# EvalML Documentation

***Release 0.7.0***

**Feature Labs, Inc.**

**Mar 11, 2020**

# GETTING STARTED

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

# INSTALL

EvalML is available for Python 3.5+. It can be installed by running the following command:

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

**Note for Windows users**: 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.

CHAPTER

TWO

# QUICK START

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

## 2.1 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()
```

## 2.2 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 = AutoClassificationSearch(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)
```

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

Optimizing for F1. Greater score is better.

Searching up to 5 pipelines.
Possible model types: linear_model, catboost, xgboost, random_forest
```

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

```
 CatBoost Classifier w/ Simple Imput...    20%|        | Elapsed:00:02
 CatBoost Classifier w/ Simple Imput...    40%|      | Elapsed:00:17
 Random Forest Classifier w/ One Hot...    60%|    | Elapsed:00:27
 Logistic Regression Classifier w/ O...    80%|  | Elapsed:00:28
 Logistic Regression Classifier w/ O...   100%|| Elapsed:00:28
 Optimization finished                    100%|| Elapsed:00:28
```

## 2.3 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 *guardrails* to analyze your data to assist you in producing the best performing pipeline.

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

[6]:
```
   id             pipeline_class_name     score  high_variance_cv  \
0   1  CatBoostClassificationPipeline  0.972289             False
1   4      LogisticRegressionPipeline  0.970398             False
2   3      LogisticRegressionPipeline  0.968758             False
3   0  CatBoostClassificationPipeline  0.961876             False
4   2        RFClassificationPipeline  0.959823             False

                                      parameters
0  {'impute_strategy': 'most_frequent', 'n_estima...
1  {'penalty': 'l2', 'C': 6.239401330891865, 'imp...
2  {'penalty': 'l2', 'C': 8.444214828324364, 'imp...
3  {'impute_strategy': 'most_frequent', 'n_estima...
4  {'n_estimators': 569, 'max_depth': 22, 'impute...
```

## 2.4 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)
```

```
*********************************************************************************
* Logistic Regression Classifier w/ One Hot Encoder + Simple Imputer + Standard␣
↪Scaler *
```

(continues on next page)

(continued from previous page)

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

Problem Types: Binary Classification, Multiclass Classification
Model Type: Linear Model
Objective to Optimize: F1 (greater is better)
Number of features: 30

Pipeline Steps
==============
1. One Hot Encoder
2. Simple Imputer
         * impute_strategy : most_frequent
3. Standard Scaler
4. Logistic Regression Classifier
         * penalty : l2
         * C : 8.444214828324364

Training
========
Training for Binary Classification problems.
Total training time (including CV): 1.3 seconds

Cross Validation
----------
                F1  Precision  Recall   AUC  Log Loss   MCC # Training # Testing
0            0.974      0.979   0.968 0.997     0.082 0.930    303.000   152.000
1            0.959      0.931   0.989 0.985     0.214 0.889    303.000   152.000
2            0.974      0.979   0.968 0.984     0.158 0.929    304.000   151.000
mean         0.969      0.963   0.975 0.989     0.151 0.916          -         -
std          0.008      0.028   0.012 0.007     0.067 0.024          -         -
coef of var  0.009      0.029   0.012 0.007     0.440 0.026          -         -
```

## 2.5 Select Best pipeline

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

[8]:
```
pipeline = automl.best_pipeline
pipeline.score(X_holdout, y_holdout)
```

[8]:
```
(0.951048951048951, {})
```

We can also visualize the structure of our pipeline:

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

[9]:

## 2.6 Whats next?

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

### 2.6.1 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 the 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*.

### 2.6.2 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 AutoClassificationSearch
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
customer_present       bool
lat                 float64
lng                 float64
dtype: object
```

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

[5]:
```
automl = AutoClassificationSearch(objective=fraud_objective,
                                  additional_objectives=['auc', 'recall', 'precision
↪'],
                                  max_pipelines=5)

automl.search(X_train, y_train)
```

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

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

(continues on next page)

(continued from previous page)

```
Searching up to 5 pipelines.
Possible model types: random_forest, linear_model, catboost, xgboost
```

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

```
 CatBoost Classifier w/ Simple Imput...    20%|        | Elapsed:00:02
 CatBoost Classifier w/ Simple Imput...    40%|      | Elapsed:00:08
 Random Forest Classifier w/ One Hot...    60%|    | Elapsed:00:26
 Logistic Regression Classifier w/ O...    80%|  | Elapsed:00:29
 Logistic Regression Classifier w/ O...   100%|| Elapsed:00:31
 Optimization finished                    100%|| Elapsed:00:31
```

### 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_class_name     score  high_variance_cv  \
0   3      LogisticRegressionPipeline  0.007960             False
1   2        RFClassificationPipeline  0.008168             False
2   4      LogisticRegressionPipeline  0.008179             False
3   0  CatBoostClassificationPipeline  0.008512             False
4   1  CatBoostClassificationPipeline  0.009529             False

                                      parameters
0  {'penalty': 'l2', 'C': 8.444214828324364, 'imp...
1  {'n_estimators': 569, 'max_depth': 22, 'impute...
2  {'penalty': 'l2', 'C': 6.239401330891865, 'imp...
3  {'impute_strategy': 'most_frequent', 'n_estima...
4  {'impute_strategy': 'most_frequent', 'n_estima...
```

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"])
```

```
*****************************************************************************
* Logistic Regression Classifier w/ One Hot Encoder + Simple Imputer + Standard␣
↪Scaler *
*****************************************************************************

Problem Types: Binary Classification, Multiclass Classification
Model Type: Linear Model
```

(continues on next page)

(continued from previous page)

```
Objective to Optimize: Fraud Cost (lower is better)
Number of features: 170

Pipeline Steps
==============
1. One Hot Encoder
2. Simple Imputer
         * impute_strategy : most_frequent
3. Standard Scaler
4. Logistic Regression Classifier
         * penalty : l2
         * C : 8.444214828324364

Training
========
Training for Binary Classification problems.
Total training time (including CV): 3.4 seconds

Cross Validation
----------
             Fraud Cost    AUC  Recall  Precision # Training # Testing
0                 0.008 0.664   0.979      0.153   1333.000   667.000
1                 0.008 0.665   0.979      0.142   1333.000   667.000
2                 0.008 0.612   1.000      0.144   1334.000   666.000
mean              0.008 0.647   0.986      0.146          -         -
std               0.000 0.030   0.012      0.006          -         -
coef of var       0.012 0.047   0.012      0.039          -         -
```

### 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.logistic_regression.LogisticRegressionPipeline at␣
↪0x7fc7b3569048>
```

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, other_objectives=["auc", fraud_objective])
```

[10]:
```
(0.007745336400937289,
 OrderedDict([('AUC', 0.7252159468438538),
              ('Fraud Cost', 0.007745336400937289)]))
```

### 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 = AutoClassificationSearch(objective='auc',
                                      additional_objectives=['recall', 'precision'],
                                      max_pipelines=5)
```

(continues on next page)

(continued from previous page)

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

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

Optimizing for AUC. Greater score is better.

Searching up to 5 pipelines.
Possible model types: random_forest, linear_model, catboost, xgboost
```

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

```
 CatBoost Classifier w/ Simple Imput...    20%|        | Elapsed:00:01
 CatBoost Classifier w/ Simple Imput...    40%|      | Elapsed:00:08
 Random Forest Classifier w/ One Hot...    60%|    | Elapsed:00:24
 Logistic Regression Classifier w/ O...    80%|  | Elapsed:00:25
 Logistic Regression Classifier w/ O...   100%|| Elapsed:00:26
 Optimization finished                    100%|| Elapsed:00:26
```

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

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

[12]:
```
   id             pipeline_class_name     score  high_variance_cv  \
0   2        RFClassificationPipeline  0.860800             False
1   0  CatBoostClassificationPipeline  0.842237             False
2   1  CatBoostClassificationPipeline  0.827765             False
3   4      LogisticRegressionPipeline  0.648769             False
4   3      LogisticRegressionPipeline  0.647251             False

                                      parameters
0  {'n_estimators': 569, 'max_depth': 22, 'impute...
1  {'impute_strategy': 'most_frequent', 'n_estima...
2  {'impute_strategy': 'most_frequent', 'n_estima...
3  {'penalty': 'l2', 'C': 6.239401330891865, 'imp...
4  {'penalty': 'l2', 'C': 8.444214828324364, 'imp...
```

[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.classification.random_forest.RFClassificationPipeline at␣
↪0x7fc7b319bac8>
```

[14]:
```
# get the fraud score on holdout data
best_pipeline_auc.score(X_holdout, y_holdout,  other_objectives=["auc", fraud_
↪objective])
```

[14]:
```
(0.8354983388704318,
 OrderedDict([('AUC', 0.8354983388704318),
              ('Fraud Cost', 0.03655681280302016)]))
```

[15]:
```
# fraud score on fraud optimized again
best_pipeline.score(X_holdout, y_holdout, other_objectives=["auc", fraud_objective])
```

[15]:
```
(0.007745336400937289,
 OrderedDict([('AUC', 0.7252159468438538),
              ('Fraud Cost', 0.007745336400937289)]))
```

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.

### 2.6.3 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 AutoClassificationSearch
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]:
```
import pandas as pd

customers = pd.read_csv('s3://featurelabs-static/lead_scoring_ml_apps/customers.csv')
interactions = pd.read_csv('s3://featurelabs-static/lead_scoring_ml_apps/interactions.
↪csv')
```

(continues on next page)

(continued from previous page)

```
leads = pd.read_csv('s3://featurelabs-static/lead_scoring_ml_apps/previous_leads.csv')

X = customers.merge(interactions, on='customer_id').merge(leads, on='customer_id')
y = X['label']

X = X.drop(['customer_id', 'date_registered', 'birthday','phone', 'email',
        'owner', 'company', 'id', 'time_x',
        'session', 'referrer', 'time_y', 'label'], 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 `AutoClassificationSearch`. When we call `.search()`, the search for the best pipeline will begin.

[5]:
```
automl = AutoClassificationSearch(objective=lead_scoring_objective,
                                  additional_objectives=['auc'],
                                  max_pipelines=5)

automl.search(X_train, y_train)
```

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

Optimizing for Lead Scoring. Greater score is better.

Searching up to 5 pipelines.
Possible model types: random_forest, xgboost, linear_model, catboost
```

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

```
 CatBoost Classifier w/ Simple Imput...    20%|        | Elapsed:00:03
 CatBoost Classifier w/ Simple Imput...    40%|      | Elapsed:00:13
 Random Forest Classifier w/ One Hot...    60%|    | Elapsed:00:33
 Logistic Regression Classifier w/ O...    80%|  | Elapsed:00:39
 Logistic Regression Classifier w/ O...   100%|| Elapsed:00:44
 Optimization finished                    100%|| Elapsed:00:44
```

### 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_class_name      score  high_variance_cv  \
0   2        RFClassificationPipeline  14.242075             False
1   3      LogisticRegressionPipeline  12.654899              True
2   4      LogisticRegressionPipeline  12.652749              True
3   0  CatBoostClassificationPipeline  11.869532              True
4   1  CatBoostClassificationPipeline   9.868867              True

                                      parameters
0  {'n_estimators': 569, 'max_depth': 22, 'impute...
1  {'penalty': 'l2', 'C': 8.444214828324364, 'imp...
2  {'penalty': 'l2', 'C': 6.239401330891865, 'imp...
3  {'impute_strategy': 'most_frequent', 'n_estima...
4  {'impute_strategy': 'most_frequent', 'n_estima...
```

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"])
```

```
*****************************************************************************************
* Random Forest Classifier w/ One Hot Encoder + Simple Imputer + RF Classifier Select␣
↪From Model *
*****************************************************************************************

Problem Types: Binary Classification, Multiclass Classification
Model Type: Random Forest
Objective to Optimize: Lead Scoring (greater is better)
Number of features: 5

Pipeline Steps
==============
```

(continues on next page)

(continued from previous page)

```
1. One Hot Encoder
2. Simple Imputer
         * impute_strategy : most_frequent
3. RF Classifier Select From Model
         * percent_features : 0.8593661614465293
         * threshold : -inf
4. Random Forest Classifier
         * n_estimators : 569
         * max_depth : 22

Training
========
Training for Binary Classification problems.
Total training time (including CV): 20.1 seconds

Cross Validation
----------
             Lead Scoring   AUC # Training # Testing
0                  11.477 0.587   3099.000  1550.000
1                  15.600 0.527   3099.000  1550.000
2                  15.649 0.601   3100.000  1549.000
mean               14.242 0.572          -         -
std                 2.394 0.039          -         -
coef of var         0.168 0.069          -         -
```

### 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.random_forest.RFClassificationPipeline at␣
↪0x7f1c438e9fd0>
```

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, other_objectives=["auc", lead_scoring_
↪objective])
```

[10]:
```
(10.60189165950129,
 OrderedDict([('AUC', 0.5471365971592625),
              ('Lead Scoring', 10.60189165950129)]))
```

### 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.AutoClassificationSearch(objective='auc',
                                 additional_objectives=[],
                                 max_pipelines=5)

automl_auc.search(X_train, y_train)
```

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

Optimizing for AUC. Greater score is better.

Searching up to 5 pipelines.
Possible model types: random_forest, xgboost, linear_model, catboost
```

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

```
 CatBoost Classifier w/ Simple Imput...    20%|        | Elapsed:00:02
 CatBoost Classifier w/ Simple Imput...    40%|      | Elapsed:00:13
 Random Forest Classifier w/ One Hot...    60%|    | Elapsed:00:30
 Logistic Regression Classifier w/ O...    80%|  | Elapsed:00:33
 Logistic Regression Classifier w/ O...   100%|| Elapsed:00:37
 Optimization finished                    100%|| Elapsed:00:37
```

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

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

[12]:
```
   id             pipeline_class_name     score  high_variance_cv  \
0   3      LogisticRegressionPipeline  0.926479             False
1   4      LogisticRegressionPipeline  0.926282             False
2   0  CatBoostClassificationPipeline  0.915464             False
3   1  CatBoostClassificationPipeline  0.885380             False
4   2        RFClassificationPipeline  0.569268             False

                                        parameters
0  {'penalty': 'l2', 'C': 8.444214828324364, 'imp...
1  {'penalty': 'l2', 'C': 6.239401330891865, 'imp...
2  {'impute_strategy': 'most_frequent', 'n_estima...
3  {'impute_strategy': 'most_frequent', 'n_estima...
4  {'n_estimators': 569, 'max_depth': 22, 'impute...
```

[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.classification.logistic_regression.LogisticRegressionPipeline at␣
↪0x7f1c434912b0>
```

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

[14]:
```
(0.9272061045633122,
 OrderedDict([('AUC', 0.9272061045633122),
              ('Lead Scoring', -0.017196904557179708)]))
```

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.

### 2.6.4 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.objective_base import ObjectiveBase

class FraudCost(ObjectiveBase):
    """Score the percentage of money lost of the total transaction amount process due␣
↪to fraud"""
    name = "Fraud Cost"
    needs_fitting = True
    greater_is_better = False
    uses_extra_columns = True
    score_needs_proba = False

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

        Args:
            retry_percentage (float): what percentage of customers will retry a␣
↪transaction if it
                is declined? Between 0 and 1. Defaults to .5

            interchange_fee (float): how much of each successful transaction do you␣
↪collect?
                Between 0 and 1. Defaults to .02

            fraud_payout_percentage (float):  how percentage of fraud will you be␣
↪unable to collect.
```

(continues on next page)

(continued from previous page)

```
                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
        super().__init__(verbose=verbose)

    def decision_function(self, y_predicted, extra_cols, threshold):
        """Determine if transaction is fraud given predicted probabilities,
            dataframe with transaction amount, and threshold"""

        transformed_probs = (y_predicted * extra_cols[self.amount_col])
        return transformed_probs > threshold

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

        # extract transaction using the amount columns in users data
        transaction_amount = extra_cols[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
```

### 2.6.5 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 importing the appropriate class.

[1]:
```
import evalml
from evalml import AutoClassificationSearch, AutoRegressionSearch
```

[2]:
```
AutoClassificationSearch()
```

[2]:
```
<evalml.automl.auto_classification_search.AutoClassificationSearch at 0x7f9499395358>
```

[3]:
```
AutoRegressionSearch()
```

[3]:
```
<evalml.automl.auto_regression_search.AutoRegressionSearch at 0x7f949936b470>
```

### 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

[4]:
```
from evalml.objectives import FraudCost

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

AutoClassificationSearch(objective=fraud_objective)
```

[4]:
```
<evalml.automl.auto_classification_search.AutoClassificationSearch at 0x7f9499373358>
```

### Evaluate on Additional Objectives

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

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

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

AutoClassificationSearch(objective='AUC', additional_objectives=[fraud_objective])
```

[5]:
```
<evalml.automl.auto_classification_search.AutoClassificationSearch at 0x7f94992fb5c0>
```

### Selecting Model Types

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

[6]:
```
automl = AutoClassificationSearch(objective="f1",
                                  model_types=["random_forest"])
```

you can see the possible pipelines that will be searched after initialization

[7]:
```
automl.possible_pipelines
```

[7]:
```
[evalml.pipelines.classification.random_forest.RFClassificationPipeline]
```

you can see a list of all supported models like this

[8]:
```
evalml.list_model_types("binary") # `binary` for binary classification and␣
↪`multiclass` for multiclass classification
```

[8]:
```
[<ModelTypes.RANDOM_FOREST: 'random_forest'>,
 <ModelTypes.CATBOOST: 'catboost'>,
 <ModelTypes.LINEAR_MODEL: 'linear_model'>,
 <ModelTypes.XGBOOST: 'xgboost'>]
```

[9]:
```
evalml.list_model_types("regression")
```

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

### 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.

[10]:
```
AutoClassificationSearch(objective="f1",
                         max_pipelines=5,
                         max_time=60)
```

(continues on next page)

(continued from previous page)

```
AutoClassificationSearch(objective="f1",
                         max_time="1 minute")
```

[10]:
```
<evalml.automl.auto_classification_search.AutoClassificationSearch at 0x7f949930ae80>
```

To start, EvalML samples 10 sets of hyperparameters chosen randomly for each possible pipeline. Therefore, we recommend setting `max_pipelines` at least 10 times the number of possible pipelines.

