml.pipelines.ETBinaryClassificationPipeline method*), 105
describe() (*evalml.pipelines.ETMulticlassClassificationPipeline method*), 109
describe() (*evalml.pipelines.ETRegressionPipeline method*), 160
describe() (*evalml.pipelines.LinearRegressionPipeline method*), 164
describe() (*evalml.pipelines.LogisticRegressionBinaryPipeline method*), 113
describe() (*evalml.pipelines.LogisticRegressionMulticlassPipeline method*), 116
describe() (*evalml.pipelines.MeanBaselineRegressionPipeline method*), 174
describe() (*evalml.pipelines.ModeBaselineBinaryPipeline method*), 142
describe() (*evalml.pipelines.ModeBaselineMulticlassPipeline method*), 146
describe() (*evalml.pipelines.MulticlassClassificationPipeline method*), 84
describe() (*evalml.pipelines.PipelineBase method*), 74
describe() (*evalml.pipelines.RegressionPipeline method*), 87
describe() (*evalml.pipelines.RFBinaryClassificationPipeline method*), 120
describe() (*evalml.pipelines.RFMulticlassClassificationPipeline method*), 124
describe() (*evalml.pipelines.RFRegressionPipeline method*), 150
describe() (*evalml.pipelines.XGBoostBinaryPipeline method*), 127
describe() (*evalml.pipelines.XGBoostMulticlassPipeline method*), 131
describe() (*evalml.pipelines.XGBoostRegressionPipeline method*), 167
describe_pipeline() (*evalml.automl.AutoMLSearch method*), 68
drop_nan_target_rows() (*in module evalml.preprocessing*), 64
DropColumns (*class in evalml.pipelines.components*), 187

E

ElasticNetClassifier (*class in evalml.pipelines.components*), 208
ElasticNetRegressor (*class in evalml.pipelines.components*), 222
ENBinaryPipeline (*class in evalml.pipelines*), 97
ENMulticlassPipeline (*class in evalml.pipelines*), 100
ENRegressionPipeline (*class in evalml.pipelines*), 155
Estimator (*class in evalml.pipelines.components*), 185
ETBinaryClassificationPipeline (*class in evalml.pipelines*), 104
ETMulticlassClassificationPipeline (*class in evalml.pipelines*), 108
ETRegressionPipeline (*class in evalml.pipelines*), 159
ExpVariance (*class in evalml.objectives*), 287
ExtraTreesClassifier (*class in evalml.pipelines.components*), 210
ExtraTreesRegressor (*class in evalml.pipelines.components*), 226

F

F1 (*class in evalml.objectives*), 257
F1Macro (*class in evalml.objectives*), 260
F1Micro (*class in evalml.objectives*), 259
F1Weighted (*class in evalml.objectives*), 261
fit() (*evalml.pipelines.BaselineBinaryPipeline method*), 135
fit() (*evalml.pipelines.BaselineMulticlassPipeline method*), 139
fit() (*evalml.pipelines.BaselineRegressionPipeline method*), 171
fit() (*evalml.pipelines.BinaryClassificationPipeline method*), 81
fit() (*evalml.pipelines.CatBoostBinaryClassificationPipeline method*), 91
fit() (*evalml.pipelines.CatBoostMulticlassClassificationPipeline method*), 95
fit() (*evalml.pipelines.CatBoostRegressionPipeline method*), 153
fit() (*evalml.pipelines.ClassificationPipeline method*), 78
fit() (*evalml.pipelines.components.BaselineClassifier method*), 219
fit() (*evalml.pipelines.components.BaselineRegressor method*), 234

fit() (*evalml.pipelines.components.CatBoostClassifier method*), 207
fit() (*evalml.pipelines.components.CatBoostRegressor method*), 221
fit() (*evalml.pipelines.components.ComponentBase method*), 183
fit() (*evalml.pipelines.components.DropColumns method*), 188
fit() (*evalml.pipelines.components.ElasticNetClassifier method*), 209
fit() (*evalml.pipelines.components.ElasticNetRegressor method*), 223
fit() (*evalml.pipelines.components.Estimator method*), 186
fit() (*evalml.pipelines.components.ExtraTreesClassifier method*), 211
fit() (*evalml.pipelines.components.ExtraTreesRegressor method*), 228
fit() (*evalml.pipelines.components.LinearRegressor method*), 225
fit() (*evalml.pipelines.components.LogisticRegressionClassifier method*), 215
fit() (*evalml.pipelines.components.OneHotEncoder method*), 193
fit() (*evalml.pipelines.components.PerColumnImputer method*), 195
fit() (*evalml.pipelines.components.RandomForestClassifier method*), 213
fit() (*evalml.pipelines.components.RandomForestRegressor method*), 229
fit() (*evalml.pipelines.components.RFClassifierSelectFromModel method*), 204
fit() (*evalml.pipelines.components.RFRegressorSelectFromModel method*), 202
fit() (*evalml.pipelines.components.SelectColumns method*), 190
fit() (*evalml.pipelines.components.SimpleImputer method*), 197
fit() (*evalml.pipelines.components.StandardScaler method*), 199
fit() (*evalml.pipelines.components.Transformer method*), 184
fit() (*evalml.pipelines.components.XGBoostClassifier method*), 217
fit() (*evalml.pipelines.components.XGBoostRegressor method*), 231
fit() (*evalml.pipelines.ENBinaryPipeline method*), 98
fit() (*evalml.pipelines.ENMulticlassPipeline method*), 102
fit() (*evalml.pipelines.ENRegressionPipeline method*), 157
fit() (*evalml.pipelines.ETBinaryClassificationPipeline method*), 106
fit() (*evalml.pipelines.ETMulticlassClassificationPipeline method*), 109
fit() (*evalml.pipelines.ETRegressionPipeline method*), 160
fit() (*evalml.pipelines.LinearRegressionPipeline method*), 164
fit() (*evalml.pipelines.LogisticRegressionBinaryPipeline method*), 113
fit() (*evalml.pipelines.LogisticRegressionMulticlassPipeline method*), 117
fit() (*evalml.pipelines.MeanBaselineRegressionPipeline method*), 174
fit() (*evalml.pipelines.ModeBaselineBinaryPipeline method*), 142
fit() (*evalml.pipelines.ModeBaselineMulticlassPipeline method*), 146
fit() (*evalml.pipelines.MulticlassClassificationPipeline method*), 84
fit() (*evalml.pipelines.PipelineBase method*), 74
fit() (*evalml.pipelines.RegressionPipeline method*), 87
fit() (*evalml.pipelines.RFBinaryClassificationPipeline method*), 120
fit() (*evalml.pipelines.RFMulticlassClassificationPipeline method*), 124
fit() (*evalml.pipelines.RFRegressionPipeline method*), 150
fit() (*evalml.pipelines.XGBoostBinaryPipeline method*), 128
fit() (*evalml.pipelines.XGBoostMulticlassPipeline method*), 131
fit() (*evalml.pipelines.XGBoostRegressionPipeline method*), 167
fit_transform() (*evalml.pipelines.components.DropColumns method*), 189
fit_transform() (*evalml.pipelines.components.OneHotEncoder method*), 193
fit_transform() (*evalml.pipelines.components.PerColumnImputer method*), 195
fit_transform() (*evalml.pipelines.components.RFClassifierSelectFro method*), 204
fit_transform() (*evalml.pipelines.components.RFRegressorSelectFro method*), 202
fit_transform() (*evalml.pipelines.components.SelectColumns method*), 191
fit_transform() (*evalml.pipelines.components.SimpleImputer method*), 198
fit_transform() (*evalml.pipelines.components.StandardScaler method*), 200
fit_transform() (*evalml.pipelines.components.Transformer method*), 184
FraudCost (*class in evalml.objectives*), 241