[11]:
```
n_possible_pipelines = len(AutoClassificationSearch(objective="f1").possible_
↪pipelines)
```

[12]:
```
AutoClassificationSearch(objective="f1",
                         max_time=60)
```

[12]:
```
<evalml.automl.auto_classification_search.AutoClassificationSearch at 0x7f949930af60>
```

### 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.

[13]:
```
from evalml.demos import load_diabetes

X, y = load_diabetes()
automl = AutoRegressionSearch(objective="MSE", patience=2, tolerance=0.01, max_
↪pipelines=10)
automl.search(X, y)
```

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

Optimizing for MSE. Lower score is better.

Searching up to 10 pipelines.
Possible model types: random_forest, catboost, linear_model
```

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

```
 Random Forest Regressor w/ One Hot ...    10%|          | Elapsed:00:10
 Random Forest Regressor w/ One Hot ...    20%|         | Elapsed:00:16
 Linear Regressor w/ One Hot Encoder...    30%|        | Elapsed:00:16
 Random Forest Regressor w/ One Hot ...    40%|       | Elapsed:00:26
 CatBoost Regressor w/ Simple Imputer:     50%|      | Elapsed:00:26


2 iterations without improvement. Stopping search early...
 Optimization finished                     50%|      | Elapsed:00:26
```

[14]:
```
automl.rankings
```

[14]:
```
   id         pipeline_class_name        score  high_variance_cv  \
0   2    LinearRegressionPipeline  3027.144520             False
1   0        RFRegressionPipeline  3413.567159             False
2   3        RFRegressionPipeline  3648.302977             False
3   1        RFRegressionPipeline  3656.887580             False
4   4  CatBoostRegressionPipeline  4436.269202             False

                                        parameters
0  {'impute_strategy': 'mean', 'normalize': True,...
1  {'n_estimators': 569, 'max_depth': 22, 'impute...
2  {'n_estimators': 609, 'max_depth': 7, 'impute_...
3  {'n_estimators': 369, 'max_depth': 10, 'impute...
4  {'impute_strategy': 'most_frequent', 'n_estima...
```

### 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.

[15]:
```
from sklearn.model_selection import StratifiedKFold

automl = AutoClassificationSearch(objective="f1",
                                  cv=StratifiedKFold(5))
```

## 2.6.6 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 AutoClassificationSearch

X, y = evalml.demos.load_breast_cancer()

automl = AutoClassificationSearch(objective="f1",
                                  max_pipelines=5)

automl.search(X, y)
```

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

Optimizing for F1. Greater score is better.

Searching up to 5 pipelines.
Possible model types: random_forest, xgboost, catboost, linear_model
```

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

```
CatBoost Classifier w/ Simple Imput...    20%|        | Elapsed:00:02
CatBoost Classifier w/ Simple Imput...    40%|      | Elapsed:00:16
Random Forest Classifier w/ One Hot...    60%|    | Elapsed:00:27
Logistic Regression Classifier w/ O...    80%|  | Elapsed:00:28
Logistic Regression Classifier w/ O...   100%|| Elapsed:00:28
Optimization finished                    100%|| Elapsed:00:28
```

### 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_class_name     score  high_variance_cv  \
0   0  CatBoostClassificationPipeline  0.979274             False
1   4      LogisticRegressionPipeline  0.976371             False
2   3      LogisticRegressionPipeline  0.974941             False
3   1  CatBoostClassificationPipeline  0.974830             False
4   2        RFClassificationPipeline  0.963874             False

                                       parameters
0  {'impute_strategy': 'most_frequent', 'n_estima...
1  {'penalty': 'l2', 'C': 6.239401330891865, 'imp...
2  {'penalty': 'l2', 'C': 8.444214828324364, 'imp...
3  {'impute_strategy': 'most_frequent', 'n_estima...
4  {'n_estimators': 569, 'max_depth': 22, 'impute...
```

### 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(0)
```

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

Problem Types: Binary Classification, Multiclass Classification
Model Type: CatBoost Classifier
Objective to Optimize: F1 (greater is better)
Number of features: 30

Pipeline Steps
==============
1. Simple Imputer
         * impute_strategy : most_frequent
2. CatBoost Classifier
         * n_estimators : 202
         * eta : 0.602763376071644
         * max_depth : 4

Training
========
```

(continues on next page)

(continued from previous page)

```
Training for Binary Classification problems.
Total training time (including CV): 2.6 seconds

Cross Validation
-----------
                F1  Precision  Recall   AUC  Log Loss   MCC # Training # Testing
0            0.979      0.975   0.983 0.983     0.156 0.944    379.000   190.000
1            0.975      0.952   1.000 0.995     0.118 0.934    379.000   190.000
2            0.983      0.975   0.992 0.995     0.085 0.955    380.000   189.000
mean         0.979      0.967   0.992 0.991     0.120 0.944          -         -
std          0.004      0.013   0.008 0.007     0.035 0.011          -         -
coef of var  0.004      0.014   0.008 0.007     0.293 0.011          -         -
```

### Get Pipeline

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

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

[4]:
```
<evalml.pipelines.classification.catboost.CatBoostClassificationPipeline at␣
↪0x7f7bf81f5fd0>
```

### Get best pipeline

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

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

[5]:
```
<evalml.pipelines.classification.catboost.CatBoostClassificationPipeline at␣
↪0x7f7bf81f5fd0>
```

### Feature Importances

We can get the feature importances of the resulting pipeline

[6]:
```
pipeline = automl.get_pipeline(0)
pipeline.feature_importances
```

[6]:
```
                    feature  importance
0              mean texture   11.352969
1          worst smoothness    8.196440
2       mean concave points    8.066988
3                 mean area    7.985677
4           worst perimeter    7.985116
5      worst concave points    6.362056
6                worst area    5.524540
7             worst texture    5.120554
8           perimeter error    4.753916
9           worst concavity    4.293226
10         mean compactness    3.599787
11               area error    3.043145
12             worst radius    2.995585
13     concave points error    2.714613
```

(continues on next page)

(continued from previous page)

```
14  fractal dimension error    2.428115
15            mean symmetry    2.194052
16   mean fractal dimension    2.026659
17           mean concavity    1.730882
18           symmetry error    1.514178
19        compactness error    1.326043
20         smoothness error    1.227851
21           worst symmetry    1.205117
22          mean smoothness    1.041237
23              mean radius    0.939787
24  worst fractal dimension    0.869844
25        worst compactness    0.527123
26           mean perimeter    0.349190
27            texture error    0.324993
28             radius error    0.223363
29          concavity error    0.076957
```

We can also create a bar plot of the feature importances

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

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

### Plot ROC

For binary classification tasks, we can also plot the ROC plot of a specific pipeline:

[8]:
```
automl.plot.generate_roc_plot(0)
```

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

[9]:
```
automl.results
```

[9]:
```
{'pipeline_results': {0: {'id': 0,
   'pipeline_class_name': 'CatBoostClassificationPipeline',
   'pipeline_name': 'CatBoost Classifier w/ Simple Imputer',
   'parameters': {'impute_strategy': 'most_frequent',
    'n_estimators': 202,
    'eta': 0.602763376071644,
    'max_depth': 4},
   'score': 0.979274222435619,
   'high_variance_cv': False,
   'training_time': 2.6439478397369385,
   'cv_data': [{'all_objective_scores': OrderedDict([('F1',
                  0.9790794979079498),
                 ('Precision', 0.975),
```

(continues on next page)

(continued from previous page)

```
                 ('Recall', 0.9831932773109243),
                 ('AUC', 0.9831932773109243),
                 ('Log Loss', 0.1556496891601824),
                 ('MCC', 0.9436801731761278),
                 ('ROC',
                  (array([0.        , 0.        , 0.        , 0.01408451, 0.01408451,
                          0.02816901, 0.02816901, 0.04225352, 0.04225352, 0.07042254,
                          0.07042254, 0.08450704, 0.08450704, 1.        ]),
                   array([0.        , 0.00840336, 0.17647059, 0.17647059, 0.73109244,
                          0.73109244, 0.94117647, 0.94117647, 0.98319328, 0.98319328,
                          0.99159664, 0.99159664, 1.        , 1.        ]),
                   array([1.99999959e+00, 9.99999592e-01, 9.99993074e-01, 9.
↪99992019e-01,
                          9.99465386e-01, 9.99344491e-01, 8.38501415e-01, 7.
↪93190872e-01,
                          5.97384979e-01, 2.01626440e-01, 9.86793713e-02, 8.
↪60714520e-02,
                          4.44265520e-02, 1.79769175e-06]))),
                 ('Confusion Matrix',
                      0    1
                  0  68    3
                  1   2  117),
                 ('# Training', 379),
                 ('# Testing', 190)]),
     'score': 0.9790794979079498},
    {'all_objective_scores': OrderedDict([('F1', 0.9754098360655737),
                 ('Precision', 0.952),
                 ('Recall', 1.0),
                 ('AUC', 0.9946739259083915),
                 ('Log Loss', 0.11838678188748847),
                 ('MCC', 0.933568045604951),
                 ('ROC',
                  (array([0.        , 0.        , 0.        , 0.01408451, 0.01408451,
                          0.02816901, 0.02816901, 0.04225352, 0.04225352, 0.07042254,
                          0.07042254, 1.        ]),
                   array([0.        , 0.00840336, 0.72268908, 0.72268908, 0.95798319,
                          0.95798319, 0.97478992, 0.97478992, 0.98319328, 0.98319328,
                          1.        , 1.        ]),
                   array([1.99999873e+00, 9.99998731e-01, 9.99727090e-01, 9.
↪99712236e-01,
                          9.76909119e-01, 9.72407651e-01, 9.39800583e-01, 9.
↪06770293e-01,
                          8.95490079e-01, 8.87456065e-01, 6.89141765e-01, 5.
↪52202128e-07]))),
                 ('Confusion Matrix',
                      0    1
                  0  65    6
                  1   0  119),
                 ('# Training', 379),
                 ('# Testing', 190)]),
     'score': 0.9754098360655737},
    {'all_objective_scores': OrderedDict([('F1', 0.9833333333333334),
                 ('Precision', 0.9752066115702479),
                 ('Recall', 0.9915966386554622),
                 ('AUC', 0.9949579831932773),
                 ('Log Loss', 0.0853788718666447),
                 ('MCC', 0.9546019995535027),
```

(continues on next page)

(continued from previous page)

```
                  ('ROC',
                   (array([0.        , 0.        , 0.        , 0.01428571, 0.01428571,
                           0.02857143, 0.02857143, 0.04285714, 0.04285714, 1.        ␣
↪]),
                    array([0.        , 0.00840336, 0.80672269, 0.80672269, 0.8487395 ,
                           0.8487395 , 0.99159664, 0.99159664, 1.        , 1.        ␣
↪]),
                    array([1.99999994e+00, 9.99999943e-01, 9.98328495e-01, 9.
↪98258660e-01,
                           9.97198567e-01, 9.96795136e-01, 6.43926686e-01, 5.
↪98010642e-01,
                           2.80390400e-01, 2.73644141e-06]))),
                  ('Confusion Matrix',
                       0    1
                   0  67    3
                   1   1  118),
                  ('# Training', 380),
                  ('# Testing', 189)]),
     'score': 0.9833333333333334}]},
  1: {'id': 1,
   'pipeline_class_name': 'CatBoostClassificationPipeline',
   'pipeline_name': 'CatBoost Classifier w/ Simple Imputer',
   'parameters': {'impute_strategy': 'most_frequent',
    'n_estimators': 733,
    'eta': 0.6458941130666562,
    'max_depth': 5},
   'score': 0.974829648719306,
   'high_variance_cv': False,
   'training_time': 14.085938692092896,
   'cv_data': [{'all_objective_scores': OrderedDict([('F1',
                   0.9658119658119659),
                  ('Precision', 0.9826086956521739),
                  ('Recall', 0.9495798319327731),
                  ('AUC', 0.9818913480885312),
                  ('Log Loss', 0.1915961986850965),
                  ('MCC', 0.9119613020615657),
                  ('ROC',
                   (array([0.        , 0.        , 0.        , 0.01408451, 0.01408451,
                           0.02816901, 0.02816901, 0.04225352, 0.04225352, 0.05633803,
                           0.05633803, 0.18309859, 0.18309859, 1.        ]),
                    array([0.        , 0.00840336, 0.13445378, 0.13445378, 0.71428571,
                           0.71428571, 0.95798319, 0.95798319, 0.98319328, 0.98319328,
                           0.99159664, 0.99159664, 1.        , 1.        ]),
                    array([1.99999983e+00, 9.99999825e-01, 9.99998932e-01, 9.
↪99998801e-01,
                           9.99834828e-01, 9.99823067e-01, 4.84501334e-01, 3.
↪89762952e-01,
                           2.57815232e-01, 2.50082621e-01, 1.31435892e-01, 5.
↪28135450e-03,
                           4.39505689e-03, 1.54923584e-06]))),
                  ('Confusion Matrix',
                       0    1
                   0  69    2
                   1   6  113),
                  ('# Training', 379),
                  ('# Testing', 190)]),
     'score': 0.9658119658119659},
```

(continues on next page)

(continued from previous page)

```
    {'all_objective_scores': OrderedDict([('F1', 0.9794238683127572),
                  ('Precision', 0.9596774193548387),
                  ('Recall', 1.0),
                  ('AUC', 0.9944372115043201),
                  ('Log Loss', 0.13193419179315086),
                  ('MCC', 0.9445075449666159),
                  ('ROC',
                   (array([0.        , 0.        , 0.        , 0.01408451, 0.01408451,
                           0.02816901, 0.02816901, 0.04225352, 0.04225352, 1.
↪]),
                    array([0.        , 0.00840336, 0.71428571, 0.71428571, 0.93277311,
                           0.93277311, 0.95798319, 0.95798319, 1.        , 1.
↪]),
                    array([1.99999995e+00, 9.99999955e-01, 9.99950084e-01, 9.
↪99950050e-01,
                           9.98263072e-01, 9.98075367e-01, 9.92107468e-01, 9.
↪81626880e-01,
                           8.48549116e-01, 9.80310846e-08]))),
                  ('Confusion Matrix',
                       0    1
                   0  66    5
                   1   0  119),
                  ('# Training', 379),
                  ('# Testing', 190)]),
     'score': 0.9794238683127572},
    {'all_objective_scores': OrderedDict([('F1', 0.979253112033195),
                  ('Precision', 0.9672131147540983),
                  ('Recall', 0.9915966386554622),
                  ('AUC', 0.9965186074429772),
                  ('Log Loss', 0.08008269134314074),
                  ('MCC', 0.9433286178446474),
                  ('ROC',
                   (array([0.        , 0.        , 0.        , 0.01428571, 0.01428571,
                           0.02857143, 0.02857143, 0.04285714, 0.04285714, 0.05714286,
                           0.05714286, 1.        ]),
                    array([0.        , 0.00840336, 0.86554622, 0.86554622, 0.93277311,
                           0.93277311, 0.97478992, 0.97478992, 0.98319328, 0.98319328,
                           1.        , 1.        ]),
                    array([1.99999993e+00, 9.99999933e-01, 9.96824758e-01, 9.
↪96754879e-01,
                           9.87204663e-01, 9.69865725e-01, 8.64226060e-01, 7.
↪98271414e-01,
                           6.31950137e-01, 5.76401828e-01, 2.61211320e-01, 6.
↪22541309e-08]))),
                  ('Confusion Matrix',
                       0    1
                   0  66    4
                   1   1  118),
                  ('# Training', 380),
                  ('# Testing', 189)]),
     'score': 0.979253112033195}]},
  2: {'id': 2,
   'pipeline_class_name': 'RFClassificationPipeline',
   'pipeline_name': 'Random Forest Classifier w/ One Hot Encoder + Simple Imputer +␣
↪RF Classifier Select From Model',
   'parameters': {'n_estimators': 569,
    'max_depth': 22,
```

(continues on next page)

(continued from previous page)

```
    'impute_strategy': 'most_frequent',
    'percent_features': 0.8593661614465293},
   'score': 0.9638735269218625,
   'high_variance_cv': False,
   'training_time': 10.724831104278564,
   'cv_data': [{'all_objective_scores': OrderedDict([('F1',
                 0.9531914893617022),
                ('Precision', 0.9655172413793104),
                ('Recall', 0.9411764705882353),
                ('AUC', 0.9839625991241567),
                ('Log Loss', 0.15191818463098422),
                ('MCC', 0.8778529707465901),
                ('ROC',
                 (array([0.        , 0.        , 0.        , 0.        , 0.        ,
                         0.01408451, 0.01408451, 0.01408451, 0.01408451, 0.01408451,
                         0.01408451, 0.01408451, 0.01408451, 0.01408451, 0.01408451,
                         0.01408451, 0.01408451, 0.01408451, 0.01408451, 0.02816901,
                         0.02816901, 0.02816901, 0.02816901, 0.04225352, 0.04225352,
                         0.05633803, 0.05633803, 0.07042254, 0.07042254, 0.08450704,
                         0.08450704, 0.09859155, 0.09859155, 0.11267606, 0.11267606,
                         0.12676056, 0.12676056, 0.16901408, 0.16901408, 0.33802817,
                         0.38028169, 0.4084507 , 0.47887324, 0.66197183, 1.
↪]),
                  array([0.        , 0.29411765, 0.38655462, 0.42857143, 0.45378151,
                         0.49579832, 0.5210084 , 0.52941176, 0.56302521, 0.57983193,
                         0.61344538, 0.65546218, 0.67226891, 0.68067227, 0.70588235,
                         0.71428571, 0.73109244, 0.7394958 , 0.75630252, 0.75630252,
                         0.80672269, 0.82352941, 0.88235294, 0.88235294, 0.94117647,
                         0.94117647, 0.94957983, 0.94957983, 0.95798319, 0.95798319,
                         0.96638655, 0.96638655, 0.97478992, 0.97478992, 0.98319328,
                         0.98319328, 0.99159664, 0.99159664, 1.        , 1.        ,
                         1.        , 1.        , 1.        , 1.        , 1.
↪]),
                  array([2.00000000e+00, 1.00000000e+00, 9.98242531e-01, 9.
↪96485062e-01,
                         9.94727592e-01, 9.92970123e-01, 9.91212654e-01, 9.
↪89455185e-01,
                         9.87697715e-01, 9.85940246e-01, 9.84182777e-01, 9.
↪82425308e-01,
                         9.80667838e-01, 9.78910369e-01, 9.77152900e-01, 9.
↪71880492e-01,
                         9.70123023e-01, 9.63093146e-01, 9.56063269e-01, 9.
↪54305800e-01,
                         8.98066784e-01, 8.84007030e-01, 7.48681898e-01, 7.
↪46924429e-01,
                         5.07908612e-01, 5.06151142e-01, 4.88576450e-01, 4.
↪51669596e-01,
                         4.39367311e-01, 4.21792619e-01, 4.20035149e-01, 3.
↪32161687e-01,
                         2.86467487e-01, 2.49560633e-01, 2.33743409e-01, 1.
↪73989455e-01,
                         1.68717047e-01, 1.10720562e-01, 8.26010545e-02, 1.
↪40597540e-02,
                         1.23022847e-02, 5.27240773e-03, 3.51493849e-03, 1.
↪75746924e-03,
                         0.00000000e+00]))),
                ('Confusion Matrix',
```

(continues on next page)

(continued from previous page)

```
                    0    1
                0   67    4
                1    7  112),
               ('# Training', 379),
               ('# Testing', 190)]),
    'score': 0.9531914893617022},
   {'all_objective_scores': OrderedDict([('F1', 0.959349593495935),
               ('Precision', 0.9291338582677166),
               ('Recall', 0.9915966386554622),
               ('AUC', 0.9915374600544443),
               ('Log Loss', 0.11252387200612265),
               ('MCC', 0.8887186971360161),
               ('ROC',
                (array([0.        , 0.        , 0.        , 0.        , 0.        ,
                        0.01408451, 0.01408451, 0.01408451, 0.01408451, 0.01408451,
                        0.01408451, 0.01408451, 0.01408451, 0.01408451, 0.01408451,
                        0.01408451, 0.01408451, 0.01408451, 0.01408451, 0.01408451,
                        0.01408451, 0.07042254, 0.09859155, 0.14084507, 0.14084507,
                        0.25352113, 0.28169014, 0.49295775, 0.52112676, 0.56338028,
                        0.64788732, 1.        ]),
                 array([0.        , 0.28571429, 0.37815126, 0.40336134, 0.45378151,
                        0.49579832, 0.53781513, 0.54621849, 0.57142857, 0.60504202,
                        0.62184874, 0.63865546, 0.66386555, 0.67226891, 0.69747899,
                        0.72268908, 0.7394958 , 0.77310924, 0.78991597, 0.80672269,
                        0.85714286, 0.87394958, 0.8907563 , 0.91596639, 0.93277311,
                        0.99159664, 0.99159664, 0.99159664, 0.99159664, 1.        ,
                        1.        , 1.        , 1.        , 1.        , 1.        ,
                        1.        , 1.        ]),
                 array([2.00000000e+00, 1.00000000e+00, 9.98242531e-01, 9.
↪96485062e-01,
                        9.94727592e-01, 9.92970123e-01, 9.91212654e-01, 9.
↪89455185e-01,
                        9.87697715e-01, 9.84182777e-01, 9.80667838e-01, 9.
↪77152900e-01,
                        9.75395431e-01, 9.73637961e-01, 9.71880492e-01, 9.
↪63093146e-01,
                        9.61335677e-01, 9.47275923e-01, 9.45518453e-01, 9.
↪42003515e-01,
                        9.31458699e-01, 9.22671353e-01, 9.19156415e-01, 9.
↪01581722e-01,
                        8.84007030e-01, 6.88927944e-01, 5.46572935e-01, 5.
↪18453427e-01,
                        4.63971880e-01, 4.62214411e-01, 2.05623902e-01, 1.
↪81019332e-01,
                        1.40597540e-02, 5.27240773e-03, 3.51493849e-03, 1.
↪75746924e-03,
                        0.00000000e+00]))),
               ('Confusion Matrix',
                    0    1
                0   62    9
                1    1  118),
               ('# Training', 379),
               ('# Testing', 190)]),
    'score': 0.959349593495935},
   {'all_objective_scores': OrderedDict([('F1', 0.9790794979079498),
               ('Precision', 0.975),
```

(continues on next page)

(continued from previous page)

```
                 ('Recall', 0.9831932773109243),
                 ('AUC', 0.9966386554621849),
                 ('Log Loss', 0.11505562573216208),
                 ('MCC', 0.9431710402960837),
                 ('ROC',
                  (array([0.        , 0.        , 0.        , 0.        , 0.        ,
                          0.        , 0.        , 0.        , 0.        , 0.        ,
                          0.        , 0.        , 0.        , 0.        , 0.        ,
                          0.        , 0.        , 0.        , 0.        , 0.        ,
                          0.        , 0.        , 0.        , 0.        , 0.01428571,
                          0.01428571, 0.02857143, 0.02857143, 0.04285714, 0.04285714,
                          0.11428571, 0.11428571, 0.4       , 0.44285714, 0.51428571,
                          0.54285714, 0.55714286, 0.58571429, 0.61428571, 0.64285714,
                          0.71428571, 1.        ]),
                   array([0.        , 0.19327731, 0.27731092, 0.35294118, 0.37815126,
                          0.41176471, 0.43697479, 0.47058824, 0.49579832, 0.51260504,
                          0.52941176, 0.55462185, 0.57983193, 0.59663866, 0.6302521 ,
                          0.64705882, 0.66386555, 0.68907563, 0.70588235, 0.76470588,
                          0.78151261, 0.82352941, 0.84033613, 0.8907563 , 0.8907563 ,
                          0.93277311, 0.93277311, 0.98319328, 0.98319328, 0.99159664,
                          0.99159664, 1.        , 1.        , 1.        , 1.        ,
                          1.        , 1.        , 1.        , 1.        , 1.        ,
                          1.        , 1.        , 1.        ]),
                   array([2.00000000e+00, 1.00000000e+00, 9.98242531e-01, 9.
↪96485062e-01,
                          9.94727592e-01, 9.92970123e-01, 9.91212654e-01, 9.
↪89455185e-01,
                          9.87697715e-01, 9.84182777e-01, 9.82425308e-01, 9.
↪80667838e-01,
                          9.73637961e-01, 9.64850615e-01, 9.59578207e-01, 9.
↪50790861e-01,
                          9.43760984e-01, 9.34973638e-01, 9.31458699e-01, 8.
↪94551845e-01,
                          8.91036907e-01, 8.40070299e-01, 8.18980668e-01, 7.
↪59226714e-01,
                          7.50439367e-01, 7.13532513e-01, 6.97715290e-01, 5.
↪37785589e-01,
                          5.00878735e-01, 4.93848858e-01, 4.14762742e-01, 3.
↪84885764e-01,
                          5.27240773e-02, 4.21792619e-02, 1.58172232e-02, 1.
↪40597540e-02,
                          1.23022847e-02, 1.05448155e-02, 5.27240773e-03, 3.
↪51493849e-03,
                          1.75746924e-03, 0.00000000e+00]))),
                 ('Confusion Matrix',
                      0    1
                  0  67    3
                  1   2  117),
                 ('# Training', 380),
                 ('# Testing', 189)]),
     'score': 0.9790794979079498}]},
  3: {'id': 3,
   'pipeline_class_name': 'LogisticRegressionPipeline',
   'pipeline_name': 'Logistic Regression Classifier w/ One Hot Encoder + Simple␣
↪Imputer + Standard Scaler',
   'parameters': {'penalty': 'l2',
    'C': 8.444214828324364,
```

(continues on next page)

(continued from previous page)

```
    'impute_strategy': 'most_frequent'},
   'score': 0.9749409107621198,
   'high_variance_cv': False,
   'training_time': 1.2414195537567139,
   'cv_data': [{'all_objective_scores': OrderedDict([('F1',
                 0.9666666666666667),
                ('Precision', 0.9586776859504132),
                ('Recall', 0.9747899159663865),
                ('AUC', 0.9888744230086401),
                ('Log Loss', 0.15428164230084002),
                ('MCC', 0.9097672817424011),
                ('ROC',
                 (array([0.        , 0.        , 0.        , 0.01408451, 0.01408451,
                         0.02816901, 0.02816901, 0.04225352, 0.04225352, 0.05633803,
                         0.05633803, 0.07042254, 0.07042254, 0.21126761, 0.21126761,
                         1.        ]),
                  array([0.        , 0.00840336, 0.59663866, 0.59663866, 0.85714286,
                         0.85714286, 0.92436975, 0.92436975, 0.94117647, 0.94117647,
                         0.97478992, 0.97478992, 0.99159664, 0.99159664, 1.        ,
                         1.        ]),
                  array([2.00000000e+00, 1.00000000e+00, 9.99791297e-01, 9.
↪99773957e-01,
                         9.78489727e-01, 9.76152961e-01, 8.46462522e-01, 8.
↪36499157e-01,
                         7.95514397e-01, 7.53467428e-01, 5.57693049e-01, 5.
↪27060963e-01,
                         3.63797569e-01, 5.00394079e-04, 4.83643881e-04, 2.
↪02468070e-23]))),
                ('Confusion Matrix',
                     0    1
                 0  66    5
                 1   3  116),
                ('# Training', 379),
                ('# Testing', 190)]),
    'score': 0.9666666666666667},
   {'all_objective_scores': OrderedDict([('F1', 0.979253112033195),
                ('Precision', 0.9672131147540983),
                ('Recall', 0.9915966386554622),
                ('AUC', 0.9984613563735354),
                ('Log Loss', 0.053534692439125425),
                ('MCC', 0.943843520216036),
                ('ROC',
                 (array([0.        , 0.        , 0.        , 0.01408451, 0.01408451,
                         0.02816901, 0.02816901, 0.04225352, 0.04225352, 0.09859155,
                         0.09859155, 1.        ]),
                  array([0.        , 0.00840336, 0.96638655, 0.96638655, 0.97478992,
                         0.97478992, 0.98319328, 0.98319328, 0.99159664, 0.99159664,
                         1.        , 1.        ]),
                  array([2.00000000e+00, 1.00000000e+00, 9.05699907e-01, 8.
↪58386233e-01,
                         8.38649439e-01, 7.59595197e-01, 7.33539439e-01, 7.
↪17465204e-01,
                         5.35804763e-01, 2.35496200e-01, 2.27785221e-01, 1.
↪68800601e-42]))),
                ('Confusion Matrix',
                     0    1
                 0  67    4
```

(continues on next page)

(continued from previous page)

```
                 1   1  118),
                ('# Training', 379),
                ('# Testing', 190)]),
    'score': 0.979253112033195},
   {'all_objective_scores': OrderedDict([('F1', 0.9789029535864979),
                ('Precision', 0.9830508474576272),
                ('Recall', 0.9747899159663865),
                ('AUC', 0.9963985594237695),
                ('Log Loss', 0.07005450515930882),
                ('MCC', 0.9435040132749904),
                ('ROC',
                 (array([0.        , 0.        , 0.        , 0.01428571, 0.01428571,
                         0.02857143, 0.02857143, 0.04285714, 0.04285714, 1.
↪]),
                  array([0.        , 0.00840336, 0.78991597, 0.78991597, 0.97478992,
                         0.97478992, 0.98319328, 0.98319328, 1.        , 1.
↪]),
                  array([2.00000000e+00, 9.99999996e-01, 9.88096914e-01, 9.
↪87891833e-01,
                         5.69263068e-01, 5.39434729e-01, 4.87956909e-01, 4.
↪17720767e-01,
                         3.49086653e-01, 6.54254829e-17]))),
                ('Confusion Matrix',
                     0    1
                 0  68    2
                 1   3  116),
                ('# Training', 380),
                ('# Testing', 189)]),
    'score': 0.9789029535864979}]},
 4: {'id': 4,
  'pipeline_class_name': 'LogisticRegressionPipeline',
  'pipeline_name': 'Logistic Regression Classifier w/ One Hot Encoder + Simple␣
↪Imputer + Standard Scaler',
  'parameters': {'penalty': 'l2',
   'C': 6.239401330891865,
   'impute_strategy': 'median'},
  'score': 0.976371018670262,
  'high_variance_cv': False,
  'training_time': 0.17299914360046387,
  'cv_data': [{'all_objective_scores': OrderedDict([('F1',
                 0.9666666666666667),
                ('Precision', 0.9586776859504132),
                ('Recall', 0.9747899159663865),
                ('AUC', 0.9894662090188188),
                ('Log Loss', 0.14024941178893052),
                ('MCC', 0.9097672817424011),
                ('ROC',
                 (array([0.        , 0.        , 0.        , 0.01408451, 0.01408451,
                         0.02816901, 0.02816901, 0.04225352, 0.04225352, 0.05633803,
                         0.05633803, 0.07042254, 0.07042254, 0.16901408, 0.16901408,
                         1.        ]),
                  array([0.        , 0.00840336, 0.59663866, 0.59663866, 0.85714286,
                         0.85714286, 0.93277311, 0.93277311, 0.94957983, 0.94957983,
                         0.97478992, 0.97478992, 0.99159664, 0.99159664, 1.        ,
                         1.        ]),
                  array([2.00000000e+00, 1.00000000e+00, 9.99666579e-01, 9.
↪99609134e-01,
```

(continues on next page)

(continued from previous page)

```
                           9.74987821e-01, 9.70181648e-01, 8.22360338e-01, 8.
↪20657330e-01,
                           6.90424546e-01, 6.67942883e-01, 5.59753184e-01, 5.
↪55141738e-01,
                           3.76389954e-01, 4.85366478e-03, 2.54470198e-03, 9.
↪55683397e-22]))),
                  ('Confusion Matrix',
                       0    1
                   0  66    5
                   1   3  116),
                  ('# Training', 379),
                  ('# Testing', 190)]),
     'score': 0.9666666666666667},
    {'all_objective_scores': OrderedDict([('F1', 0.979253112033195),
                  ('Precision', 0.9672131147540983),
                  ('Recall', 0.9915966386554622),
                  ('AUC', 0.9986980707776069),
                  ('Log Loss', 0.05225479208679104),
                  ('MCC', 0.943843520216036),
                  ('ROC',
                   (array([0.        , 0.        , 0.        , 0.01408451, 0.01408451,
                           0.04225352, 0.04225352, 0.08450704, 0.08450704, 1.
↪]),
                    array([0.        , 0.00840336, 0.96638655, 0.96638655, 0.98319328,
                           0.98319328, 0.99159664, 0.99159664, 1.        , 1.
↪]),
                    array([2.00000000e+00, 9.99999999e-01, 9.12346914e-01, 8.
↪61927597e-01,
                           7.43130971e-01, 7.06151816e-01, 5.87382590e-01, 3.
↪82148043e-01,
                           2.73319359e-01, 3.76404976e-39]))),
                  ('Confusion Matrix',
                       0    1
                   0  67    4
                   1   1  118),
                  ('# Training', 379),
                  ('# Testing', 190)]),
     'score': 0.979253112033195},
    {'all_objective_scores': OrderedDict([('F1', 0.9831932773109243),
                  ('Precision', 0.9831932773109243),
                  ('Recall', 0.9831932773109243),
                  ('AUC', 0.9963985594237695),
                  ('Log Loss', 0.06645759473825),
                  ('MCC', 0.9546218487394958),
                  ('ROC',
                   (array([0.        , 0.        , 0.        , 0.01428571, 0.01428571,
                           0.02857143, 0.02857143, 0.04285714, 0.04285714, 1.
↪]),
                    array([0.        , 0.00840336, 0.78991597, 0.78991597, 0.97478992,
                           0.97478992, 0.98319328, 0.98319328, 1.        , 1.
↪]),
                    array([1.99999999e+00, 9.99999985e-01, 9.82750455e-01, 9.
↪82286323e-01,
                           5.46864728e-01, 5.21939119e-01, 5.19466434e-01, 4.
↪30449096e-01,
                           3.98630517e-01, 1.92092409e-15]))),
                  ('Confusion Matrix',
```

(continues on next page)

(continued from previous page)

```
                     0    1
                  0  68    2
                  1   2  117),
                 ('# Training', 380),
                 ('# Testing', 189)]),
     'score': 0.9831932773109243}]}},
 'search_order': [0, 1, 2, 3, 4]}
```