G

get_component() (*evalml.pipelines.BaselineBinaryPipeline method*), 135

get_component() (evalml.pipelines.BaselineMulticlassPipeline method), 139
get_component() (evalml.pipelines.BaselineRegressionPipeline method), 171
get_component() (evalml.pipelines.BinaryClassificationPipeline method), 81
get_component() (evalml.pipelines.CatBoostBinaryClassificationPipeline method), 91
get_component() (evalml.pipelines.CatBoostMulticlassClassificationPipeline method), 95
get_component() (evalml.pipelines.CatBoostRegressionPipeline method), 154
get_component() (evalml.pipelines.ClassificationPipeline method), 78
get_component() (evalml.pipelines.ENBinaryPipeline method), 99
get_component() (evalml.pipelines.ENMulticlassPipeline method), 102
get_component() (evalml.pipelines.ENRegressionPipeline method), 157
get_component() (evalml.pipelines.ETBinaryClassificationPipeline method), 106
get_component() (evalml.pipelines.ETMulticlassClassificationPipeline method), 110
get_component() (evalml.pipelines.ETRegressionPipeline method), 161
get_component() (evalml.pipelines.LinearRegressionPipeline method), 164
get_component() (evalml.pipelines.LogisticRegressionBinaryPipeline method), 113
get_component() (evalml.pipelines.LogisticRegressionMulticlassPipeline method), 117
get_component() (evalml.pipelines.MeanBaselineRegressionPipeline method), 175
get_component() (evalml.pipelines.ModeBaselineBinaryPipeline method), 143
get_component() (evalml.pipelines.ModeBaselineMulticlassPipeline method), 146
get_component() (evalml.pipelines.MulticlassClassificationPipeline method), 84
get_component() (evalml.pipelines.PipelineBase method), 75
get_component() (evalml.pipelines.RegressionPipeline method), 88
get_component() (evalml.pipelines.RFBinaryClassificationPipeline method), 121
get_component() (evalml.pipelines.RFMulticlassClassificationPipeline method), 124
get_component() (evalml.pipelines.RFRegressionPipeline method), 150
get_component() (evalml.pipelines.XGBoostBinaryPipeline method), 128
get_component() (evalml.pipelines.XGBoostMulticlassPipeline method), 132
get_component() (evalml.pipelines.XGBoostRegressionPipeline method), 168
get_estimators() (in module evalml.pipelines), 177
get_feature_names() (evalml.pipelines.components.OneHotEncoder method), 193
get_indices() (evalml.pipelines.components.RFClassifierSelectFromModel method), 205
get_indices() (evalml.pipelines.components.RFRegressorSelectFromModel method), 202
get_names() (evalml.pipelines.components.RFClassifierSelectFromModel method), 205
get_names() (evalml.pipelines.components.RFRegressorSelectFromModel method), 202
get_pipeline() (evalml.automl.AutoMLSearch method), 68
get_pipelines() (in module evalml.pipelines), 176
get_random_seed() (in module evalml.utils), 311
get_random_state() (in module evalml.utils), 310
graph() (evalml.pipelines.BaselineBinaryPipeline method), 135
graph() (evalml.pipelines.BaselineMulticlassPipeline method), 139
graph() (evalml.pipelines.BaselineRegressionPipeline method), 171
graph() (evalml.pipelines.BinaryClassificationPipeline method), 81
graph() (evalml.pipelines.CatBoostBinaryClassificationPipeline method), 92
graph() (evalml.pipelines.CatBoostMulticlassClassificationPipeline method), 95
graph() (evalml.pipelines.CatBoostRegressionPipeline method), 154
graph() (evalml.pipelines.ClassificationPipeline method), 78
graph() (evalml.pipelines.ENBinaryPipeline method), 99
graph() (evalml.pipelines.ENMulticlassPipeline method), 102
graph() (evalml.pipelines.ENRegressionPipeline method), 157
graph() (evalml.pipelines.ETBinaryClassificationPipeline method), 106
graph() (evalml.pipelines.ETMulticlassClassificationPipeline method), 110
graph() (evalml.pipelines.ETRegressionPipeline method), 161
graph() (evalml.pipelines.LinearRegressionPipeline method), 164
graph() (evalml.pipelines.LogisticRegressionBinaryPipeline method), 113
graph() (evalml.pipelines.LogisticRegressionMulticlassPipeline method), 117

graph() (*evalml.pipelines.MeanBaselineRegressionPipeline method*), 175
graph() (*evalml.pipelines.ModeBaselineBinaryPipeline method*), 143
graph() (*evalml.pipelines.ModeBaselineMulticlassPipeline method*), 146
graph() (*evalml.pipelines.MulticlassClassificationPipeline method*), 85
graph() (*evalml.pipelines.PipelineBase method*), 75
graph() (*evalml.pipelines.RegressionPipeline method*), 88
graph() (*evalml.pipelines.RFBinaryClassificationPipeline method*), 121
graph() (*evalml.pipelines.RFMulticlassClassificationPipeline method*), 124
graph() (*evalml.pipelines.RFRegressionPipeline method*), 150
graph() (*evalml.pipelines.XGBoostBinaryPipeline method*), 128
graph() (*evalml.pipelines.XGBoostMulticlassPipeline method*), 132
graph() (*evalml.pipelines.XGBoostRegressionPipeline method*), 168
graph_confusion_matrix() (*in module evalml.pipelines*), 180
graph_feature_importance() (*evalml.pipelines.BaselineBinaryPipeline method*), 136
graph_feature_importance() (*evalml.pipelines.BaselineMulticlassPipeline method*), 139
graph_feature_importance() (*evalml.pipelines.BaselineRegressionPipeline method*), 171
graph_feature_importance() (*evalml.pipelines.BinaryClassificationPipeline method*), 82
graph_feature_importance() (*evalml.pipelines.CatBoostBinaryClassificationPipeline method*), 92
graph_feature_importance() (*evalml.pipelines.CatBoostMulticlassClassificationPipeline method*), 95
graph_feature_importance() (*evalml.pipelines.CatBoostRegressionPipeline method*), 154
graph_feature_importance() (*evalml.pipelines.ClassificationPipeline method*), 78
graph_feature_importance() (*evalml.pipelines.ENBinaryPipeline method*), 99
graph_feature_importance() (*evalml.pipelines.ENMulticlassPipeline method*), 103
graph_feature_importance() (*evalml.pipelines.ENRegressionPipeline method*), 157
graph_feature_importance() (*evalml.pipelines.ETBinaryClassificationPipeline method*), 106
graph_feature_importance() (*evalml.pipelines.ETMulticlassClassificationPipeline method*), 110
graph_feature_importance() (*evalml.pipelines.ETRegressionPipeline method*), 161
graph_feature_importance() (*evalml.pipelines.LinearRegressionPipeline method*), 164
graph_feature_importance() (*evalml.pipelines.LogisticRegressionBinaryPipeline method*), 114
graph_feature_importance() (*evalml.pipelines.LogisticRegressionMulticlassPipeline method*), 117
graph_feature_importance() (*evalml.pipelines.MeanBaselineRegressionPipeline method*), 175
graph_feature_importance() (*evalml.pipelines.ModeBaselineBinaryPipeline method*), 143
graph_feature_importance() (*evalml.pipelines.ModeBaselineMulticlassPipeline method*), 147
graph_feature_importance() (*evalml.pipelines.MulticlassClassificationPipeline method*), 85
graph_feature_importance() (*evalml.pipelines.PipelineBase method*), 75
graph_feature_importance() (*evalml.pipelines.RegressionPipeline method*), 88
graph_feature_importance() (*evalml.pipelines.RFBinaryClassificationPipeline method*), 121
graph_feature_importance() (*evalml.pipelines.RFMulticlassClassificationPipeline method*), 125
graph_feature_importance() (*evalml.pipelines.RFRegressionPipeline method*), 150
graph_feature_importance() (*evalml.pipelines.XGBoostBinaryPipeline method*), 128
graph_feature_importance() (*evalml.pipelines.XGBoostMulticlassPipeline*

method), 132
graph_feature_importance() (*evalml.pipelines.XGBoostRegressionPipeline method*), 168
graph_permutation_importance() (*in module evalml.pipelines*), 181
graph_precision_recall_curve() (*in module evalml.pipelines*), 178
graph_roc_curve() (*in module evalml.pipelines*), 179
GridSearchTuner (*class in evalml.tuners*), 295

H

handle_problem_types() (*in module evalml.problem_types*), 291
HighlyNullDataCheck (*class in evalml.data_checks*), 300
hyperparameter_ranges (*evalml.pipelines.components.BaselineClassifier attribute*), 218
hyperparameter_ranges (*evalml.pipelines.components.BaselineRegressor attribute*), 232
hyperparameter_ranges (*evalml.pipelines.components.CatBoostClassifier attribute*), 206
hyperparameter_ranges (*evalml.pipelines.components.CatBoostRegressor attribute*), 220
hyperparameter_ranges (*evalml.pipelines.components.DropColumns attribute*), 187
hyperparameter_ranges (*evalml.pipelines.components.ElasticNetClassifier attribute*), 208
hyperparameter_ranges (*evalml.pipelines.components.ElasticNetRegressor attribute*), 222
hyperparameter_ranges (*evalml.pipelines.components.ExtraTreesClassifier attribute*), 210
hyperparameter_ranges (*evalml.pipelines.components.ExtraTreesRegressor attribute*), 226
hyperparameter_ranges (*evalml.pipelines.components.LinearRegressor attribute*), 224
hyperparameter_ranges (*evalml.pipelines.components.LogisticRegressionClassifier attribute*), 214
hyperparameter_ranges (*evalml.pipelines.components.OneHotEncoder attribute*), 191
hyperparameter_ranges (*evalml.pipelines.components.PerColumnImputer attribute*), 194
hyperparameter_ranges (*evalml.pipelines.components.RandomForestClassifier attribute*), 212
hyperparameter_ranges (*evalml.pipelines.components.RandomForestRegressor attribute*), 228
hyperparameter_ranges (*evalml.pipelines.components.RFClassifierSelectFromModel attribute*), 203
hyperparameter_ranges (*evalml.pipelines.components.RFRegressorSelectFromModel attribute*), 200
hyperparameter_ranges (*evalml.pipelines.components.SelectColumns attribute*), 189
hyperparameter_ranges (*evalml.pipelines.components.SimpleImputer attribute*), 196
hyperparameter_ranges (*evalml.pipelines.components.StandardScaler attribute*), 198
hyperparameter_ranges (*evalml.pipelines.components.XGBoostClassifier attribute*), 216
hyperparameter_ranges (*evalml.pipelines.components.XGBoostRegressor attribute*), 230
hyperparameters (*evalml.pipelines.BaselineBinaryPipeline attribute*), 133
hyperparameters (*evalml.pipelines.BaselineMulticlassPipeline attribute*), 137
hyperparameters (*evalml.pipelines.BaselineRegressionPipeline attribute*), 169
hyperparameters (*evalml.pipelines.CatBoostBinaryClassificationPipel attribute*), 90
hyperparameters (*evalml.pipelines.CatBoostMulticlassClassificationP attribute*), 93
hyperparameters (*evalml.pipelines.CatBoostRegressionPipeline attribute*), 152
hyperparameters (*evalml.pipelines.ENBinaryPipeline attribute*), 97
hyperparameters (*evalml.pipelines.ENMulticlassPipeline attribute*), 100
hyperparameters (*evalml.pipelines.ENRegressionPipeline attribute*), 155
hyperparameters (*evalml.pipelines.ETBinaryClassificationPipeline attribute*), 104
hyperparameters (*evalml.pipelines.ETMulticlassClassificationPipeline attribute*), 108
hyperparameters (*evalml.pipelines.ETRegressionPipeline attribute*), 159
hyperparameters (*evalml.pipelines.LinearRegressionPipeline*

attribute), 162
hyperparameters (*evalml.pipelines.LogisticRegressionBinaryPipeline attribute*), 111
hyperparameters (*evalml.pipelines.LogisticRegressionMulticlassPipeline attribute*), 115
hyperparameters (*evalml.pipelines.MeanBaselineRegressionPipeline attribute*), 173
hyperparameters (*evalml.pipelines.ModeBaselineBinaryPipeline attribute*), 141
hyperparameters (*evalml.pipelines.ModeBaselineMulticlassPipeline attribute*), 144
hyperparameters (*evalml.pipelines.RFBinaryClassificationPipeline attribute*), 119
hyperparameters (*evalml.pipelines.RFMulticlassClassificationPipeline attribute*), 122
hyperparameters (*evalml.pipelines.RFRegressionPipeline attribute*), 148
hyperparameters (*evalml.pipelines.XGBoostBinaryPipeline attribute*), 126
hyperparameters (*evalml.pipelines.XGBoostMulticlassPipeline attribute*), 130
hyperparameters (*evalml.pipelines.XGBoostRegressionPipeline attribute*), 166

I

IDColumnsDataCheck (*class in evalml.data_checks*), 301
import_or_raise() (*in module evalml.utils*), 310
InvalidTargetDataCheck (*class in evalml.data_checks*), 299
is_search_space_exhausted() (*evalml.tuners.GridSearchTuner method*), 296
is_search_space_exhausted() (*evalml.tuners.RandomSearchTuner method*), 297
is_search_space_exhausted() (*evalml.tuners.SKOptTuner method*), 294
is_search_space_exhausted() (*evalml.tuners.Tuner method*), 293
IterativeAlgorithm (*class in evalml.automl.automl_algorithm*), 71

L

label_distribution() (*in module evalml.preprocessing*), 64
LabelLeakageDataCheck (*class in evalml.data_checks*), 302
LeadScoring (*class in evalml.objectives*), 243
LinearRegressionPipeline (*class in evalml.pipelines*), 162
LinearRegressor (*class in evalml.pipelines.components*), 224
list_model_families() (*in module evalml.pipelines*), 177
load() (*evalml.automl.AutoMLSearch static method*), 88
load() (*evalml.pipelines.BaselineBinaryPipeline static method*), 136
load() (*evalml.pipelines.BaselineMulticlassPipeline static method*), 139
load() (*evalml.pipelines.BaselineRegressionPipeline static method*), 172
load() (*evalml.pipelines.BinaryClassificationPipeline static method*), 82
load() (*evalml.pipelines.CatBoostBinaryClassificationPipeline static method*), 92
load() (*evalml.pipelines.CatBoostMulticlassClassificationPipeline static method*), 96
load() (*evalml.pipelines.CatBoostRegressionPipeline static method*), 154
load() (*evalml.pipelines.ClassificationPipeline static method*), 78
load() (*evalml.pipelines.ENBinaryPipeline static method*), 99
load() (*evalml.pipelines.ENMulticlassPipeline static method*), 103
load() (*evalml.pipelines.ENRegressionPipeline static method*), 158
load() (*evalml.pipelines.ETBinaryClassificationPipeline static method*), 106
load() (*evalml.pipelines.ETMulticlassClassificationPipeline static method*), 110
load() (*evalml.pipelines.ETRegressionPipeline static method*), 161
load() (*evalml.pipelines.LinearRegressionPipeline static method*), 165
load() (*evalml.pipelines.LogisticRegressionBinaryPipeline static method*), 114
load() (*evalml.pipelines.LogisticRegressionMulticlassPipeline static method*), 117
load() (*evalml.pipelines.MeanBaselineRegressionPipeline static method*), 175
load() (*evalml.pipelines.ModeBaselineBinaryPipeline static method*), 143
load() (*evalml.pipelines.ModeBaselineMulticlassPipeline static method*), 147
load() (*evalml.pipelines.MulticlassClassificationPipeline static method*), 85
load() (*evalml.pipelines.PipelineBase static method*), 75
load() (*evalml.pipelines.RegressionPipeline static method*), 88
load() (*evalml.pipelines.RFBinaryClassificationPipeline static method*), 121
load() (*evalml.pipelines.RFMulticlassClassificationPipeline static method*), 125

load() (*evalml.pipelines.RFRegressionPipeline static method*), 151
load() (*evalml.pipelines.XGBoostBinaryPipeline static method*), 128
load() (*evalml.pipelines.XGBoostMulticlassPipeline static method*), 132
load() (*evalml.pipelines.XGBoostRegressionPipeline static method*), 168
load_breast_cancer() (*in module evalml.demos*), 63
load_data() (*in module evalml.preprocessing*), 64
load_diabetes() (*in module evalml.demos*), 63
load_fraud() (*in module evalml.demos*), 63
load_wine() (*in module evalml.demos*), 63
LogisticRegressionBinaryPipeline (*class in evalml.pipelines*), 111
LogisticRegressionClassifier (*class in evalml.pipelines.components*), 214
LogisticRegressionMulticlassPipeline (*class in evalml.pipelines*), 115
LogLossBinary (*class in evalml.objectives*), 263
LogLossMulticlass (*class in evalml.objectives*), 264