### 2.6.7 Regression Example

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


X, y = evalml.demos.load_diabetes()

automl = AutoRegressionSearch(objective="R2", max_pipelines=5)

automl.search(X, y)
```

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

Optimizing for R2. Greater score is better.

Searching up to 5 pipelines.
Possible model types: linear_model, catboost, random_forest
```

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

```
 Random Forest Regressor w/ One Hot ...    20%|        | Elapsed:00:10
 Random Forest Regressor w/ One Hot ...    40%|      | Elapsed:00:16
 Linear Regressor w/ One Hot Encoder...    60%|    | Elapsed:00:16
 Random Forest Regressor w/ One Hot ...    80%|  | Elapsed:00:25
 CatBoost Regressor w/ Simple Imputer:    100%|| Elapsed:00:26
 Optimization finished                    100%|| Elapsed:00:26
```

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

[2]:
```
   id         pipeline_class_name     score  high_variance_cv  \
0   2    LinearRegressionPipeline  0.488703             False
1   0        RFRegressionPipeline  0.422322             False
2   3        RFRegressionPipeline  0.383134             False
3   1        RFRegressionPipeline  0.381204             False
4   4  CatBoostRegressionPipeline  0.250449             False

                                      parameters
0  {'impute_strategy': 'mean', 'normalize': True,...
1  {'n_estimators': 569, 'max_depth': 22, 'impute...
```

(continues on next page)

(continued from previous page)

```
2  {'n_estimators': 609, 'max_depth': 7, 'impute_...
3  {'n_estimators': 369, 'max_depth': 10, 'impute...
4  {'impute_strategy': 'most_frequent', 'n_estima...
```

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

[3]:
```
<evalml.pipelines.regression.linear_regression.LinearRegressionPipeline at␣
↪0x7ff63842d828>
```

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

[4]:
```
<evalml.pipelines.regression.random_forest.RFRegressionPipeline at 0x7ff639548c18>
```

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

```
*****************************************************************************************
* Random Forest Regressor w/ One Hot Encoder + Simple Imputer + RF Regressor Select␣
↪From Model *
*****************************************************************************************

Problem Types: Regression
Model Type: Random Forest
Objective to Optimize: R2 (greater is better)
Number of features: 8

Pipeline Steps
==============
1. One Hot Encoder
2. Simple Imputer
         * impute_strategy : most_frequent
3. RF Regressor Select From Model
         * percent_features : 0.8593661614465293
         * threshold : -inf
4. Random Forest Regressor
         * n_estimators : 569
         * max_depth : 22

Training
========
Training for Regression problems.
Total training time (including CV): 10.1 seconds

Cross Validation
----------
              R2    MAE      MSE  MedianAE  MaxError  ExpVariance # Training #␣
↪Testing
0          0.427 46.033 3276.018    39.699   161.858        0.428    294.000   148.
↪000
1          0.450 48.953 3487.566    44.344   160.513        0.451    295.000   147.
↪000
2          0.390 47.401 3477.117    41.297   171.420        0.390    295.000   147.
↪000
mean       0.422 47.462 3413.567    41.780   164.597        0.423          -
↪-
std        0.031  1.461  119.235     2.360     5.947        0.031          -
↪-
coef of var 0.072  0.031    0.035     0.056     0.036        0.073          -
↪-
```

(continues on next page)

(continued from previous page)

### 2.6.8 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. We can see them here by calling `.plot()`:

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

xgp = XGBoostPipeline(objective='recall', eta=0.5, min_child_weight=5, max_depth=10,␣
↪impute_strategy='mean', percent_features=0.5, number_features=10)
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 Classifier w/ One Hot Encoder + Simple Imputer + RF Classifier Select From␣
↪Model *
*********************************************************************************************

Problem Types: Binary Classification, Multiclass Classification
Model Type: XGBoost Classifier
Objective to Optimize: Recall (greater is better)

Pipeline Steps
==============
1. One Hot Encoder
2. Simple Imputer
         * impute_strategy : mean
3. RF Classifier Select From Model
         * percent_features : 0.5
         * threshold : -inf
4. XGBoost Classifier
         * eta : 0.5
         * max_depth : 10
         * min_child_weight : 5
         * n_estimators : 10
```

You can then fit and score an individual pipeline:

[3]:
```
import evalml

X, y = evalml.demos.load_breast_cancer()
```

(continues on next page)

(continued from previous page)

```
xgp.fit(X, y)

xgp.score(X, y)
```

```
[3]: (0.9775910364145658, {})
```

### 2.6.9 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]
```

(continues on next page)

(continued from previous page)

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

[3]:
```
array([0, 0])
```

### 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*

## 2.6.10 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. You can create your own pipeline like this:

[1]:
```
from evalml.pipelines import PipelineBase
from evalml.pipelines.components import StandardScaler, SimpleImputer
from evalml.pipelines.components.estimators import LogisticRegressionClassifier


# objectives can be either a str or the evalml objective object
objective = 'Precision_Macro'

# components can be passed in as objects or as component name strings
```

(continues on next page)

(continued from previous page)

```
component_list = ['Simple Imputer', StandardScaler(), 'Logistic Regression Classifier
↪']
pipeline = PipelineBase(objective, component_list, n_jobs=-1, random_state=0)
```

[2]:
```
from evalml.demos import load_wine

X, y = load_wine()

pipeline.fit(X, y)
pipeline.score(X, y)
```

[2]:
```
(1.0, {})
```

### 2.6.11 Guardrails

EvalML provides guardrails to help guide you in achieving the highest performing model. These utility functions help deal with overfitting, abnormal data, and missing data. These guardrails can be found under `evalml/guardrails/utils`. Below we will cover abnormal and missing data guardrails. You can find an in-depth look into overfitting guardrails *here*.

#### 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 running the `detect_highly_null()` guardrail, 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.guardrails.utils import detect_highly_null

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

detect_highly_null(X, percent_threshold=0.8)
```

[1]:
```
{2: 0.8}
```

#### Abnormal Data

EvalML provides two utility functions to check for abnormal data: `detect_outliers()` and `detect_id_columns()`.

### 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, `detect_id_columns()` reminds you if these columns exists.

[2]:
```
from evalml.guardrails.utils import detect_id_columns

X = pd.DataFrame([[0, 53, 6325, 5],[1, 90, 6325, 10],[2, 90, 18, 20]], columns=['user_
↪number', 'cost', 'revenue', 'id'])


display(X)
print(detect_id_columns(X, threshold=0.95))
```

```
   user_number  cost  revenue  id
0            0    53     6325   5
1            1    90     6325  10
2            2    90       18  20
```

```
{'id': 1.0, 'user_number': 0.95}
```

### 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. `detect_outliers()` 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)

# outliers
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 `detect_outliers` to rediscover these outliers.

[4]:
```
from evalml.guardrails.utils import detect_outliers

detect_outliers(X)
```

[4]:
```
[3, 25, 55, 72]
```

## 2.6.12 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 guardrails 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

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])

automl = evalml.AutoClassificationSearch(
    max_pipelines=1,
    model_types=["linear_model"],
)

automl.search(X, y)
```

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

Optimizing for Precision. Greater score is better.

Searching up to 1 pipelines.
Possible model types: linear_model

WARNING: Possible label leakage: leaked_feature, leaked_feature_2
```

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

```
 Logistic Regression Classifier w/ O...   100%|| Elapsed:00:01
 Optimization finished                    100%|| Elapsed:00:01
```

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. If you'd like to turn this check off, set `detect_label_leakage=False`.

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

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

[3]:

| | feature | importance |
|---|---|---|
| 0 | leaked_feature | -1.782149 |
| 1 | leaked_feature_2 | -1.638703 |
| 2 | valid_feature | -0.398194 |

### 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.AutoClassificationSearch(
    cv=TimeSeriesSplit(n_splits=6),
    max_pipelines=1
)

automl.search(X, y)
```

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

Optimizing for Precision. Greater score is better.

Searching up to 1 pipelines.
Possible model types: xgboost, catboost, random_forest, linear_model
```

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

```
 CatBoost Classifier w/ Simple Imput...   100%|| Elapsed:00:04
 Optimization finished                    100%|| Elapsed:00:04
```

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)
```

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

Problem Types: Binary Classification, Multiclass Classification
Model Type: CatBoost Classifier
Objective to Optimize: Precision (greater is better)
Number of features: 30

Pipeline Steps
==============
1. Simple Imputer
         * impute_strategy : most_frequent
2. CatBoost Classifier
```

(continues on next page)

(continued from previous page)

```
         * n_estimators : 202
         * eta : 0.602763376071644
         * max_depth : 4

Training
========
Training for Binary Classification problems.
Total training time (including CV): 4.7 seconds

Cross Validation
----------
             Precision    F1  Recall   AUC  Log Loss   MCC # Training # Testing
0                0.953 0.863   0.788 0.938     0.908 0.691     83.000    81.000
1                1.000 0.925   0.860 0.998     0.133 0.862    164.000    81.000
2                0.964 0.982   1.000 0.968     0.153 0.945    245.000    81.000
3                1.000 0.982   0.966 0.999     0.063 0.942    326.000    81.000
4                1.000 0.984   0.969 1.000     0.042 0.931    407.000    81.000
5                0.983 0.983   0.983 0.996     0.065 0.936    488.000    81.000
mean             0.984 0.953   0.928 0.983     0.227 0.885          -         -
std              0.020 0.050   0.084 0.025     0.336 0.100          -         -
coef of var      0.021 0.052   0.091 0.026     1.477 0.113          -         -
```

### 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_class_name     score  high_variance_cv  \
0   0  CatBoostClassificationPipeline  0.983518             False

                                       parameters
0  {'impute_strategy': 'most_frequent', 'n_estima...
```

### 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.AutoClassificationSearch(
    objective="recall",
    max_pipelines=3,
    detect_label_leakage=True
)
automl.search(X_train, y_train)
```

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

Optimizing for Recall. Greater score is better.

Searching up to 3 pipelines.
Possible model types: xgboost, catboost, random_forest, linear_model
```

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

```
 CatBoost Classifier w/ Simple Imput...    33%|      | Elapsed:00:02
 CatBoost Classifier w/ Simple Imput...    67%|   | Elapsed:00:17
 Random Forest Classifier w/ One Hot...   100%|| Elapsed:00:27
 Optimization finished                    100%|| Elapsed:00:27
```