M

MAE (*class in evalml.objectives*), 281
make_pipeline() (*in module evalml.pipelines*), 177
MaxError (*class in evalml.objectives*), 286
MCCBinary (*class in evalml.objectives*), 266
MCCMulticlass (*class in evalml.objectives*), 267
MeanBaselineRegressionPipeline (*class in evalml.pipelines*), 173
MeanSquaredLogError (*class in evalml.objectives*), 283
MedianAE (*class in evalml.objectives*), 284
message_type (*evalml.data_checks.DataCheckError attribute*), 308
message_type (*evalml.data_checks.DataCheckMessage attribute*), 307
message_type (*evalml.data_checks.DataCheckWarning attribute*), 309
ModeBaselineBinaryPipeline (*class in evalml.pipelines*), 141
ModeBaselineMulticlassPipeline (*class in evalml.pipelines*), 144
model_family (*evalml.pipelines.BaselineBinaryPipeline attribute*), 133
model_family (*evalml.pipelines.BaselineMulticlassPipeline attribute*), 137
model_family (*evalml.pipelines.BaselineRegressionPipeline attribute*), 169
model_family (*evalml.pipelines.CatBoostBinaryClassificationPipeline attribute*), 90
model_family (*evalml.pipelines.CatBoostMulticlassClassificationPipeli attribute*), 93
model_family (*evalml.pipelines.CatBoostRegressionPipeline attribute*), 152
model_family (*evalml.pipelines.components.BaselineClassifier attribute*), 218
model_family (*evalml.pipelines.components.BaselineRegressor attribute*), 232
model_family (*evalml.pipelines.components.CatBoostClassifier attribute*), 206
model_family (*evalml.pipelines.components.CatBoostRegressor attribute*), 220
model_family (*evalml.pipelines.components.DropColumns attribute*), 187
model_family (*evalml.pipelines.components.ElasticNetClassifier attribute*), 208
model_family (*evalml.pipelines.components.ElasticNetRegressor attribute*), 222
model_family (*evalml.pipelines.components.ExtraTreesClassifier attribute*), 210
model_family (*evalml.pipelines.components.ExtraTreesRegressor attribute*), 226
model_family (*evalml.pipelines.components.LinearRegressor attribute*), 224
model_family (*evalml.pipelines.components.LogisticRegressionClassifi attribute*), 214
model_family (*evalml.pipelines.components.OneHotEncoder attribute*), 191
model_family (*evalml.pipelines.components.PerColumnImputer attribute*), 194
model_family (*evalml.pipelines.components.RandomForestClassifier attribute*), 212
model_family (*evalml.pipelines.components.RandomForestRegressor attribute*), 228
model_family (*evalml.pipelines.components.RFClassifierSelectFromMo attribute*), 203
model_family (*evalml.pipelines.components.RFRegressorSelectFromMo attribute*), 200
model_family (*evalml.pipelines.components.SelectColumns attribute*), 189
model_family (*evalml.pipelines.components.SimpleImputer attribute*), 196
model_family (*evalml.pipelines.components.StandardScaler attribute*), 198
model_family (*evalml.pipelines.components.XGBoostClassifier attribute*), 216
model_family (*evalml.pipelines.components.XGBoostRegressor attribute*), 230
model_family (*evalml.pipelines.ENBinaryPipeline attribute*), 97
model_family (*evalml.pipelines.ENMulticlassPipeline attribute*), 100
model_family (*evalml.pipelines.ENRegressionPipeline attribute*), 155

model_family (*evalml.pipelines.ETBinaryClassificationPipeline attribute*), 104
model_family (*evalml.pipelines.ETMulticlassClassificationPipeline attribute*), 108
model_family (*evalml.pipelines.ETRegressionPipeline attribute*), 159
model_family (*evalml.pipelines.LinearRegressionPipeline attribute*), 162
model_family (*evalml.pipelines.LogisticRegressionBinaryPipeline attribute*), 111
model_family (*evalml.pipelines.LogisticRegressionMulticlassPipeline attribute*), 115
model_family (*evalml.pipelines.MeanBaselineRegressionPipeline attribute*), 173
model_family (*evalml.pipelines.ModeBaselineBinaryPipeline attribute*), 141
model_family (*evalml.pipelines.ModeBaselineMulticlassPipeline attribute*), 144
model_family (*evalml.pipelines.RFBinaryClassificationPipeline attribute*), 119
model_family (*evalml.pipelines.RFMulticlassClassificationPipeline attribute*), 122
model_family (*evalml.pipelines.RFRegressionPipeline attribute*), 148
model_family (*evalml.pipelines.XGBoostBinaryPipeline attribute*), 126
model_family (*evalml.pipelines.XGBoostMulticlassPipeline attribute*), 130
model_family (*evalml.pipelines.XGBoostRegressionPipeline attribute*), 166
ModelFamily (*class in evalml.model_family*), 291
MSE (*class in evalml.objectives*), 282
MulticlassClassificationObjective (*class in evalml.objectives*), 238
MulticlassClassificationPipeline (*class in evalml.pipelines*), 83

N

name (*evalml.data_checks.DataCheck attribute*), 298
name (*evalml.data_checks.HighlyNullDataCheck attribute*), 300
name (*evalml.data_checks.IDColumnsDataCheck attribute*), 301
name (*evalml.data_checks.InvalidTargetDataCheck attribute*), 299
name (*evalml.data_checks.LabelLeakageDataCheck attribute*), 302
name (*evalml.data_checks.OutliersDataCheck attribute*), 304
name (*evalml.pipelines.BaselineBinaryPipeline attribute*), 133
name (*evalml.pipelines.BaselineMulticlassPipeline attribute*), 137
name (*evalml.pipelines.BaselineRegressionPipeline attribute*), 169
name (*evalml.pipelines.CatBoostBinaryClassificationPipeline attribute*), 90
name (*evalml.pipelines.CatBoostMulticlassClassificationPipeline attribute*), 93
name (*evalml.pipelines.CatBoostRegressionPipeline attribute*), 152
name (*evalml.pipelines.components.BaselineClassifier attribute*), 218
name (*evalml.pipelines.components.BaselineRegressor attribute*), 232
name (*evalml.pipelines.components.CatBoostClassifier attribute*), 206
name (*evalml.pipelines.components.CatBoostRegressor attribute*), 220
name (*evalml.pipelines.components.DropColumns attribute*), 187
name (*evalml.pipelines.components.ElasticNetClassifier attribute*), 208
name (*evalml.pipelines.components.ElasticNetRegressor attribute*), 222
name (*evalml.pipelines.components.ExtraTreesClassifier attribute*), 210
name (*evalml.pipelines.components.ExtraTreesRegressor attribute*), 226
name (*evalml.pipelines.components.LinearRegressor attribute*), 224
name (*evalml.pipelines.components.LogisticRegressionClassifier attribute*), 214
name (*evalml.pipelines.components.OneHotEncoder attribute*), 191
name (*evalml.pipelines.components.PerColumnImputer attribute*), 194
name (*evalml.pipelines.components.RandomForestClassifier attribute*), 212
name (*evalml.pipelines.components.RandomForestRegressor attribute*), 228
name (*evalml.pipelines.components.RFClassifierSelectFromModel attribute*), 203
name (*evalml.pipelines.components.RFRegressorSelectFromModel attribute*), 200
name (*evalml.pipelines.components.SelectColumns attribute*), 189
name (*evalml.pipelines.components.SimpleImputer attribute*), 196
name (*evalml.pipelines.components.StandardScaler attribute*), 198
name (*evalml.pipelines.components.XGBoostClassifier attribute*), 216
name (*evalml.pipelines.components.XGBoostRegressor attribute*), 230
name (*evalml.pipelines.ENBinaryPipeline attribute*), 97
name (*evalml.pipelines.ENMulticlassPipeline attribute*),

100
name (evalml.pipelines.ENRegressionPipeline attribute), 155
name (evalml.pipelines.ETBinaryClassificationPipeline attribute), 104
name (evalml.pipelines.ETMulticlassClassificationPipeline attribute), 108
name (evalml.pipelines.ETRegressionPipeline attribute), 159
name (evalml.pipelines.LinearRegressionPipeline attribute), 162
name (evalml.pipelines.LogisticRegressionBinaryPipeline attribute), 111
name (evalml.pipelines.LogisticRegressionMulticlassPipeline attribute), 115
name (evalml.pipelines.MeanBaselineRegressionPipeline attribute), 173
name (evalml.pipelines.ModeBaselineBinaryPipeline attribute), 141
name (evalml.pipelines.ModeBaselineMulticlassPipeline attribute), 144
name (evalml.pipelines.RFBinaryClassificationPipeline attribute), 119
name (evalml.pipelines.RFMulticlassClassificationPipeline attribute), 122
name (evalml.pipelines.RFRegressionPipeline attribute), 148
name (evalml.pipelines.XGBoostBinaryPipeline attribute), 126
name (evalml.pipelines.XGBoostMulticlassPipeline attribute), 130
name (evalml.pipelines.XGBoostRegressionPipeline attribute), 166
next_batch() (evalml.automl.automl_algorithm.AutoMLAlgorithm method), 71
next_batch() (evalml.automl.automl_algorithm.IterativeAlgorithm method), 72
normalize_confusion_matrix() (in module evalml.pipelines), 180
number_of_features() (in module evalml.preprocessing), 65

O

objective_function() (evalml.objectives.AccuracyBinary method), 246
objective_function() (evalml.objectives.AccuracyMulticlass method), 248
objective_function() (evalml.objectives.AUC method), 250
objective_function() (evalml.objectives.AUCMacro method), 251
objective_function() (evalml.objectives.AUCMicro method), 252
objective_function() (evalml.objectives.AUCWeighted method), 253
objective_function() (evalml.objectives.BalancedAccuracyBinary method), 255
objective_function() (evalml.objectives.BalancedAccuracyMulticlass method), 256
objective_function() (evalml.objectives.BinaryClassificationObjective class method), 237
objective_function() (evalml.objectives.ExpVariance method), 287
objective_function() (evalml.objectives.F1 method), 258
objective_function() (evalml.objectives.F1Macro method), 260
objective_function() (evalml.objectives.F1Micro method), 259
objective_function() (evalml.objectives.F1Weighted method), 262
objective_function() (evalml.objectives.FraudCost method), 242
objective_function() (evalml.objectives.LeadScoring method), 244
objective_function() (evalml.objectives.LogLossBinary method), 263
objective_function() (evalml.objectives.LogLossMulticlass method), 265
objective_function() (evalml.objectives.MAE method), 281
objective_function() (evalml.objectives.MaxError method), 286
objective_function() (evalml.objectives.MCCBinary method), 266
objective_function() (evalml.objectives.MCCMulticlass method), 268
objective_function() (evalml.objectives.MeanSquaredLogError method), 284
objective_function() (evalml.objectives.MedianAE method), 285
objective_function() (evalml.objectives.MSE method), 282

objective_function() (evalml.objectives.MulticlassClassificationObjective class method), 238
objective_function() (evalml.objectives.ObjectiveBase class method), 235
objective_function() (evalml.objectives.Precision method), 269
objective_function() (evalml.objectives.PrecisionMacro method), 272
objective_function() (evalml.objectives.PrecisionMicro method), 271
objective_function() (evalml.objectives.PrecisionWeighted method), 273
objective_function() (evalml.objectives.R2 method), 280
objective_function() (evalml.objectives.Recall method), 275
objective_function() (evalml.objectives.RecallMacro method), 277
objective_function() (evalml.objectives.RecallMicro method), 276
objective_function() (evalml.objectives.RecallWeighted method), 278
objective_function() (evalml.objectives.RegressionObjective class method), 240
objective_function() (evalml.objectives.RootMeanSquaredError method), 288
objective_function() (evalml.objectives.RootMeanSquaredLogError method), 290
ObjectiveBase (class in evalml.objectives), 235
OneHotEncoder (class in evalml.pipelines.components), 191
optimize_threshold() (evalml.objectives.AccuracyBinary method), 247
optimize_threshold() (evalml.objectives.AUC method), 250
optimize_threshold() (evalml.objectives.BalancedAccuracyBinary method), 255
optimize_threshold() (evalml.objectives.BinaryClassificationObjective method), 237
optimize_threshold() (evalml.objectives.F1 method), 258
optimize_threshold() (evalml.objectives.FraudCost method), 242
optimize_threshold() (evalml.objectives.LeadScoring method), 244
optimize_threshold() (evalml.objectives.LogLossBinary method), 264
optimize_threshold() (evalml.objectives.MCCBinary method), 267
optimize_threshold() (evalml.objectives.Precision method), 270
optimize_threshold() (evalml.objectives.Recall method), 275
OutliersDataCheck (class in evalml.data_checks), 304

P

PerColumnImputer (class in evalml.pipelines.components), 194
PipelineBase (class in evalml.pipelines), 73
Precision (class in evalml.objectives), 269
precision_recall_curve() (in module evalml.pipelines), 178
PrecisionMacro (class in evalml.objectives), 272
PrecisionMicro (class in evalml.objectives), 270
PrecisionWeighted (class in evalml.objectives), 273
predict() (evalml.pipelines.BaselineBinaryPipeline method), 136
predict() (evalml.pipelines.BaselineMulticlassPipeline method), 140
predict() (evalml.pipelines.BaselineRegressionPipeline method), 172
predict() (evalml.pipelines.BinaryClassificationPipeline method), 82
predict() (evalml.pipelines.CatBoostBinaryClassificationPipeline method), 92
predict() (evalml.pipelines.CatBoostMulticlassClassificationPipeline method), 96
predict() (evalml.pipelines.CatBoostRegressionPipeline method), 154
predict() (evalml.pipelines.ClassificationPipeline method), 79
predict() (evalml.pipelines.components.BaselineClassifier method), 219
predict() (evalml.pipelines.components.BaselineRegressor method), 234
predict() (evalml.pipelines.components.CatBoostClassifier method), 207
predict() (evalml.pipelines.components.CatBoostRegressor method), 222