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)
```

[9]:
```
(0.9305555555555556, {})
```

### 2.6.13 Changelog

**Future Releases**

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

**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
  - Dropped support for Python 3.5 #438
  - Removed unused *apply.py* file #449
  - Clean up requirements.txt to remove unused deps #451
- **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
- **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
- **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

### 2.6.14 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*)

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

**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

| | |
|---|---|
| *load_data* | Load features and labels from file(s). |
| *split_data* | Splits data into train and test sets. |

### 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(s).

**Parameters**

- **path** (*str*) – path to file(s)
- **index** (*str*) – column for index
- **label** (*str*) – column for labels
- **n_rows** (*int*) – number of rows to return
- **drop** (*list*) – columns to drop
- **verbose** (*bool*) – whether to print information about features and labels

**Returns** features and labels

**Return type** pd.DataFrame, 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*) – 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

| | |
|---|---|
| *AutoClassificationSearch* | Automatic pipeline search class for classification problems |
| *AutoRegressionSearch* | Automatic pipeline search for regression problems |

### evalml.AutoClassificationSearch

**class** evalml.**AutoClassificationSearch**(*objective=None*, *multiclass=False*, *max_pipelines=None*, *max_time=None*, *patience=None*, *tolerance=None*, *model_types=None*, *cv=None*, *tuner=None*, *detect_label_leakage=True*, *start_iteration_callback=None*, *add_result_callback=None*, *additional_objectives=None*, *random_state=0*, *n_jobs=-1*, *verbose=True*)

Automatic pipeline search class for classification problems

#### Methods

| | |
|---|---|
| *__init__* | Automated classifier pipeline search |

#### evalml.AutoClassificationSearch.__init__

AutoClassificationSearch.**__init__**(*objective=None*, *multiclass=False*, *max_pipelines=None*, *max_time=None*, *patience=None*, *tolerance=None*, *model_types=None*, *cv=None*, *tuner=None*, *detect_label_leakage=True*, *start_iteration_callback=None*, *add_result_callback=None*, *additional_objectives=None*, *random_state=0*, *n_jobs=-1*, *verbose=True*)

Automated classifier pipeline search

**Parameters**

- **objective** (*Object*) – the objective to optimize
- **multiclass** (*bool*) – If True, expecting multiclass data. By default: False.
- **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.
- **model_types** (*list*) – The model types to search. By default searches over all model_types. Run evalml.list_model_types("classification") to see options.
- **cv** – cross validation method to use. By default StratifiedKFold
- **tuner** – the tuner class to use. Defaults to scikit-optimize tuner
- **detect_label_leakage** (*bool*) – If True, check input features for label leakage and warn if found. Defaults to true.
- **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*) – the random_state
- **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

#### Attributes

| | |
|---|---|
| best_pipeline | Returns the best model found |
| rankings | Returns the rankings of the models searched |

### evalml.AutoRegressionSearch

**class** evalml.**AutoRegressionSearch**(*objective=None*, *max_pipelines=None*, *max_time=None*, *patience=None*, *tolerance=None*, *model_types=None*, *cv=None*, *tuner=None*, *detect_label_leakage=True*, *start_iteration_callback=None*, *add_result_callback=None*, *additional_objectives=None*, *random_state=0*, *n_jobs=-1*, *verbose=True*)

Automatic pipeline search for regression problems

#### Methods

| | |
|---|---|
| *__init__* | Automated regressors pipeline search |

### evalml.AutoRegressionSearch.__init__

AutoRegressionSearch.**__init__**(*objective=None*, *max_pipelines=None*, *max_time=None*, *patience=None*, *tolerance=None*, *model_types=None*, *cv=None*, *tuner=None*, *detect_label_leakage=True*, *start_iteration_callback=None*, *add_result_callback=None*, *additional_objectives=None*, *random_state=0*, *n_jobs=-1*, *verbose=True*)

Automated regressors pipeline search

**Parameters**

- **objective** (*Object*) – the objective to optimize
- **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.
- **model_types** (*list*) – The model types to search. By default searches over all model_types. Run evalml.list_model_types("regression") to see options.
- **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.
- **cv** – cross validation method to use. By default StratifiedKFold
- **tuner** – the tuner class to use. Defaults to scikit-optimize tuner
- **detect_label_leakage** (*bool*) – If True, check input features for label leakage and warn if found. Defaults to true.
- **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*) – the random_state
- **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

### Attributes

| | |
|---|---|
| best_pipeline | Returns the best model found |
| rankings | Returns the rankings of the models searched |

### Plotting

| | |
|---|---|
| *AutoClassificationSearch.plot.get_roc_data* | Gets data that can be used to create a ROC plot. |
| *AutoClassificationSearch.plot.generate_roc_plot* | Generate Receiver Operating Characteristic (ROC) plot for a given pipeline using cross-validation using the data returned from get_roc_data(). |
| *AutoRegressionSearch.plot.get_roc_data* | Gets data that can be used to create a ROC plot. |
| *AutoRegressionSearch.plot.generate_roc_plot* | Generate Receiver Operating Characteristic (ROC) plot for a given pipeline using cross-validation using the data returned from get_roc_data(). |
| *AutoClassificationSearch.plot.get_confusion_matrix_data* | Gets data that can be used to create a confusion matrix plot. |
| *AutoClassificationSearch.plot.generate_confusion_matrix* | Generate confusion matrix plot for a given pipeline using the data returned from get_confusion_matrix_data(). |
| *AutoRegressionSearch.plot.get_confusion_matrix_data* | Gets data that can be used to create a confusion matrix plot. |
| *AutoRegressionSearch.plot.generate_confusion_matrix* | Generate confusion matrix plot for a given pipeline using the data returned from get_confusion_matrix_data(). |

#### evalml.AutoClassificationSearch.plot.get_roc_data

AutoClassificationSearch.plot.**get_roc_data**(*pipeline_id*)

Gets data that can be used to create a ROC plot.

**Returns** Dictionary containing metrics used to generate an ROC plot.

#### evalml.AutoClassificationSearch.plot.generate_roc_plot

AutoClassificationSearch.plot.**generate_roc_plot**(*pipeline_id*)

Generate Receiver Operating Characteristic (ROC) plot for a given pipeline using cross-validation using the data returned from get_roc_data().

**Returns** plotly.Figure representing the ROC plot generated

#### evalml.AutoRegressionSearch.plot.get_roc_data

AutoRegressionSearch.plot.**get_roc_data**(*pipeline_id*)

Gets data that can be used to create a ROC plot.

**Returns** Dictionary containing metrics used to generate an ROC plot.

#### evalml.AutoRegressionSearch.plot.generate_roc_plot

AutoRegressionSearch.plot.**generate_roc_plot**(*pipeline_id*)

Generate Receiver Operating Characteristic (ROC) plot for a given pipeline using cross-validation using the data returned from get_roc_data().

**Returns** plotly.Figure representing the ROC plot generated

### evalml.AutoClassificationSearch.plot.get_confusion_matrix_data

AutoClassificationSearch.plot.**get_confusion_matrix_data**(*pipeline_id*)

Gets data that can be used to create a confusion matrix plot.

**Returns** List containing information used to generate a confusion matrix plot. Each element in the list contains the confusion matrix data for that fold.

### evalml.AutoClassificationSearch.plot.generate_confusion_matrix

AutoClassificationSearch.plot.**generate_confusion_matrix**(*pipeline_id*, *fold_num=None*)

Generate confusion matrix plot for a given pipeline using the data returned from get_confusion_matrix_data().

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

### evalml.AutoRegressionSearch.plot.get_confusion_matrix_data

AutoRegressionSearch.plot.**get_confusion_matrix_data**(*pipeline_id*)

Gets data that can be used to create a confusion matrix plot.

**Returns** List containing information used to generate a confusion matrix plot. Each element in the list contains the confusion matrix data for that fold.

### evalml.AutoRegressionSearch.plot.generate_confusion_matrix

AutoRegressionSearch.plot.**generate_confusion_matrix**(*pipeline_id*, *fold_num=None*)

Generate confusion matrix plot for a given pipeline using the data returned from get_confusion_matrix_data().

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

## Model Types

| | |
|---|---|
| *list_model_types* | List model type for a particular problem type |

### evalml.list_model_types

evalml.**list_model_types**(*problem_type*)

List model type for a particular problem type

**Parameters** **problem_types** (*ProblemTypes or str*) – binary, multiclass, or regression

**Returns** model_types, list of model types

## Components

### Transformers

| | |
|---|---|
| *OneHotEncoder* | Creates one-hot encoding for non-numeric data |
| *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 |
| *SimpleImputer* | Imputes missing data with either mean, median and most_frequent for numerical data or most_frequent for categorical data |
| *StandardScaler* | Standardize features: removes mean and scales to unit variance |

### evalml.pipelines.components.OneHotEncoder

**class** evalml.pipelines.components.**OneHotEncoder**

Creates one-hot encoding for non-numeric data

#### Methods

| | |
|---|---|
| *__init__* | Initialize self. |
| *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* | Transforms data X |

#### evalml.pipelines.components.OneHotEncoder.__init__

OneHotEncoder.**__init__**()

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

#### 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*)

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*)

Selects top features based on importance weights using a Random Forest regressor

#### Methods

| | |
|---|---|
| *__init__* | Initialize self. |
| *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*)

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

#### 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*)

Selects top features based on importance weights using a Random Forest classifier

#### Methods

| | |
|---|---|
| *__init__* | Initialize self. |
| *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*)

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

#### 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

### evalml.pipelines.components.SimpleImputer

**class** evalml.pipelines.components.**SimpleImputer**(*impute_strategy='most_frequent'*)

Imputes missing data with either mean, median and most_frequent for numerical data or most_frequent for categorical data

#### Methods

| | |
|---|---|
| *__init__* | Initialize self. |
| *describe* | Describe a component and its parameters |
| *fit* | Fits component to data |
| *fit_transform* | Fits imputer on data X then imputes missing values in X |
| *transform* | Transforms data X by imputing missing values |

#### evalml.pipelines.components.SimpleImputer.__init__

SimpleImputer.**__init__**(*impute_strategy='most_frequent'*)

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

#### 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 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.SimpleImputer.fit_transform

SimpleImputer.**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.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**
Standardize features: removes mean and scales to unit variance

#### Methods

| | |
|---|---|
| *__init__* | Initialize self. |
| *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__**()
Initialize self. See help(type(self)) for accurate signature.

#### 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

## Estimators

| | |
|---|---|
| *LogisticRegressionClassifier* | Logistic Regression Classifier |
| *RandomForestClassifier* | Random Forest Classifier |

Continued on next page

Table 16 – continued from previous page

| | |
|---|---|
| *XGBoostClassifier* | XGBoost Classifier |
| *LinearRegressor* | Linear Regressor |
| *RandomForestRegressor* | Random Forest Regressor |

### evalml.pipelines.components.LogisticRegressionClassifier

**class** evalml.pipelines.components.**LogisticRegressionClassifier**(*penalty='l2'*, *C=1.0*, *n_jobs=-1*, *random_state=0*)

Logistic Regression Classifier

#### Methods

| | |
|---|---|
| *__init__* | Initialize self. |
| *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*)

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

#### 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.RandomForestClassifier

**class** evalml.pipelines.components.**RandomForestClassifier**(*n_estimators=10*, *max_depth=None*, *n_jobs=-1*, *random_state=0*)

Random Forest Classifier

### Methods

| | |
|---|---|
| *__init__* | Initialize self. |
| *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=10*, *max_depth=None*, *n_jobs=-1*, *random_state=0*)

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

### 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.XGBoostClassifier

**class** evalml.pipelines.components.**XGBoostClassifier**(*eta=0.1*, *max_depth=3*, *min_child_weight=1*, *n_estimators=100*, *random_state=0*)

XGBoost Classifier

### Methods

| | |
|---|---|
| *__init__* | Initialize self. |
| *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=3*, *min_child_weight=1*, *n_estimators=100*, *random_state=0*)

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

#### 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.LinearRegressor

**class** evalml.pipelines.components.**LinearRegressor**(*fit_intercept=True*, *normalize=False*, *n_jobs=-1*)
Linear Regressor

#### Methods

| | |
|---|---|
| *__init__* | Initialize self. |
| *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*)
Initialize self. See help(type(self)) for accurate signature.

#### 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.RandomForestRegressor

**class** evalml.pipelines.components.**RandomForestRegressor**(*n_estimators=10*, *max_depth=None*, *n_jobs=-1*, *random_state=0*)

Random Forest Regressor

#### Methods

| | |
|---|---|
| *__init__* | Initialize self. |
| *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=10*, *max_depth=None*, *n_jobs=-1*, *random_state=0*)

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

### 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

## Pipelines

| | |
|---|---|
| *get_pipelines* | Returns potential pipelines by model type |
| *save_pipeline* | Saves pipeline at file path |

Continued on next page

Table 22 – continued from previous page

| | |
|---|---|
| *load_pipeline* | Loads pipeline at file path |

### evalml.pipelines.get_pipelines

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

Returns potential pipelines by model type

**Parameters**

- **problem_type** (*ProblemTypes or str*) – the problem type the pipelines work for.
- **model_types** (*list[ModelTypes or str]*) – model types to match. if none, return all pipelines

**Returns** pipelines, list of all pipeline

### evalml.pipelines.save_pipeline

evalml.pipelines.**save_pipeline**(*pipeline*, *file_path*)

Saves pipeline at file path

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

**Returns** None

### evalml.pipelines.load_pipeline

evalml.pipelines.**load_pipeline**(*file_path*)

Loads pipeline at file path

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

**Returns** Pipeline obj

| | |
|---|---|
| *PipelineBase* | |
| *RFClassificationPipeline* | Random Forest Pipeline for both binary and multiclass classification |
| *XGBoostPipeline* | XGBoost Pipeline for both binary and multiclass classification |
| *LogisticRegressionPipeline* | Logistic Regression Pipeline for both binary and multiclass classification |
| *RFRegressionPipeline* | Random Forest Pipeline for regression problems |
| *LinearRegressionPipeline* | Linear Regression Pipeline for regression problems |

### evalml.pipelines.PipelineBase

**class** evalml.pipelines.**PipelineBase**(*objective*, *component_list*, *n_jobs*, *random_state*)

#### Methods

| | |
|---|---|
| *__init__* | Machine learning pipeline made out of transformers and a estimator. |
| *describe* | Outputs pipeline details including component parameters |
| *feature_importance_graph* | Generate a bar graph of the pipeline's feature importances |
| *fit* | Build a model |
| *get_component* | Returns component by name |
| *graph* | Generate an image representing the pipeline graph |
| *predict* | Make predictions using selected features. |
| *predict_proba* | Make probability estimates for labels. |
| *score* | Evaluate model performance on current and additional objectives |

### evalml.pipelines.PipelineBase.__init__

PipelineBase.**__init__**(*objective*, *component_list*, *n_jobs*, *random_state*)

Machine learning pipeline made out of transformers and a estimator.

**Parameters**

- **objective** (*Object*) – the objective to optimize
- **component_list** (*list*) – List of components in order
- **random_state** (*int*) – random seed/state
- **n_jobs** (*int*) – Number of jobs to run in parallel

### evalml.pipelines.PipelineBase.describe

PipelineBase.**describe**(*return_dict=False*)

Outputs pipeline details including component parameters

**Parameters** **return_dict** (*bool*) – If True, return dictionary of information about pipeline. Defaults to false

**Returns** dictionary of all component parameters if return_dict is True, else None

**Return type** dict

### evalml.pipelines.PipelineBase.feature_importance_graph

PipelineBase.**feature_importance_graph**(*show_all_features=False*)

Generate a bar graph of the pipeline's feature importances

**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 importances

### evalml.pipelines.PipelineBase.fit

PipelineBase.**fit**(*X*, *y*, *objective_fit_size=0.2*)

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]
- **feature_types** (*list, optional*) – list of feature types. either numeric of categorical. categorical features will automatically be encoded

**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.predict

PipelineBase.**predict**(*X*)

Make predictions using selected features.

**Parameters** **X** (*pd.DataFrame or np.array*) – data of shape [n_samples, n_features]

**Returns** estimated labels

**Return type** pd.Series

### evalml.pipelines.PipelineBase.predict_proba

PipelineBase.**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.PipelineBase.score

PipelineBase.**score**(*X*, *y*, *other_objectives=None*)

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]
- **other_objectives** (*list*) – list of other objectives to score

**Returns** score, ordered dictionary of other objective scores

**Return type** float, dict

## evalml.pipelines.RFClassificationPipeline

**class** evalml.pipelines.**RFClassificationPipeline**(*objective*, *n_estimators*, *max_depth*, *impute_strategy*, *percent_features*, *number_features*, *n_jobs=-1*, *random_state=0*)

Random Forest Pipeline for both binary and multiclass classification

### Methods

| | |
|---|---|
| *__init__* | Machine learning pipeline made out of transformers and a estimator. |
| *describe* | Outputs pipeline details including component parameters |
| *feature_importance_graph* | Generate a bar graph of the pipeline's feature importances |
| *fit* | Build a model |
| *get_component* | Returns component by name |
| *graph* | Generate an image representing the pipeline graph |
| *predict* | Make predictions using selected features. |
| *predict_proba* | Make probability estimates for labels. |
| *score* | Evaluate model performance on current and additional objectives |

### evalml.pipelines.RFClassificationPipeline.__init__

RFClassificationPipeline.**__init__**(*objective*, *n_estimators*, *max_depth*, *impute_strategy*, *percent_features*, *number_features*, *n_jobs=-1*, *random_state=0*)

Machine learning pipeline made out of transformers and a estimator.

**Parameters**

- **objective** (*Object*) – the objective to optimize
- **component_list** (*list*) – List of components in order
- **random_state** (*int*) – random seed/state

- **n_jobs** (*int*) – Number of jobs to run in parallel

### evalml.pipelines.RFClassificationPipeline.describe

RFClassificationPipeline.**describe**(*return_dict=False*)

Outputs pipeline details including component parameters

**Parameters** **return_dict** (*bool*) – If True, return dictionary of information about pipeline. Defaults to false

**Returns** dictionary of all component parameters if return_dict is True, else None

**Return type** dict

### evalml.pipelines.RFClassificationPipeline.feature_importance_graph

RFClassificationPipeline.**feature_importance_graph**(*show_all_features=False*)

Generate a bar graph of the pipeline's feature importances

**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 importances

### evalml.pipelines.RFClassificationPipeline.fit

RFClassificationPipeline.**fit**(*X*, *y*, *objective_fit_size=0.2*)

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]
- **feature_types** (*list, optional*) – list of feature types. either numeric of categorical. categorical features will automatically be encoded

**Returns** self

### evalml.pipelines.RFClassificationPipeline.get_component

RFClassificationPipeline.**get_component**(*name*)

Returns component by name

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

**Returns** component to return

**Return type** Component

#### evalml.pipelines.RFClassificationPipeline.graph

RFClassificationPipeline.**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.RFClassificationPipeline.predict

RFClassificationPipeline.**predict**(*X*)
Make predictions using selected features.

**Parameters** **X** (*pd.DataFrame or np.array*) – data of shape [n_samples, n_features]

**Returns** estimated labels

**Return type** pd.Series

#### evalml.pipelines.RFClassificationPipeline.predict_proba

RFClassificationPipeline.**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.RFClassificationPipeline.score

RFClassificationPipeline.**score**(*X*, *y*, *other_objectives=None*)
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]
- **other_objectives** (*list*) – list of other objectives to score

**Returns** score, ordered dictionary of other objective scores

**Return type** float, dict

### evalml.pipelines.XGBoostPipeline

**class** evalml.pipelines.**XGBoostPipeline**(*objective*, *eta*, *min_child_weight*, *max_depth*, *impute_strategy*, *percent_features*, *number_features*, *n_estimators=10*, *n_jobs=-1*, *random_state=0*)
XGBoost Pipeline for both binary and multiclass classification

### Methods

| | |
|---|---|
| *__init__* | Machine learning pipeline made out of transformers and a estimator. |
| *describe* | Outputs pipeline details including component parameters |
| *feature_importance_graph* | Generate a bar graph of the pipeline's feature importances |
| *fit* | Build a model |
| *get_component* | Returns component by name |
| *graph* | Generate an image representing the pipeline graph |
| *predict* | Make predictions using selected features. |
| *predict_proba* | Make probability estimates for labels. |
| *score* | Evaluate model performance on current and additional objectives |

### evalml.pipelines.XGBoostPipeline.__init__

XGBoostPipeline.**__init__**(*objective*, *eta*, *min_child_weight*, *max_depth*, *impute_strategy*, *percent_features*, *number_features*, *n_estimators=10*, *n_jobs=-1*, *random_state=0*)

Machine learning pipeline made out of transformers and a estimator.

**Parameters**

- **objective** (*Object*) – the objective to optimize
- **component_list** (*list*) – List of components in order
- **random_state** (*int*) – random seed/state
- **n_jobs** (*int*) – Number of jobs to run in parallel

### evalml.pipelines.XGBoostPipeline.describe

XGBoostPipeline.**describe**(*return_dict=False*)

Outputs pipeline details including component parameters

**Parameters** **return_dict** (*bool*) – If True, return dictionary of information about pipeline. Defaults to false

**Returns** dictionary of all component parameters if return_dict is True, else None

**Return type** dict

### evalml.pipelines.XGBoostPipeline.feature_importance_graph

XGBoostPipeline.**feature_importance_graph**(*show_all_features=False*)

Generate a bar graph of the pipeline's feature importances

**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 importances

### evalml.pipelines.XGBoostPipeline.fit

XGBoostPipeline.**fit**(*X*, *y*, *objective_fit_size=0.2*)

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]
- **feature_types** (*list, optional*) – list of feature types. either numeric of categorical. categorical features will automatically be encoded

**Returns** self

### evalml.pipelines.XGBoostPipeline.get_component

XGBoostPipeline.**get_component**(*name*)

Returns component by name

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

**Returns** component to return

**Return type** Component

### evalml.pipelines.XGBoostPipeline.graph

XGBoostPipeline.**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.XGBoostPipeline.predict

XGBoostPipeline.**predict**(*X*)

Make predictions using selected features.

**Parameters** **X** (*pd.DataFrame or np.array*) – data of shape [n_samples, n_features]

**Returns** estimated labels

**Return type** pd.Series

### evalml.pipelines.XGBoostPipeline.predict_proba

XGBoostPipeline.**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.XGBoostPipeline.score

XGBoostPipeline.**score**(*X*, *y*, *other_objectives=None*)

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]
- **other_objectives** (*list*) – list of other objectives to score

**Returns** score, ordered dictionary of other objective scores

**Return type** float, dict

## evalml.pipelines.LogisticRegressionPipeline

**class** evalml.pipelines.**LogisticRegressionPipeline**(*objective*, *penalty*, *C*, *impute_strategy*, *number_features*, *n_jobs=-1*, *random_state=0*)

Logistic Regression Pipeline for both binary and multiclass classification

### Methods

| | |
|---|---|
| *__init__* | Machine learning pipeline made out of transformers and a estimator. |
| *describe* | Outputs pipeline details including component parameters |
| *feature_importance_graph* | Generate a bar graph of the pipeline's feature importances |
| *fit* | Build a model |
| *get_component* | Returns component by name |
| *graph* | Generate an image representing the pipeline graph |
| *predict* | Make predictions using selected features. |
| *predict_proba* | Make probability estimates for labels. |
| *score* | Evaluate model performance on current and additional objectives |

### evalml.pipelines.LogisticRegressionPipeline.__init__

LogisticRegressionPipeline.**__init__**(*objective*, *penalty*, *C*, *impute_strategy*, *number_features*, *n_jobs=-1*, *random_state=0*)

Machine learning pipeline made out of transformers and a estimator.

**Parameters**

- **objective** (*Object*) – the objective to optimize
- **component_list** (*list*) – List of components in order

- **random_state** (*int*) – random seed/state
- **n_jobs** (*int*) – Number of jobs to run in parallel

### evalml.pipelines.LogisticRegressionPipeline.describe

LogisticRegressionPipeline.**describe**(*return_dict=False*)

Outputs pipeline details including component parameters

**Parameters** **return_dict** (*bool*) – If True, return dictionary of information about pipeline. Defaults to false

**Returns** dictionary of all component parameters if return_dict is True, else None

**Return type** dict

### evalml.pipelines.LogisticRegressionPipeline.feature_importance_graph

LogisticRegressionPipeline.**feature_importance_graph**(*show_all_features=False*)

Generate a bar graph of the pipeline's feature importances

**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 importances

### evalml.pipelines.LogisticRegressionPipeline.fit

LogisticRegressionPipeline.**fit**(*X*, *y*, *objective_fit_size=0.2*)

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]
- **feature_types** (*list, optional*) – list of feature types. either numeric of categorical. categorical features will automatically be encoded

**Returns** self

### evalml.pipelines.LogisticRegressionPipeline.get_component

LogisticRegressionPipeline.**get_component**(*name*)

Returns component by name

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

**Returns** component to return

**Return type** Component

### evalml.pipelines.LogisticRegressionPipeline.graph

LogisticRegressionPipeline.**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.LogisticRegressionPipeline.predict

LogisticRegressionPipeline.**predict**(*X*)

Make predictions using selected features.

**Parameters** **X** (*pd.DataFrame or np.array*) – data of shape [n_samples, n_features]

**Returns** estimated labels

**Return type** pd.Series

### evalml.pipelines.LogisticRegressionPipeline.predict_proba

LogisticRegressionPipeline.**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.LogisticRegressionPipeline.score

LogisticRegressionPipeline.**score**(*X*, *y*, *other_objectives=None*)

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]
- **other_objectives** (*list*) – list of other objectives to score

**Returns** score, ordered dictionary of other objective scores

**Return type** float, dict

## evalml.pipelines.RFRegressionPipeline

**class** evalml.pipelines.**RFRegressionPipeline**(*objective*, *n_estimators*, *max_depth*, *impute_strategy*, *percent_features*, *number_features*, *n_jobs=-1*, *random_state=0*)

Random Forest Pipeline for regression problems

### Methods

| | |
|---|---|
| *__init__* | Machine learning pipeline made out of transformers and a estimator. |
| *describe* | Outputs pipeline details including component parameters |
| *feature_importance_graph* | Generate a bar graph of the pipeline's feature importances |
| *fit* | Build a model |
| *get_component* | Returns component by name |
| *graph* | Generate an image representing the pipeline graph |
| *predict* | Make predictions using selected features. |
| *predict_proba* | Make probability estimates for labels. |
| *score* | Evaluate model performance on current and additional objectives |

#### evalml.pipelines.RFRegressionPipeline.__init__

RFRegressionPipeline.**__init__**(*objective*, *n_estimators*, *max_depth*, *impute_strategy*, *percent_features*, *number_features*, *n_jobs=-1*, *random_state=0*)

Machine learning pipeline made out of transformers and a estimator.

**Parameters**

- **objective** (*Object*) – the objective to optimize
- **component_list** (*list*) – List of components in order
- **random_state** (*int*) – random seed/state
- **n_jobs** (*int*) – Number of jobs to run in parallel

#### evalml.pipelines.RFRegressionPipeline.describe

RFRegressionPipeline.**describe**(*return_dict=False*)

Outputs pipeline details including component parameters

**Parameters** **return_dict** (*bool*) – If True, return dictionary of information about pipeline. Defaults to false

**Returns** dictionary of all component parameters if return_dict is True, else None

**Return type** dict

#### evalml.pipelines.RFRegressionPipeline.feature_importance_graph

RFRegressionPipeline.**feature_importance_graph**(*show_all_features=False*)

Generate a bar graph of the pipeline's feature importances

**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 importances

### evalml.pipelines.RFRegressionPipeline.fit

RFRegressionPipeline.**fit**(*X*, *y*, *objective_fit_size=0.2*)

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]
- **feature_types** (*list, optional*) – list of feature types. either numeric of categorical. categorical features will automatically be encoded

**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.predict

RFRegressionPipeline.**predict**(*X*)

Make predictions using selected features.

**Parameters** **X** (*pd.DataFrame or np.array*) – data of shape [n_samples, n_features]

**Returns** estimated labels

**Return type** pd.Series

### evalml.pipelines.RFRegressionPipeline.predict_proba

RFRegressionPipeline.**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.RFRegressionPipeline.score

RFRegressionPipeline.**score**(*X*, *y*, *other_objectives=None*)

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]
- **other_objectives** (*list*) – list of other objectives to score

**Returns** score, ordered dictionary of other objective scores

**Return type** float, dict

### evalml.pipelines.LinearRegressionPipeline

**class** evalml.pipelines.**LinearRegressionPipeline**(*objective*, *number_features*, *impute_strategy*, *normalize=False*, *fit_intercept=True*, *random_state=0*, *n_jobs=-1*)

Linear Regression Pipeline for regression problems

#### Methods

| | |
|---|---|
| *__init__* | Machine learning pipeline made out of transformers and a estimator. |
| *describe* | Outputs pipeline details including component parameters |
| *feature_importance_graph* | Generate a bar graph of the pipeline's feature importances |
| *fit* | Build a model |
| *get_component* | Returns component by name |
| *graph* | Generate an image representing the pipeline graph |
| *predict* | Make predictions using selected features. |
| *predict_proba* | Make probability estimates for labels. |
| *score* | Evaluate model performance on current and additional objectives |

#### evalml.pipelines.LinearRegressionPipeline.__init__

LinearRegressionPipeline.**__init__**(*objective*, *number_features*, *impute_strategy*, *normalize=False*, *fit_intercept=True*, *random_state=0*, *n_jobs=-1*)

Machine learning pipeline made out of transformers and a estimator.

**Parameters**

- **objective** (*Object*) – the objective to optimize
- **component_list** (*list*) – List of components in order
- **random_state** (*int*) – random seed/state
- **n_jobs** (*int*) – Number of jobs to run in parallel

### evalml.pipelines.LinearRegressionPipeline.describe

LinearRegressionPipeline.**describe**(*return_dict=False*)

Outputs pipeline details including component parameters

**Parameters** **return_dict** (*bool*) – If True, return dictionary of information about pipeline. Defaults to false

**Returns** dictionary of all component parameters if return_dict is True, else None

**Return type** dict

### evalml.pipelines.LinearRegressionPipeline.feature_importance_graph

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

Generate a bar graph of the pipeline's feature importances

**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 importances

### evalml.pipelines.LinearRegressionPipeline.fit

LinearRegressionPipeline.**fit**(*X*, *y*, *objective_fit_size=0.2*)

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]
- **feature_types** (*list, optional*) – list of feature types. either numeric of categorical. categorical features will automatically be encoded

**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.predict

LinearRegressionPipeline.**predict**(*X*)

Make predictions using selected features.

**Parameters** **X** (*pd.DataFrame or np.array*) – data of shape [n_samples, n_features]

**Returns** estimated labels

**Return type** pd.Series

### evalml.pipelines.LinearRegressionPipeline.predict_proba

LinearRegressionPipeline.**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.LinearRegressionPipeline.score

LinearRegressionPipeline.**score**(*X*, *y*, *other_objectives=None*)

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]
- **other_objectives** (*list*) – list of other objectives to score

**Returns** score, ordered dictionary of other objective scores

**Return type** float, dict

## Plotting

PipelineBase.plot

### Objective Functions

### Domain Specific

| | |
|---|---|
| *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'*, *verbose=False*)

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 transaction is fraud given predicted probabilities, dataframe with transaction amount, and threshold |
| *fit* | Learn the objective function based on the predictions from a model. |
| *objective_function* | Calculate amount lost to fraud per transaction given predictions, true values, and dataframe with transaction amount |
| *predict* | Apply the learned objective function to the output of a model. |
| *score* | Calculate score from applying fitted objective to predicted values |
| *supports_problem_type* | Checks if objective supports given ProblemType |

#### evalml.objectives.FraudCost.__init__

FraudCost.**__init__**(*retry_percentage=0.5*, *interchange_fee=0.02*, *fraud_payout_percentage=1.0*, *amount_col='amount'*, *verbose=False*)

Create instance of FraudCost

**Parameters**

- **retry_percentage** (*float*) – what percentage of customers will retry a transaction if it is declined? Between 0 and 1. Defaults to .5
- **interchange_fee** (*float*) – how much of each successful transaction do you collect? Between 0 and 1. Defaults to .02
- **fraud_payout_percentage** (*float*) – how percentage of fraud will you be unable 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**(*y_predicted*, *extra_cols*, *threshold*)

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

**Parameters**

- **y_predicted** (*pd.Series*) – predicted labels
- **extra_cols** (*pd.DataFrame*) – extra data needed
- **threshold** (*float*) – dollar threshold to determine if transaction is fraud

**Returns** series of predicted fraud label using extra cols and threshold

**Return type** pd.Series

### evalml.objectives.FraudCost.fit

FraudCost.**fit**(*y_predicted*, *y_true*, *extra_cols=None*)

Learn the objective function based on the predictions from a model.

If needs_fitting is false, this method won't be called

**Parameters**

- **y_predicted** (*list*) – the predictions from the model. If needs_proba is True, it is the probability estimates
- **y_true** (*list*) – the ground truth for the predictions.
- **extra_cols** (*pd.DataFrame*) – any extra columns that are needed from training data to fit. Only provided if uses_extra_columns is True.

**Returns** self

### evalml.objectives.FraudCost.objective_function

FraudCost.**objective_function**(*y_predicted*, *y_true*, *extra_cols*)

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
- **extra_cols** (*pd.DataFrame*) – extra data needed

**Returns** amount lost to fraud per transaction

**Return type** float

### evalml.objectives.FraudCost.predict

FraudCost.**predict**(*y_predicted*, *extra_cols=None*)

Apply the learned objective function to the output of a model.

If needs_fitting is false, this method won't be called

**Parameters** **y_predicted** – the prediction to transform to final prediction

**Returns** predictions

### evalml.objectives.FraudCost.score

FraudCost.**score**(*y_predicted*, *y_true*, *extra_cols=None*)

Calculate score from applying fitted objective to predicted values

If a higher score is better than a lower score, set greater_is_better attribute to True

**Parameters**

- **y_predicted** (*list*) – the predictions from the model. If needs_proba is True, it is the probability estimates
- **y_true** (*list*) – the ground truth for the predictions.
- **extra_cols** (*pd.DataFrame*) – any extra columns that are needed from training data to fit. Only provided if uses_extra_columns is True.

**Returns** score

### evalml.objectives.FraudCost.supports_problem_type

**classmethod** FraudCost.**supports_problem_type**(*problem_type*)

Checks if objective supports given ProblemType

**Parameters** **problem_type** (*str or ProblemType*) – problem type to check

**Returns** whether objective supports ProblemType

**Return type** bool

## evalml.objectives.LeadScoring

**class** evalml.objectives.**LeadScoring**(*true_positives=1*, *false_positives=-1*, *verbose=False*)

Lead scoring

### Methods

| | |
|---|---|
| *__init__* | Create instance. |
| *decision_function* | |
| *fit* | Learn the objective function based on the predictions from a model. |
| *objective_function* | |
| *predict* | Apply the learned objective function to the output of a model. |
| *score* | Calculate score from applying fitted objective to predicted values |
| *supports_problem_type* | Checks if objective supports given ProblemType |

### evalml.objectives.LeadScoring.__init__

LeadScoring.**__init__**(*true_positives=1*, *false_positives=-1*, *verbose=False*)

Create instance.

**Parameters**

- **label** (*int*) – label to optimize threshold for
- **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**(*y_predicted*, *threshold*)

### evalml.objectives.LeadScoring.fit

LeadScoring.**fit**(*y_predicted*, *y_true*, *extra_cols=None*)

Learn the objective function based on the predictions from a model.

If needs_fitting is false, this method won't be called

**Parameters**

- **y_predicted** (*list*) – the predictions from the model. If needs_proba is True, it is the probability estimates
- **y_true** (*list*) – the ground truth for the predictions.
- **extra_cols** (*pd.DataFrame*) – any extra columns that are needed from training data to fit. Only provided if uses_extra_columns is True.

**Returns** self

### evalml.objectives.LeadScoring.objective_function

LeadScoring.**objective_function**(*y_predicted*, *y_true*)

### evalml.objectives.LeadScoring.predict

LeadScoring.**predict**(*y_predicted*, *extra_cols=None*)

Apply the learned objective function to the output of a model.

If needs_fitting is false, this method won't be called

**Parameters** **y_predicted** – the prediction to transform to final prediction

**Returns** predictions

### evalml.objectives.LeadScoring.score

LeadScoring.**score**(*y_predicted*, *y_true*, *extra_cols=None*)

Calculate score from applying fitted objective to predicted values

If a higher score is better than a lower score, set greater_is_better attribute to True

**Parameters**

- **y_predicted** (*list*) – the predictions from the model. If needs_proba is True, it is the probability estimates
- **y_true** (*list*) – the ground truth for the predictions.
- **extra_cols** (*pd.DataFrame*) – any extra columns that are needed from training data to fit. Only provided if uses_extra_columns is True.

**Returns** score

### evalml.objectives.LeadScoring.supports_problem_type

**classmethod** LeadScoring.**supports_problem_type**(*problem_type*)

Checks if objective supports given ProblemType

**Parameters** **problem_type** (*str or ProblemType*) – problem type to check

**Returns** whether objective supports ProblemType

**Return type** bool

## 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 |
| *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 |
| *RecallWeighted* | Recall score for multiclass classification using weighted averaging |
| *AUC* | AUC score for binary classification |

Continued on next page

Table 34 – continued from previous page

| | |
|---|---|
| *AUCMicro* | AUC score for multiclass classification using micro averaging |
| *AUCMacro* | AUC score for multiclass classification using macro averaging |
| *AUCWeighted* | AUC Score for multiclass classification using weighted averaging |
| *LogLoss* | Log Loss for both binary and multiclass classification |
| *MCC* | Matthews correlation coefficient for both binary and multiclass classification |
| *ROC* | Receiver Operating Characteristic score for binary classification. |
| *ConfusionMatrix* | Confusion matrix for classification problems |