predict() (*evalml.pipelines.components.ElasticNetClassifier method*), 209
predict() (*evalml.pipelines.components.ElasticNetRegressor method*), 224
predict() (*evalml.pipelines.components.Estimator method*), 186
predict() (*evalml.pipelines.components.ExtraTreesClassifier method*), 211
predict() (*evalml.pipelines.components.ExtraTreesRegressor method*), 228
predict() (*evalml.pipelines.components.LinearRegressor method*), 226
predict() (*evalml.pipelines.components.LogisticRegressionClassifier method*), 215
predict() (*evalml.pipelines.components.RandomForestClassifier method*), 213
predict() (*evalml.pipelines.components.RandomForestRegressor method*), 230
predict() (*evalml.pipelines.components.XGBoostClassifier method*), 217
predict() (*evalml.pipelines.components.XGBoostRegressor method*), 232
predict() (*evalml.pipelines.ENBinaryPipeline method*), 99
predict() (*evalml.pipelines.ENMulticlassPipeline method*), 103
predict() (*evalml.pipelines.ENRegressionPipeline method*), 158
predict() (*evalml.pipelines.ETBinaryClassificationPipeline method*), 107
predict() (*evalml.pipelines.ETMulticlassClassificationPipeline method*), 110
predict() (*evalml.pipelines.ETRegressionPipeline method*), 161
predict() (*evalml.pipelines.LinearRegressionPipeline method*), 165
predict() (*evalml.pipelines.LogisticRegressionBinaryPipeline method*), 114
predict() (*evalml.pipelines.LogisticRegressionMulticlassPipeline method*), 118
predict() (*evalml.pipelines.MeanBaselineRegressionPipeline method*), 175
predict() (*evalml.pipelines.ModeBaselineBinaryPipeline method*), 143
predict() (*evalml.pipelines.ModeBaselineMulticlassPipeline method*), 147
predict() (*evalml.pipelines.MulticlassClassificationPipeline method*), 85
predict() (*evalml.pipelines.PipelineBase method*), 75
predict() (*evalml.pipelines.RegressionPipeline method*), 88
predict() (*evalml.pipelines.RFBinaryClassificationPipeline method*), 121
predict() (*evalml.pipelines.RFMulticlassClassificationPipeline method*), 125
predict() (*evalml.pipelines.RFRegressionPipeline method*), 151
predict() (*evalml.pipelines.XGBoostBinaryPipeline method*), 129
predict() (*evalml.pipelines.XGBoostMulticlassPipeline method*), 132
predict() (*evalml.pipelines.XGBoostRegressionPipeline method*), 168
predict_proba() (*evalml.pipelines.BaselineBinaryPipeline method*), 136
predict_proba() (*evalml.pipelines.BaselineMulticlassPipeline method*), 140
predict_proba() (*evalml.pipelines.BinaryClassificationPipeline method*), 82
predict_proba() (*evalml.pipelines.CatBoostBinaryClassificationPipeline method*), 92
predict_proba() (*evalml.pipelines.CatBoostMulticlassClassificationPipeline method*), 96
predict_proba() (*evalml.pipelines.ClassificationPipeline method*), 79
predict_proba() (*evalml.pipelines.components.BaselineClassifier method*), 220
predict_proba() (*evalml.pipelines.components.BaselineRegressor method*), 234
predict_proba() (*evalml.pipelines.components.CatBoostClassifier method*), 207
predict_proba() (*evalml.pipelines.components.CatBoostRegressor method*), 222
predict_proba() (*evalml.pipelines.components.ElasticNetClassifier method*), 209
predict_proba() (*evalml.pipelines.components.ElasticNetRegressor method*), 224
predict_proba() (*evalml.pipelines.components.Estimator method*), 186
predict_proba() (*evalml.pipelines.components.ExtraTreesClassifier method*), 211
predict_proba() (*evalml.pipelines.components.ExtraTreesRegressor method*), 228
predict_proba() (*evalml.pipelines.components.LinearRegressor method*), 226
predict_proba() (*evalml.pipelines.components.LogisticRegressionClassifier method*), 215
predict_proba() (*evalml.pipelines.components.RandomForestClassifier method*), 213
predict_proba() (*evalml.pipelines.components.RandomForestRegressor method*), 230
predict_proba() (*evalml.pipelines.components.XGBoostClassifier method*), 217
predict_proba() (*evalml.pipelines.components.XGBoostRegressor method*), 232
predict_proba() (*evalml.pipelines.ENBinaryPipeline method*), 100
predict_proba() (*evalml.pipelines.ENMulticlassPipeline*

method), 103
predict_proba() (*evalml.pipelines.ETBinaryClassificationPipeline method*), 107
predict_proba() (*evalml.pipelines.ETMulticlassClassificationPipeline method*), 111
predict_proba() (*evalml.pipelines.LogisticRegressionBinaryPipeline method*), 114
predict_proba() (*evalml.pipelines.LogisticRegressionMulticlassPipeline method*), 118
predict_proba() (*evalml.pipelines.ModeBaselineBinaryPipeline method*), 144
predict_proba() (*evalml.pipelines.ModeBaselineMulticlassPipeline method*), 147
predict_proba() (*evalml.pipelines.MulticlassClassificationPipeline method*), 85
predict_proba() (*evalml.pipelines.RFBinaryClassificationPipeline method*), 122
predict_proba() (*evalml.pipelines.RFMulticlassClassificationPipeline method*), 125
predict_proba() (*evalml.pipelines.XGBoostBinaryPipeline method*), 129
predict_proba() (*evalml.pipelines.XGBoostMulticlassPipeline method*), 133
problem_type (*evalml.pipelines.BaselineBinaryPipeline attribute*), 133
problem_type (*evalml.pipelines.BaselineMulticlassPipeline attribute*), 137
problem_type (*evalml.pipelines.BaselineRegressionPipeline attribute*), 169
problem_type (*evalml.pipelines.CatBoostBinaryClassificationPipeline attribute*), 90
problem_type (*evalml.pipelines.CatBoostMulticlassClassificationPipeline attribute*), 93
problem_type (*evalml.pipelines.CatBoostRegressionPipeline attribute*), 152
problem_type (*evalml.pipelines.ENBinaryPipeline attribute*), 97
problem_type (*evalml.pipelines.ENMulticlassPipeline attribute*), 100
problem_type (*evalml.pipelines.ENRegressionPipeline attribute*), 155
problem_type (*evalml.pipelines.ETBinaryClassificationPipeline attribute*), 104
problem_type (*evalml.pipelines.ETMulticlassClassificationPipeline attribute*), 108
problem_type (*evalml.pipelines.ETRegressionPipeline attribute*), 159
problem_type (*evalml.pipelines.LinearRegressionPipeline attribute*), 162
problem_type (*evalml.pipelines.LogisticRegressionBinaryPipeline attribute*), 111
problem_type (*evalml.pipelines.LogisticRegressionMulticlassPipeline attribute*), 115
problem_type (*evalml.pipelines.MeanBaselineRegressionPipeline attribute*), 173
problem_type (*evalml.pipelines.ModeBaselineBinaryPipeline attribute*), 141
problem_type (*evalml.pipelines.ModeBaselineMulticlassPipeline attribute*), 144
problem_type (*evalml.pipelines.RFBinaryClassificationPipeline attribute*), 119
problem_type (*evalml.pipelines.RFMulticlassClassificationPipeline attribute*), 122
problem_type (*evalml.pipelines.RFRegressionPipeline attribute*), 148
problem_type (*evalml.pipelines.XGBoostBinaryPipeline attribute*), 126
problem_type (*evalml.pipelines.XGBoostMulticlassPipeline attribute*), 130
problem_type (*evalml.pipelines.XGBoostRegressionPipeline attribute*), 166
ProblemTypes (*class in evalml.problem_types*), 291
propose() (*evalml.tuners.GridSearchTuner method*), 296
propose() (*evalml.tuners.RandomSearchTuner method*), 298
propose() (*evalml.tuners.SKOptTuner method*), 294
propose() (*evalml.tuners.Tuner method*), 293

R

R2 (*class in evalml.objectives*), 280
RandomForestClassifier (*class in evalml.pipelines.components*), 212
RandomForestRegressor (*class in evalml.pipelines.components*), 228
RandomSearchTuner (*class in evalml.tuners*), 296
Recall (*class in evalml.objectives*), 274
RecallMacro (*class in evalml.objectives*), 277
RecallMicro (*class in evalml.objectives*), 276
RecallWeighted (*class in evalml.objectives*), 278
RegressionObjective (*class in evalml.objectives*), 239
RegressionPipeline (*class in evalml.pipelines*), 86
RFBinaryClassificationPipeline (*class in evalml.pipelines*), 119
RFClassifierSelectFromModel (*class in evalml.pipelines.components*), 203
RFMulticlassClassificationPipeline (*class in evalml.pipelines*), 122
RFRegressionPipeline (*class in evalml.pipelines*), 148
RFRegressorSelectFromModel (*class in evalml.pipelines.components*), 200
roc_curve() (*in module evalml.pipelines*), 179
RootMeanSquaredError (*class in evalml.objectives*), 288
RootMeanSquaredLogError (*class in evalml.objectives*), 289