### evalml.objectives.F1

**class** evalml.objectives.**F1**(*verbose=False*)

F1 score for binary classification

#### Methods

| | |
|---|---|
| *__init__* | Initialize self. |
| *fit* | Learn the objective function based on the predictions from a model. |
| *predict* | Apply the learned objective function to the output of a model. |
| *score* | Calculate score from applying fitted objective to predicted values |
| *supports_problem_type* | Checks if objective supports given ProblemType |

#### evalml.objectives.F1.__init__

F1.**__init__**(*verbose=False*)

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

#### evalml.objectives.F1.fit

F1.**fit**(*y_predicted*, *y_true*, *extra_cols=None*)

Learn the objective function based on the predictions from a model.

If needs_fitting is false, this method won't be called

**Parameters**

- **y_predicted** (*list*) – the predictions from the model. If needs_proba is True, it is the probability estimates
- **y_true** (*list*) – the ground truth for the predictions.
- **extra_cols** (*pd.DataFrame*) – any extra columns that are needed from training data to fit. Only provided if uses_extra_columns is True.

**Returns** self

### evalml.objectives.F1.predict

F1.**predict**(*y_predicted*, *extra_cols=None*)

Apply the learned objective function to the output of a model.

If needs_fitting is false, this method won't be called

**Parameters** **y_predicted** – the prediction to transform to final prediction

**Returns** predictions

### evalml.objectives.F1.score

F1.**score**(*y_predicted*, *y_true*)

Calculate score from applying fitted objective to predicted values

If a higher score is better than a lower score, set greater_is_better attribute to True

**Parameters**

- **y_predicted** (*list*) – the predictions from the model. If needs_proba is True, it is the probability estimates
- **y_true** (*list*) – the ground truth for the predictions.
- **extra_cols** (*pd.DataFrame*) – any extra columns that are needed from training data to fit. Only provided if uses_extra_columns is True.

**Returns** score

### evalml.objectives.F1.supports_problem_type

**classmethod** F1.**supports_problem_type**(*problem_type*)

Checks if objective supports given ProblemType

**Parameters** **problem_type** (*str or ProblemType*) – problem type to check

**Returns** whether objective supports ProblemType

**Return type** bool

## evalml.objectives.F1Micro

**class** evalml.objectives.**F1Micro**(*verbose=False*)

F1 score for multiclass classification using micro averaging

### Methods

| | |
|---|---|
| *__init__* | Initialize self. |
| *fit* | Learn the objective function based on the predictions from a model. |
| *predict* | Apply the learned objective function to the output of a model. |

Continued on next page

Table 36 – continued from previous page

| | |
|---|---|
| *score* | Calculate score from applying fitted objective to predicted values |
| *supports_problem_type* | Checks if objective supports given ProblemType |

### evalml.objectives.F1Micro.__init__

F1Micro.**__init__**(*verbose=False*)

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

### evalml.objectives.F1Micro.fit

F1Micro.**fit**(*y_predicted*, *y_true*, *extra_cols=None*)

Learn the objective function based on the predictions from a model.

If needs_fitting is false, this method won't be called

**Parameters**

- **y_predicted** (*list*) – the predictions from the model. If needs_proba is True, it is the probability estimates
- **y_true** (*list*) – the ground truth for the predictions.
- **extra_cols** (*pd.DataFrame*) – any extra columns that are needed from training data to fit. Only provided if uses_extra_columns is True.

**Returns** self

### evalml.objectives.F1Micro.predict

F1Micro.**predict**(*y_predicted*, *extra_cols=None*)

Apply the learned objective function to the output of a model.

If needs_fitting is false, this method won't be called

**Parameters** **y_predicted** – the prediction to transform to final prediction

**Returns** predictions

### evalml.objectives.F1Micro.score

F1Micro.**score**(*y_predicted*, *y_true*)

Calculate score from applying fitted objective to predicted values

If a higher score is better than a lower score, set greater_is_better attribute to True

**Parameters**

- **y_predicted** (*list*) – the predictions from the model. If needs_proba is True, it is the probability estimates
- **y_true** (*list*) – the ground truth for the predictions.
- **extra_cols** (*pd.DataFrame*) – any extra columns that are needed from training data to fit. Only provided if uses_extra_columns is True.

**Returns** score

#### evalml.objectives.F1Micro.supports_problem_type

**classmethod** F1Micro.**supports_problem_type**(*problem_type*)

Checks if objective supports given ProblemType

**Parameters** **problem_type** (*str or ProblemType*) – problem type to check

**Returns** whether objective supports ProblemType

**Return type** bool

### evalml.objectives.F1Macro

**class** evalml.objectives.**F1Macro**(*verbose=False*)

F1 score for multiclass classification using macro averaging

#### Methods

| | |
|---|---|
| *__init__* | Initialize self. |
| *fit* | Learn the objective function based on the predictions from a model. |
| *predict* | Apply the learned objective function to the output of a model. |
| *score* | Calculate score from applying fitted objective to predicted values |
| *supports_problem_type* | Checks if objective supports given ProblemType |

#### evalml.objectives.F1Macro.__init__

F1Macro.**__init__**(*verbose=False*)

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

#### evalml.objectives.F1Macro.fit

F1Macro.**fit**(*y_predicted*, *y_true*, *extra_cols=None*)

Learn the objective function based on the predictions from a model.

If needs_fitting is false, this method won't be called

**Parameters**

- **y_predicted** (*list*) – the predictions from the model. If needs_proba is True, it is the probability estimates
- **y_true** (*list*) – the ground truth for the predictions.
- **extra_cols** (*pd.DataFrame*) – any extra columns that are needed from training data to fit. Only provided if uses_extra_columns is True.

**Returns** self

#### evalml.objectives.F1Macro.predict

F1Macro.**predict**(*y_predicted*, *extra_cols=None*)

Apply the learned objective function to the output of a model.

If needs_fitting is false, this method won't be called

**Parameters** **y_predicted** – the prediction to transform to final prediction

**Returns** predictions

#### evalml.objectives.F1Macro.score

F1Macro.**score**(*y_predicted*, *y_true*)

Calculate score from applying fitted objective to predicted values

If a higher score is better than a lower score, set greater_is_better attribute to True

**Parameters**

- **y_predicted** (*list*) – the predictions from the model. If needs_proba is True, it is the probability estimates
- **y_true** (*list*) – the ground truth for the predictions.
- **extra_cols** (*pd.DataFrame*) – any extra columns that are needed from training data to fit. Only provided if uses_extra_columns is True.

**Returns** score

#### evalml.objectives.F1Macro.supports_problem_type

**classmethod** F1Macro.**supports_problem_type**(*problem_type*)

Checks if objective supports given ProblemType

**Parameters** **problem_type** (*str or ProblemType*) – problem type to check

**Returns** whether objective supports ProblemType

**Return type** bool

### evalml.objectives.F1Weighted

**class** evalml.objectives.**F1Weighted**(*verbose=False*)

F1 score for multiclass classification using weighted averaging

#### Methods

| | |
|---|---|
| *__init__* | Initialize self. |
| *fit* | Learn the objective function based on the predictions from a model. |
| *predict* | Apply the learned objective function to the output of a model. |

Continued on next page

Table 38 – continued from previous page

| | |
|---|---|
| *score* | Calculate score from applying fitted objective to predicted values |
| *supports_problem_type* | Checks if objective supports given ProblemType |

### evalml.objectives.F1Weighted.__init__

F1Weighted.**__init__**(*verbose=False*)

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

### evalml.objectives.F1Weighted.fit

F1Weighted.**fit**(*y_predicted*, *y_true*, *extra_cols=None*)

Learn the objective function based on the predictions from a model.

If needs_fitting is false, this method won't be called

**Parameters**

- **y_predicted** (*list*) – the predictions from the model. If needs_proba is True, it is the probability estimates
- **y_true** (*list*) – the ground truth for the predictions.
- **extra_cols** (*pd.DataFrame*) – any extra columns that are needed from training data to fit. Only provided if uses_extra_columns is True.

**Returns** self

### evalml.objectives.F1Weighted.predict

F1Weighted.**predict**(*y_predicted*, *extra_cols=None*)

Apply the learned objective function to the output of a model.

If needs_fitting is false, this method won't be called

**Parameters** **y_predicted** – the prediction to transform to final prediction

**Returns** predictions

### evalml.objectives.F1Weighted.score

F1Weighted.**score**(*y_predicted*, *y_true*)

Calculate score from applying fitted objective to predicted values

If a higher score is better than a lower score, set greater_is_better attribute to True

**Parameters**

- **y_predicted** (*list*) – the predictions from the model. If needs_proba is True, it is the probability estimates
- **y_true** (*list*) – the ground truth for the predictions.
- **extra_cols** (*pd.DataFrame*) – any extra columns that are needed from training data to fit. Only provided if uses_extra_columns is True.

**Returns** score

#### evalml.objectives.F1Weighted.supports_problem_type

**classmethod** F1Weighted.**supports_problem_type**(*problem_type*)
Checks if objective supports given ProblemType

**Parameters** **problem_type** (*str or ProblemType*) – problem type to check

**Returns** whether objective supports ProblemType

**Return type** bool

### evalml.objectives.Precision

**class** evalml.objectives.**Precision**(*verbose=False*)
Precision score for binary classification

#### Methods

| | |
|---|---|
| *__init__* | Initialize self. |
| *fit* | Learn the objective function based on the predictions from a model. |
| *predict* | Apply the learned objective function to the output of a model. |
| *score* | Calculate score from applying fitted objective to predicted values |
| *supports_problem_type* | Checks if objective supports given ProblemType |

#### evalml.objectives.Precision.__init__

Precision.**__init__**(*verbose=False*)
Initialize self. See help(type(self)) for accurate signature.

#### evalml.objectives.Precision.fit

Precision.**fit**(*y_predicted*, *y_true*, *extra_cols=None*)
Learn the objective function based on the predictions from a model.

If needs_fitting is false, this method won't be called

**Parameters**

- **y_predicted** (*list*) – the predictions from the model. If needs_proba is True, it is the probability estimates
- **y_true** (*list*) – the ground truth for the predictions.
- **extra_cols** (*pd.DataFrame*) – any extra columns that are needed from training data to fit. Only provided if uses_extra_columns is True.

**Returns** self

### evalml.objectives.Precision.predict

Precision.**predict**(*y_predicted*, *extra_cols=None*)

Apply the learned objective function to the output of a model.

If needs_fitting is false, this method won't be called

**Parameters** **y_predicted** – the prediction to transform to final prediction

**Returns** predictions

### evalml.objectives.Precision.score

Precision.**score**(*y_predicted*, *y_true*)

Calculate score from applying fitted objective to predicted values

If a higher score is better than a lower score, set greater_is_better attribute to True

**Parameters**

- **y_predicted** (*list*) – the predictions from the model. If needs_proba is True, it is the probability estimates
- **y_true** (*list*) – the ground truth for the predictions.
- **extra_cols** (*pd.DataFrame*) – any extra columns that are needed from training data to fit. Only provided if uses_extra_columns is True.

**Returns** score

### evalml.objectives.Precision.supports_problem_type

**classmethod** Precision.**supports_problem_type**(*problem_type*)

Checks if objective supports given ProblemType

**Parameters** **problem_type** (*str or ProblemType*) – problem type to check

**Returns** whether objective supports ProblemType

**Return type** bool

## evalml.objectives.PrecisionMicro

**class** evalml.objectives.**PrecisionMicro**(*verbose=False*)

Precision score for multiclass classification using micro averaging

### Methods

| | |
|---|---|
| *__init__* | Initialize self. |
| *fit* | Learn the objective function based on the predictions from a model. |
| *predict* | Apply the learned objective function to the output of a model. |

Continued on next page

Table 40 – continued from previous page

| | |
|---|---|
| *score* | Calculate score from applying fitted objective to predicted values |
| *supports_problem_type* | Checks if objective supports given ProblemType |

### evalml.objectives.PrecisionMicro.__init__

PrecisionMicro.**__init__**(*verbose=False*)

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

### evalml.objectives.PrecisionMicro.fit

PrecisionMicro.**fit**(*y_predicted*, *y_true*, *extra_cols=None*)

Learn the objective function based on the predictions from a model.

If needs_fitting is false, this method won't be called

**Parameters**

- **y_predicted** (*list*) – the predictions from the model. If needs_proba is True, it is the probability estimates
- **y_true** (*list*) – the ground truth for the predictions.
- **extra_cols** (*pd.DataFrame*) – any extra columns that are needed from training data to fit. Only provided if uses_extra_columns is True.

**Returns** self

### evalml.objectives.PrecisionMicro.predict

PrecisionMicro.**predict**(*y_predicted*, *extra_cols=None*)

Apply the learned objective function to the output of a model.

If needs_fitting is false, this method won't be called

**Parameters** **y_predicted** – the prediction to transform to final prediction

**Returns** predictions

### evalml.objectives.PrecisionMicro.score

PrecisionMicro.**score**(*y_predicted*, *y_true*)

Calculate score from applying fitted objective to predicted values

If a higher score is better than a lower score, set greater_is_better attribute to True

**Parameters**

- **y_predicted** (*list*) – the predictions from the model. If needs_proba is True, it is the probability estimates
- **y_true** (*list*) – the ground truth for the predictions.
- **extra_cols** (*pd.DataFrame*) – any extra columns that are needed from training data to fit. Only provided if uses_extra_columns is True.

**Returns** score

#### evalml.objectives.PrecisionMicro.supports_problem_type

**classmethod** PrecisionMicro.**supports_problem_type**(*problem_type*)
Checks if objective supports given ProblemType

**Parameters** **problem_type** (*str or ProblemType*) – problem type to check

**Returns** whether objective supports ProblemType

**Return type** bool

### evalml.objectives.PrecisionMacro

**class** evalml.objectives.**PrecisionMacro**(*verbose=False*)
Precision score for multiclass classification using macro averaging

#### Methods

| | |
|---|---|
| *__init__* | Initialize self. |
| *fit* | Learn the objective function based on the predictions from a model. |
| *predict* | Apply the learned objective function to the output of a model. |
| *score* | Calculate score from applying fitted objective to predicted values |
| *supports_problem_type* | Checks if objective supports given ProblemType |

#### evalml.objectives.PrecisionMacro.__init__

PrecisionMacro.**__init__**(*verbose=False*)
Initialize self. See help(type(self)) for accurate signature.

#### evalml.objectives.PrecisionMacro.fit

PrecisionMacro.**fit**(*y_predicted*, *y_true*, *extra_cols=None*)
Learn the objective function based on the predictions from a model.

If needs_fitting is false, this method won't be called

**Parameters**

- **y_predicted** (*list*) – the predictions from the model. If needs_proba is True, it is the probability estimates
- **y_true** (*list*) – the ground truth for the predictions.
- **extra_cols** (*pd.DataFrame*) – any extra columns that are needed from training data to fit. Only provided if uses_extra_columns is True.

**Returns** self

#### evalml.objectives.PrecisionMacro.predict

PrecisionMacro.**predict**(*y_predicted*, *extra_cols=None*)

Apply the learned objective function to the output of a model.

If needs_fitting is false, this method won't be called

**Parameters** **y_predicted** – the prediction to transform to final prediction

**Returns** predictions

#### evalml.objectives.PrecisionMacro.score

PrecisionMacro.**score**(*y_predicted*, *y_true*)

Calculate score from applying fitted objective to predicted values

If a higher score is better than a lower score, set greater_is_better attribute to True

**Parameters**

- **y_predicted** (*list*) – the predictions from the model. If needs_proba is True, it is the probability estimates
- **y_true** (*list*) – the ground truth for the predictions.
- **extra_cols** (*pd.DataFrame*) – any extra columns that are needed from training data to fit. Only provided if uses_extra_columns is True.

**Returns** score

#### evalml.objectives.PrecisionMacro.supports_problem_type

**classmethod** PrecisionMacro.**supports_problem_type**(*problem_type*)

Checks if objective supports given ProblemType

**Parameters** **problem_type** (*str or ProblemType*) – problem type to check

**Returns** whether objective supports ProblemType

**Return type** bool

### evalml.objectives.PrecisionWeighted

**class** evalml.objectives.**PrecisionWeighted**(*verbose=False*)

Precision score for multiclass classification using weighted averaging

#### Methods

| | |
|---|---|
| *__init__* | Initialize self. |
| *fit* | Learn the objective function based on the predictions from a model. |
| *predict* | Apply the learned objective function to the output of a model. |

Continued on next page

Table 42 – continued from previous page

| | |
|---|---|
| *score* | Calculate score from applying fitted objective to predicted values |
| *supports_problem_type* | Checks if objective supports given ProblemType |

### evalml.objectives.PrecisionWeighted.__init__

PrecisionWeighted.**__init__**(*verbose=False*)

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

### evalml.objectives.PrecisionWeighted.fit

PrecisionWeighted.**fit**(*y_predicted*, *y_true*, *extra_cols=None*)

Learn the objective function based on the predictions from a model.

If needs_fitting is false, this method won't be called

**Parameters**

- **y_predicted** (*list*) – the predictions from the model. If needs_proba is True, it is the probability estimates
- **y_true** (*list*) – the ground truth for the predictions.
- **extra_cols** (*pd.DataFrame*) – any extra columns that are needed from training data to fit. Only provided if uses_extra_columns is True.

**Returns** self

### evalml.objectives.PrecisionWeighted.predict

PrecisionWeighted.**predict**(*y_predicted*, *extra_cols=None*)

Apply the learned objective function to the output of a model.

If needs_fitting is false, this method won't be called

**Parameters** **y_predicted** – the prediction to transform to final prediction

**Returns** predictions

### evalml.objectives.PrecisionWeighted.score

PrecisionWeighted.**score**(*y_predicted*, *y_true*)

Calculate score from applying fitted objective to predicted values

If a higher score is better than a lower score, set greater_is_better attribute to True

**Parameters**

- **y_predicted** (*list*) – the predictions from the model. If needs_proba is True, it is the probability estimates
- **y_true** (*list*) – the ground truth for the predictions.
- **extra_cols** (*pd.DataFrame*) – any extra columns that are needed from training data to fit. Only provided if uses_extra_columns is True.

**Returns** score

#### evalml.objectives.PrecisionWeighted.supports_problem_type

**classmethod** PrecisionWeighted.**supports_problem_type**(*problem_type*)
Checks if objective supports given ProblemType

**Parameters** **problem_type** (*str or ProblemType*) – problem type to check

**Returns** whether objective supports ProblemType

**Return type** bool

### evalml.objectives.Recall

**class** evalml.objectives.**Recall**(*verbose=False*)
Recall score for binary classification

#### Methods

| | |
|---|---|
| *__init__* | Initialize self. |
| *fit* | Learn the objective function based on the predictions from a model. |
| *predict* | Apply the learned objective function to the output of a model. |
| *score* | Calculate score from applying fitted objective to predicted values |
| *supports_problem_type* | Checks if objective supports given ProblemType |

#### evalml.objectives.Recall.__init__

Recall.**__init__**(*verbose=False*)
Initialize self. See help(type(self)) for accurate signature.

#### evalml.objectives.Recall.fit

Recall.**fit**(*y_predicted*, *y_true*, *extra_cols=None*)
Learn the objective function based on the predictions from a model.

If needs_fitting is false, this method won't be called

**Parameters**

- **y_predicted** (*list*) – the predictions from the model. If needs_proba is True, it is the probability estimates
- **y_true** (*list*) – the ground truth for the predictions.
- **extra_cols** (*pd.DataFrame*) – any extra columns that are needed from training data to fit. Only provided if uses_extra_columns is True.

**Returns** self

### evalml.objectives.Recall.predict

Recall.**predict**(*y_predicted*, *extra_cols=None*)

Apply the learned objective function to the output of a model.

If needs_fitting is false, this method won't be called