S

save() (*evalml.automl.AutoMLSearch method*), 68
save() (*evalml.pipelines.BaselineBinaryPipeline method*), 136
save() (*evalml.pipelines.BaselineMulticlassPipeline method*), 140
save() (*evalml.pipelines.BaselineRegressionPipeline method*), 172
save() (*evalml.pipelines.BinaryClassificationPipeline method*), 82
save() (*evalml.pipelines.CatBoostBinaryClassificationPipeline method*), 93
save() (*evalml.pipelines.CatBoostMulticlassClassificationPipeline method*), 96
save() (*evalml.pipelines.CatBoostRegressionPipeline method*), 155
save() (*evalml.pipelines.ClassificationPipeline method*), 79
save() (*evalml.pipelines.ENBinaryPipeline method*), 100
save() (*evalml.pipelines.ENMulticlassPipeline method*), 103
save() (*evalml.pipelines.ENRegressionPipeline method*), 158
save() (*evalml.pipelines.ETBinaryClassificationPipeline method*), 107
save() (*evalml.pipelines.ETMulticlassClassificationPipeline method*), 111
save() (*evalml.pipelines.ETRegressionPipeline method*), 162
save() (*evalml.pipelines.LinearRegressionPipeline method*), 165
save() (*evalml.pipelines.LogisticRegressionBinaryPipeline method*), 114
save() (*evalml.pipelines.LogisticRegressionMulticlassPipeline method*), 118
save() (*evalml.pipelines.MeanBaselineRegressionPipeline method*), 176
save() (*evalml.pipelines.ModeBaselineBinaryPipeline method*), 144
save() (*evalml.pipelines.ModeBaselineMulticlassPipeline method*), 147
save() (*evalml.pipelines.MulticlassClassificationPipeline method*), 86
save() (*evalml.pipelines.PipelineBase method*), 76
save() (*evalml.pipelines.RegressionPipeline method*), 89
save() (*evalml.pipelines.RFBinaryClassificationPipeline method*), 122
save() (*evalml.pipelines.RFMulticlassClassificationPipeline method*), 125
save() (*evalml.pipelines.RFRegressionPipeline method*), 151
save() (*evalml.pipelines.XGBoostBinaryPipeline method*), 129
save() (*evalml.pipelines.XGBoostMulticlassPipeline method*), 133
save() (*evalml.pipelines.XGBoostRegressionPipeline method*), 169
score() (*evalml.objectives.AccuracyBinary method*), 247
score() (*evalml.objectives.AccuracyMulticlass method*), 248
score() (*evalml.objectives.AUC method*), 250
score() (*evalml.objectives.AUCMacro method*), 251
score() (*evalml.objectives.AUCMicro method*), 252
score() (*evalml.objectives.AUCWeighted method*), 254
score() (*evalml.objectives.BalancedAccuracyBinary method*), 255
score() (*evalml.objectives.BalancedAccuracyMulticlass method*), 257
score() (*evalml.objectives.BinaryClassificationObjective method*), 237
score() (*evalml.objectives.ExpVariance method*), 287
score() (*evalml.objectives.F1 method*), 258
score() (*evalml.objectives.F1Macro method*), 261
score() (*evalml.objectives.F1Micro method*), 260
score() (*evalml.objectives.F1Weighted method*), 262
score() (*evalml.objectives.FraudCost method*), 242
score() (*evalml.objectives.LeadScoring method*), 244
score() (*evalml.objectives.LogLossBinary method*), 264
score() (*evalml.objectives.LogLossMulticlass method*), 265
score() (*evalml.objectives.MAE method*), 281
score() (*evalml.objectives.MaxError method*), 286
score() (*evalml.objectives.MCCBinary method*), 267
score() (*evalml.objectives.MCCMulticlass method*), 268
score() (*evalml.objectives.MeanSquaredLogError method*), 284
score() (*evalml.objectives.MedianAE method*), 285
score() (*evalml.objectives.MSE method*), 283
score() (*evalml.objectives.MulticlassClassificationObjective method*), 239
score() (*evalml.objectives.ObjectiveBase method*), 235
score() (*evalml.objectives.Precision method*), 270
score() (*evalml.objectives.PrecisionMacro method*), 272
score() (*evalml.objectives.PrecisionMicro method*), 271
score() (*evalml.objectives.PrecisionWeighted method*), 273
score() (*evalml.objectives.R2 method*), 280
score() (*evalml.objectives.Recall method*), 275
score() (*evalml.objectives.RecallMacro method*), 278
score() (*evalml.objectives.RecallMicro method*), 276

score() (*evalml.objectives.RecallWeighted method*), 279
score() (*evalml.objectives.RegressionObjective method*), 240
score() (*evalml.objectives.RootMeanSquaredError method*), 289
score() (*evalml.objectives.RootMeanSquaredLogError method*), 290
score() (*evalml.pipelines.BaselineBinaryPipeline method*), 137
score() (*evalml.pipelines.BaselineMulticlassPipeline method*), 140
score() (*evalml.pipelines.BaselineRegressionPipeline method*), 172
score() (*evalml.pipelines.BinaryClassificationPipeline method*), 83
score() (*evalml.pipelines.CatBoostBinaryClassificationPipeline method*), 93
score() (*evalml.pipelines.CatBoostMulticlassClassificationPipeline method*), 96
score() (*evalml.pipelines.CatBoostRegressionPipeline method*), 155
score() (*evalml.pipelines.ClassificationPipeline method*), 79
score() (*evalml.pipelines.ENBinaryPipeline method*), 100
score() (*evalml.pipelines.ENMulticlassPipeline method*), 104
score() (*evalml.pipelines.ENRegressionPipeline method*), 158
score() (*evalml.pipelines.ETBinaryClassificationPipeline method*), 107
score() (*evalml.pipelines.ETMulticlassClassificationPipeline method*), 111
score() (*evalml.pipelines.ETRegressionPipeline method*), 162
score() (*evalml.pipelines.LinearRegressionPipeline method*), 165
score() (*evalml.pipelines.LogisticRegressionBinaryPipeline method*), 115
score() (*evalml.pipelines.LogisticRegressionMulticlassPipeline method*), 118
score() (*evalml.pipelines.MeanBaselineRegressionPipeline method*), 176
score() (*evalml.pipelines.ModeBaselineBinaryPipeline method*), 144
score() (*evalml.pipelines.ModeBaselineMulticlassPipeline method*), 148
score() (*evalml.pipelines.MulticlassClassificationPipeline method*), 86
score() (*evalml.pipelines.PipelineBase method*), 76
score() (*evalml.pipelines.RegressionPipeline method*), 89
score() (*evalml.pipelines.RFBinaryClassificationPipeline method*), 122
score() (*evalml.pipelines.RFMulticlassClassificationPipeline method*), 126
score() (*evalml.pipelines.RFRegressionPipeline method*), 151
score() (*evalml.pipelines.XGBoostBinaryPipeline method*), 129
score() (*evalml.pipelines.XGBoostMulticlassPipeline method*), 133
score() (*evalml.pipelines.XGBoostRegressionPipeline method*), 169
search() (*evalml.automl.AutoMLSearch method*), 69
SelectColumns (*class in evalml.pipelines.components*), 189
SimpleImputer (*class in evalml.pipelines.components*), 196
SKOptTuner (*class in evalml.tuners*), 293
split_data() (*in module evalml.preprocessing*), 65
StandardScaler (*class in evalml.pipelines.components*), 198
summary (*evalml.pipelines.BaselineBinaryPipeline attribute*), 133
summary (*evalml.pipelines.BaselineMulticlassPipeline attribute*), 137
summary (*evalml.pipelines.BaselineRegressionPipeline attribute*), 169
summary (*evalml.pipelines.CatBoostBinaryClassificationPipeline attribute*), 90
summary (*evalml.pipelines.CatBoostMulticlassClassificationPipeline attribute*), 93
summary (*evalml.pipelines.CatBoostRegressionPipeline attribute*), 152
summary (*evalml.pipelines.ENBinaryPipeline attribute*), 97
summary (*evalml.pipelines.ENMulticlassPipeline attribute*), 100
summary (*evalml.pipelines.ENRegressionPipeline attribute*), 155
summary (*evalml.pipelines.ETBinaryClassificationPipeline attribute*), 104
summary (*evalml.pipelines.ETMulticlassClassificationPipeline attribute*), 108
summary (*evalml.pipelines.ETRegressionPipeline attribute*), 159
summary (*evalml.pipelines.LinearRegressionPipeline attribute*), 162
summary (*evalml.pipelines.LogisticRegressionBinaryPipeline attribute*), 111
summary (*evalml.pipelines.LogisticRegressionMulticlassPipeline attribute*), 115
summary (*evalml.pipelines.MeanBaselineRegressionPipeline attribute*), 173
summary (*evalml.pipelines.ModeBaselineBinaryPipeline attribute*), 141

summary (*evalml.pipelines.ModeBaselineMulticlassPipeline attribute*), 144
summary (*evalml.pipelines.RFBinaryClassificationPipeline attribute*), 119
summary (*evalml.pipelines.RFMulticlassClassificationPipeline attribute*), 122
summary (*evalml.pipelines.RFRegressionPipeline attribute*), 148
summary (*evalml.pipelines.XGBoostBinaryPipeline attribute*), 126
summary (*evalml.pipelines.XGBoostMulticlassPipeline attribute*), 130
summary (*evalml.pipelines.XGBoostRegressionPipeline attribute*), 166
supported_problem_types (*evalml.pipelines.components.BaselineClassifier attribute*), 218
supported_problem_types (*evalml.pipelines.components.BaselineRegressor attribute*), 232
supported_problem_types (*evalml.pipelines.components.CatBoostClassifier attribute*), 206
supported_problem_types (*evalml.pipelines.components.CatBoostRegressor attribute*), 220
supported_problem_types (*evalml.pipelines.components.ElasticNetClassifier attribute*), 208
supported_problem_types (*evalml.pipelines.components.ElasticNetRegressor attribute*), 222
supported_problem_types (*evalml.pipelines.components.ExtraTreesClassifier attribute*), 210
supported_problem_types (*evalml.pipelines.components.ExtraTreesRegressor attribute*), 226
supported_problem_types (*evalml.pipelines.components.LinearRegressor attribute*), 224
supported_problem_types (*evalml.pipelines.components.LogisticRegressionClassifier attribute*), 214
supported_problem_types (*evalml.pipelines.components.RandomForestClassifier attribute*), 212
supported_problem_types (*evalml.pipelines.components.RandomForestRegressor attribute*), 228
supported_problem_types (*evalml.pipelines.components.XGBoostClassifier attribute*), 216
supported_problem_types (*evalml.pipelines.components.XGBoostRegressor attribute*), 230

T

transform() (*evalml.pipelines.components.DropColumns method*), 189
transform() (*evalml.pipelines.components.OneHotEncoder method*), 193
transform() (*evalml.pipelines.components.PerColumnImputer method*), 196
transform() (*evalml.pipelines.components.RFClassifierSelectFromMod method*), 205
transform() (*evalml.pipelines.components.RFRegressorSelectFromMod method*), 202
transform() (*evalml.pipelines.components.SelectColumns method*), 191
transform() (*evalml.pipelines.components.SimpleImputer method*), 198
transform() (*evalml.pipelines.components.StandardScaler method*), 200
transform() (*evalml.pipelines.components.Transformer method*), 185
Transformer (*class in evalml.pipelines.components*), 183
Tuner (*class in evalml.tuners*), 292

V

validate() (*evalml.data_checks.DataCheck method*), 299
validate() (*evalml.data_checks.DataChecks method*), 306
validate() (*evalml.data_checks.DefaultDataChecks method*), 306
validate() (*evalml.data_checks.HighlyNullDataCheck method*), 301
validate() (*evalml.data_checks.IDColumnsDataCheck method*), 302
validate() (*evalml.data_checks.InvalidTargetDataCheck method*), 300
validate() (*evalml.data_checks.LabelLeakageDataCheck method*), 303
validate() (*evalml.data_checks.OutliersDataCheck method*), 304
validate_inputs() (*evalml.objectives.AccuracyBinary method*), 247
validate_inputs() (*evalml.objectives.AccuracyMulticlass method*), 249
validate_inputs() (*evalml.objectives.AUC method*), 250
validate_inputs() (*evalml.objectives.AUCMacro method*), 252

validate_inputs() (*evalml.objectives.AUCMicro method*), 253
validate_inputs() (*evalml.objectives.AUCWeighted method*), 254
validate_inputs() (*evalml.objectives.BalancedAccuracyBinary method*), 256
validate_inputs() (*evalml.objectives.BalancedAccuracyMulticlass method*), 257
validate_inputs() (*evalml.objectives.BinaryClassificationObjective method*), 238
validate_inputs() (*evalml.objectives.ExpVariance method*), 288
validate_inputs() (*evalml.objectives.F1 method*), 259
validate_inputs() (*evalml.objectives.F1Macro method*), 261
validate_inputs() (*evalml.objectives.F1Micro method*), 260
validate_inputs() (*evalml.objectives.F1Weighted method*), 262
validate_inputs() (*evalml.objectives.FraudCost method*), 243
validate_inputs() (*evalml.objectives.LeadScoring method*), 245
validate_inputs() (*evalml.objectives.LogLossBinary method*), 264
validate_inputs() (*evalml.objectives.LogLossMulticlass method*), 265
validate_inputs() (*evalml.objectives.MAE method*), 282
validate_inputs() (*evalml.objectives.MaxError method*), 287
validate_inputs() (*evalml.objectives.MCCBinary method*), 267
validate_inputs() (*evalml.objectives.MCCMulticlass method*), 268
validate_inputs() (*evalml.objectives.MeanSquaredLogError method*), 284
validate_inputs() (*evalml.objectives.MedianAE method*), 285
validate_inputs() (*evalml.objectives.MSE method*), 283
validate_inputs() (*evalml.objectives.MulticlassClassificationObjective method*), 239
validate_inputs() (*evalml.objectives.ObjectiveBase method*), 236
validate_inputs() (*evalml.objectives.Precision method*), 270
validate_inputs() (*evalml.objectives.PrecisionMacro method*), 273
validate_inputs() (*evalml.objectives.PrecisionMicro method*), 271
validate_inputs() (*evalml.objectives.PrecisionWeighted method*), 274
validate_inputs() (*evalml.objectives.R2 method*), 281
validate_inputs() (*evalml.objectives.Recall method*), 276
validate_inputs() (*evalml.objectives.RecallMacro method*), 278
validate_inputs() (*evalml.objectives.RecallMicro method*), 277
validate_inputs() (*evalml.objectives.RecallWeighted method*), 279
validate_inputs() (*evalml.objectives.RegressionObjective method*), 240
validate_inputs() (*evalml.objectives.RootMeanSquaredError method*), 289
validate_inputs() (*evalml.objectives.RootMeanSquaredLogError method*), 290

X

XGBoostBinaryPipeline (*class in evalml.pipelines*), 126
XGBoostClassifier (*class in evalml.pipelines.components*), 216
XGBoostMulticlassPipeline (*class in evalml.pipelines*), 130
XGBoostRegressionPipeline (*class in evalml.pipelines*), 166
XGBoostRegressor (*class in evalml.pipelines.components*), 230