**Parameters** **y_predicted** – the prediction to transform to final prediction

**Returns** predictions

### evalml.objectives.Recall.score

Recall.**score**(*y_predicted*, *y_true*)

Calculate score from applying fitted objective to predicted values

If a higher score is better than a lower score, set greater_is_better attribute to True

**Parameters**

- **y_predicted** (*list*) – the predictions from the model. If needs_proba is True, it is the probability estimates
- **y_true** (*list*) – the ground truth for the predictions.
- **extra_cols** (*pd.DataFrame*) – any extra columns that are needed from training data to fit. Only provided if uses_extra_columns is True.

**Returns** score

### evalml.objectives.Recall.supports_problem_type

**classmethod** Recall.**supports_problem_type**(*problem_type*)

Checks if objective supports given ProblemType

**Parameters** **problem_type** (*str or ProblemType*) – problem type to check

**Returns** whether objective supports ProblemType

**Return type** bool

## evalml.objectives.RecallMicro

**class** evalml.objectives.**RecallMicro**(*verbose=False*)

Recall score for multiclass classification using micro averaging

### Methods

| | |
|---|---|
| *__init__* | Initialize self. |
| *fit* | Learn the objective function based on the predictions from a model. |
| *predict* | Apply the learned objective function to the output of a model. |

Continued on next page

Table 44 – continued from previous page

| | |
|---|---|
| *score* | Calculate score from applying fitted objective to predicted values |
| *supports_problem_type* | Checks if objective supports given ProblemType |

### evalml.objectives.RecallMicro.__init__

RecallMicro.**__init__**(*verbose=False*)

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

### evalml.objectives.RecallMicro.fit

RecallMicro.**fit**(*y_predicted*, *y_true*, *extra_cols=None*)

Learn the objective function based on the predictions from a model.

If needs_fitting is false, this method won't be called

**Parameters**

- **y_predicted** (*list*) – the predictions from the model. If needs_proba is True, it is the probability estimates
- **y_true** (*list*) – the ground truth for the predictions.
- **extra_cols** (*pd.DataFrame*) – any extra columns that are needed from training data to fit. Only provided if uses_extra_columns is True.

**Returns** self

### evalml.objectives.RecallMicro.predict

RecallMicro.**predict**(*y_predicted*, *extra_cols=None*)

Apply the learned objective function to the output of a model.

If needs_fitting is false, this method won't be called

**Parameters** **y_predicted** – the prediction to transform to final prediction

**Returns** predictions

### evalml.objectives.RecallMicro.score

RecallMicro.**score**(*y_predicted*, *y_true*)

Calculate score from applying fitted objective to predicted values

If a higher score is better than a lower score, set greater_is_better attribute to True

**Parameters**

- **y_predicted** (*list*) – the predictions from the model. If needs_proba is True, it is the probability estimates
- **y_true** (*list*) – the ground truth for the predictions.
- **extra_cols** (*pd.DataFrame*) – any extra columns that are needed from training data to fit. Only provided if uses_extra_columns is True.

**Returns** score

#### evalml.objectives.RecallMicro.supports_problem_type

**classmethod** RecallMicro.**supports_problem_type**(*problem_type*)
Checks if objective supports given ProblemType

**Parameters** **problem_type** (*str or ProblemType*) – problem type to check

**Returns** whether objective supports ProblemType

**Return type** bool

### evalml.objectives.RecallMacro

**class** evalml.objectives.**RecallMacro**(*verbose=False*)
Recall score for multiclass classification using macro averaging

#### Methods

| | |
|---|---|
| *__init__* | Initialize self. |
| *fit* | Learn the objective function based on the predictions from a model. |
| *predict* | Apply the learned objective function to the output of a model. |
| *score* | Calculate score from applying fitted objective to predicted values |
| *supports_problem_type* | Checks if objective supports given ProblemType |

#### evalml.objectives.RecallMacro.__init__

RecallMacro.**__init__**(*verbose=False*)
Initialize self. See help(type(self)) for accurate signature.

#### evalml.objectives.RecallMacro.fit

RecallMacro.**fit**(*y_predicted*, *y_true*, *extra_cols=None*)
Learn the objective function based on the predictions from a model.

If needs_fitting is false, this method won't be called

**Parameters**

- **y_predicted** (*list*) – the predictions from the model. If needs_proba is True, it is the probability estimates
- **y_true** (*list*) – the ground truth for the predictions.
- **extra_cols** (*pd.DataFrame*) – any extra columns that are needed from training data to fit. Only provided if uses_extra_columns is True.

**Returns** self

#### evalml.objectives.RecallMacro.predict

RecallMacro.**predict**(*y_predicted*, *extra_cols=None*)

Apply the learned objective function to the output of a model.

If needs_fitting is false, this method won't be called

**Parameters** **y_predicted** – the prediction to transform to final prediction

**Returns** predictions

#### evalml.objectives.RecallMacro.score

RecallMacro.**score**(*y_predicted*, *y_true*)

Calculate score from applying fitted objective to predicted values

If a higher score is better than a lower score, set greater_is_better attribute to True

**Parameters**

- **y_predicted** (*list*) – the predictions from the model. If needs_proba is True, it is the probability estimates
- **y_true** (*list*) – the ground truth for the predictions.
- **extra_cols** (*pd.DataFrame*) – any extra columns that are needed from training data to fit. Only provided if uses_extra_columns is True.

**Returns** score

#### evalml.objectives.RecallMacro.supports_problem_type

**classmethod** RecallMacro.**supports_problem_type**(*problem_type*)

Checks if objective supports given ProblemType

**Parameters** **problem_type** (*str or ProblemType*) – problem type to check

**Returns** whether objective supports ProblemType

**Return type** bool

### evalml.objectives.RecallWeighted

**class** evalml.objectives.**RecallWeighted**(*verbose=False*)

Recall score for multiclass classification using weighted averaging

#### Methods

| | |
|---|---|
| *__init__* | Initialize self. |
| *fit* | Learn the objective function based on the predictions from a model. |
| *predict* | Apply the learned objective function to the output of a model. |

Continued on next page

Table 46 – continued from previous page

| | |
|---|---|
| *score* | Calculate score from applying fitted objective to predicted values |
| *supports_problem_type* | Checks if objective supports given ProblemType |

### evalml.objectives.RecallWeighted.__init__

RecallWeighted.**__init__**(*verbose=False*)

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

### evalml.objectives.RecallWeighted.fit

RecallWeighted.**fit**(*y_predicted*, *y_true*, *extra_cols=None*)

Learn the objective function based on the predictions from a model.

If needs_fitting is false, this method won't be called

**Parameters**

- **y_predicted** (*list*) – the predictions from the model. If needs_proba is True, it is the probability estimates
- **y_true** (*list*) – the ground truth for the predictions.
- **extra_cols** (*pd.DataFrame*) – any extra columns that are needed from training data to fit. Only provided if uses_extra_columns is True.

**Returns** self

### evalml.objectives.RecallWeighted.predict

RecallWeighted.**predict**(*y_predicted*, *extra_cols=None*)

Apply the learned objective function to the output of a model.

If needs_fitting is false, this method won't be called

**Parameters** **y_predicted** – the prediction to transform to final prediction

**Returns** predictions

### evalml.objectives.RecallWeighted.score

RecallWeighted.**score**(*y_predicted*, *y_true*)

Calculate score from applying fitted objective to predicted values

If a higher score is better than a lower score, set greater_is_better attribute to True

**Parameters**

- **y_predicted** (*list*) – the predictions from the model. If needs_proba is True, it is the probability estimates
- **y_true** (*list*) – the ground truth for the predictions.
- **extra_cols** (*pd.DataFrame*) – any extra columns that are needed from training data to fit. Only provided if uses_extra_columns is True.

**Returns** score

#### evalml.objectives.RecallWeighted.supports_problem_type

**classmethod** RecallWeighted.**supports_problem_type**(*problem_type*)
Checks if objective supports given ProblemType

**Parameters** **problem_type** (*str or ProblemType*) – problem type to check

**Returns** whether objective supports ProblemType

**Return type** bool

### evalml.objectives.AUC

**class** evalml.objectives.**AUC**(*verbose=False*)
AUC score for binary classification

#### Methods

| | |
|---|---|
| *__init__* | Initialize self. |
| *fit* | Learn the objective function based on the predictions from a model. |
| *predict* | Apply the learned objective function to the output of a model. |
| *score* | Calculate score from applying fitted objective to predicted values |
| *supports_problem_type* | Checks if objective supports given ProblemType |

#### evalml.objectives.AUC.__init__

AUC.**__init__**(*verbose=False*)
Initialize self. See help(type(self)) for accurate signature.

#### evalml.objectives.AUC.fit

AUC.**fit**(*y_predicted*, *y_true*, *extra_cols=None*)
Learn the objective function based on the predictions from a model.

If needs_fitting is false, this method won't be called

**Parameters**

- **y_predicted** (*list*) – the predictions from the model. If needs_proba is True, it is the probability estimates
- **y_true** (*list*) – the ground truth for the predictions.
- **extra_cols** (*pd.DataFrame*) – any extra columns that are needed from training data to fit. Only provided if uses_extra_columns is True.

**Returns** self

### evalml.objectives.AUC.predict

AUC.**predict**(*y_predicted*, *extra_cols=None*)

Apply the learned objective function to the output of a model.

If needs_fitting is false, this method won't be called

**Parameters** **y_predicted** – the prediction to transform to final prediction

**Returns** predictions

### evalml.objectives.AUC.score

AUC.**score**(*y_predicted*, *y_true*)

Calculate score from applying fitted objective to predicted values

If a higher score is better than a lower score, set greater_is_better attribute to True

**Parameters**

- **y_predicted** (*list*) – the predictions from the model. If needs_proba is True, it is the probability estimates
- **y_true** (*list*) – the ground truth for the predictions.
- **extra_cols** (*pd.DataFrame*) – any extra columns that are needed from training data to fit. Only provided if uses_extra_columns is True.

**Returns** score

### evalml.objectives.AUC.supports_problem_type

**classmethod** AUC.**supports_problem_type**(*problem_type*)

Checks if objective supports given ProblemType

**Parameters** **problem_type** (*str or ProblemType*) – problem type to check

**Returns** whether objective supports ProblemType

**Return type** bool

## evalml.objectives.AUCMicro

**class** evalml.objectives.**AUCMicro**(*verbose=False*)

AUC score for multiclass classification using micro averaging

### Methods

| | |
|---|---|
| *__init__* | Initialize self. |
| *fit* | Learn the objective function based on the predictions from a model. |
| *predict* | Apply the learned objective function to the output of a model. |

Continued on next page

Table 48 – continued from previous page

| | |
|---|---|
| *score* | Calculate score from applying fitted objective to predicted values |
| *supports_problem_type* | Checks if objective supports given ProblemType |

### evalml.objectives.AUCMicro.__init__

AUCMicro.**__init__**(*verbose=False*)

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

### evalml.objectives.AUCMicro.fit

AUCMicro.**fit**(*y_predicted*, *y_true*, *extra_cols=None*)

Learn the objective function based on the predictions from a model.

If needs_fitting is false, this method won't be called

**Parameters**

- **y_predicted** (*list*) – the predictions from the model. If needs_proba is True, it is the probability estimates
- **y_true** (*list*) – the ground truth for the predictions.
- **extra_cols** (*pd.DataFrame*) – any extra columns that are needed from training data to fit. Only provided if uses_extra_columns is True.

**Returns** self

### evalml.objectives.AUCMicro.predict

AUCMicro.**predict**(*y_predicted*, *extra_cols=None*)

Apply the learned objective function to the output of a model.

If needs_fitting is false, this method won't be called

**Parameters** **y_predicted** – the prediction to transform to final prediction

**Returns** predictions

### evalml.objectives.AUCMicro.score

AUCMicro.**score**(*y_predicted*, *y_true*)

Calculate score from applying fitted objective to predicted values

If a higher score is better than a lower score, set greater_is_better attribute to True

**Parameters**

- **y_predicted** (*list*) – the predictions from the model. If needs_proba is True, it is the probability estimates
- **y_true** (*list*) – the ground truth for the predictions.
- **extra_cols** (*pd.DataFrame*) – any extra columns that are needed from training data to fit. Only provided if uses_extra_columns is True.

**Returns** score

#### evalml.objectives.AUCMicro.supports_problem_type

**classmethod** AUCMicro.**supports_problem_type**(*problem_type*)
Checks if objective supports given ProblemType

**Parameters** **problem_type** (*str or ProblemType*) – problem type to check

**Returns** whether objective supports ProblemType

**Return type** bool

### evalml.objectives.AUCMacro

**class** evalml.objectives.**AUCMacro**(*verbose=False*)
AUC score for multiclass classification using macro averaging

#### Methods

| | |
|---|---|
| *__init__* | Initialize self. |
| *fit* | Learn the objective function based on the predictions from a model. |
| *predict* | Apply the learned objective function to the output of a model. |
| *score* | Calculate score from applying fitted objective to predicted values |
| *supports_problem_type* | Checks if objective supports given ProblemType |

#### evalml.objectives.AUCMacro.__init__

AUCMacro.**__init__**(*verbose=False*)
Initialize self. See help(type(self)) for accurate signature.

#### evalml.objectives.AUCMacro.fit

AUCMacro.**fit**(*y_predicted*, *y_true*, *extra_cols=None*)
Learn the objective function based on the predictions from a model.

If needs_fitting is false, this method won't be called

**Parameters**

- **y_predicted** (*list*) – the predictions from the model. If needs_proba is True, it is the probability estimates
- **y_true** (*list*) – the ground truth for the predictions.
- **extra_cols** (*pd.DataFrame*) – any extra columns that are needed from training data to fit. Only provided if uses_extra_columns is True.

**Returns** self

### evalml.objectives.AUCMacro.predict

AUCMacro.**predict**(*y_predicted*, *extra_cols=None*)

Apply the learned objective function to the output of a model.

If needs_fitting is false, this method won't be called

**Parameters** **y_predicted** – the prediction to transform to final prediction

**Returns** predictions

### evalml.objectives.AUCMacro.score

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

Calculate score from applying fitted objective to predicted values

If a higher score is better than a lower score, set greater_is_better attribute to True

**Parameters**

- **y_predicted** (*list*) – the predictions from the model. If needs_proba is True, it is the probability estimates
- **y_true** (*list*) – the ground truth for the predictions.
- **extra_cols** (*pd.DataFrame*) – any extra columns that are needed from training data to fit. Only provided if uses_extra_columns is True.

**Returns** score

### evalml.objectives.AUCMacro.supports_problem_type

**classmethod** AUCMacro.**supports_problem_type**(*problem_type*)

Checks if objective supports given ProblemType

**Parameters** **problem_type** (*str or ProblemType*) – problem type to check

**Returns** whether objective supports ProblemType

**Return type** bool

## evalml.objectives.AUCWeighted

**class** evalml.objectives.**AUCWeighted**(*verbose=False*)

AUC Score for multiclass classification using weighted averaging

### Methods

| | |
|---|---|
| *__init__* | Initialize self. |
| *fit* | Learn the objective function based on the predictions from a model. |
| *predict* | Apply the learned objective function to the output of a model. |

Continued on next page

Table 50 – continued from previous page

| | |
|---|---|
| *score* | Calculate score from applying fitted objective to predicted values |
| *supports_problem_type* | Checks if objective supports given ProblemType |

### evalml.objectives.AUCWeighted.__init__

AUCWeighted.**__init__**(*verbose=False*)

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

### evalml.objectives.AUCWeighted.fit

AUCWeighted.**fit**(*y_predicted*, *y_true*, *extra_cols=None*)

Learn the objective function based on the predictions from a model.

If needs_fitting is false, this method won't be called

**Parameters**

- **y_predicted** (*list*) – the predictions from the model. If needs_proba is True, it is the probability estimates
- **y_true** (*list*) – the ground truth for the predictions.
- **extra_cols** (*pd.DataFrame*) – any extra columns that are needed from training data to fit. Only provided if uses_extra_columns is True.

**Returns** self

### evalml.objectives.AUCWeighted.predict

AUCWeighted.**predict**(*y_predicted*, *extra_cols=None*)

Apply the learned objective function to the output of a model.

If needs_fitting is false, this method won't be called

**Parameters** **y_predicted** – the prediction to transform to final prediction

**Returns** predictions

### evalml.objectives.AUCWeighted.score

AUCWeighted.**score**(*y_predicted*, *y_true*)

Calculate score from applying fitted objective to predicted values

If a higher score is better than a lower score, set greater_is_better attribute to True

**Parameters**

- **y_predicted** (*list*) – the predictions from the model. If needs_proba is True, it is the probability estimates
- **y_true** (*list*) – the ground truth for the predictions.
- **extra_cols** (*pd.DataFrame*) – any extra columns that are needed from training data to fit. Only provided if uses_extra_columns is True.

**Returns** score

#### evalml.objectives.AUCWeighted.supports_problem_type

**classmethod** AUCWeighted.**supports_problem_type**(*problem_type*)
Checks if objective supports given ProblemType

**Parameters** **problem_type** (*str or ProblemType*) – problem type to check

**Returns** whether objective supports ProblemType

**Return type** bool

### evalml.objectives.LogLoss

**class** evalml.objectives.**LogLoss**(*verbose=False*)
Log Loss for both binary and multiclass classification

#### Methods

| | |
|---|---|
| *__init__* | Initialize self. |
| *fit* | Learn the objective function based on the predictions from a model. |
| *predict* | Apply the learned objective function to the output of a model. |
| *score* | Calculate score from applying fitted objective to predicted values |
| *supports_problem_type* | Checks if objective supports given ProblemType |

#### evalml.objectives.LogLoss.__init__

LogLoss.**__init__**(*verbose=False*)
Initialize self. See help(type(self)) for accurate signature.

#### evalml.objectives.LogLoss.fit

LogLoss.**fit**(*y_predicted*, *y_true*, *extra_cols=None*)
Learn the objective function based on the predictions from a model.

If needs_fitting is false, this method won't be called

**Parameters**

- **y_predicted** (*list*) – the predictions from the model. If needs_proba is True, it is the probability estimates
- **y_true** (*list*) – the ground truth for the predictions.
- **extra_cols** (*pd.DataFrame*) – any extra columns that are needed from training data to fit. Only provided if uses_extra_columns is True.

**Returns** self

### evalml.objectives.LogLoss.predict

LogLoss.**predict**(*y_predicted*, *extra_cols=None*)

Apply the learned objective function to the output of a model.

If needs_fitting is false, this method won't be called

**Parameters** **y_predicted** – the prediction to transform to final prediction

**Returns** predictions

### evalml.objectives.LogLoss.score

LogLoss.**score**(*y_predicted*, *y_true*)

Calculate score from applying fitted objective to predicted values

If a higher score is better than a lower score, set greater_is_better attribute to True

**Parameters**

- **y_predicted** (*list*) – the predictions from the model. If needs_proba is True, it is the probability estimates
- **y_true** (*list*) – the ground truth for the predictions.
- **extra_cols** (*pd.DataFrame*) – any extra columns that are needed from training data to fit. Only provided if uses_extra_columns is True.

**Returns** score

### evalml.objectives.LogLoss.supports_problem_type

**classmethod** LogLoss.**supports_problem_type**(*problem_type*)

Checks if objective supports given ProblemType

**Parameters** **problem_type** (*str or ProblemType*) – problem type to check

**Returns** whether objective supports ProblemType

**Return type** bool

## evalml.objectives.MCC

**class** evalml.objectives.**MCC**(*verbose=False*)

Matthews correlation coefficient for both binary and multiclass classification

### Methods

| | |
|---|---|
| *__init__* | Initialize self. |
| *fit* | Learn the objective function based on the predictions from a model. |
| *predict* | Apply the learned objective function to the output of a model. |

Continued on next page

Table 52 – continued from previous page

| | |
|---|---|
| *score* | Calculate score from applying fitted objective to predicted values |
| *supports_problem_type* | Checks if objective supports given ProblemType |

### evalml.objectives.MCC.__init__

MCC.**__init__**(*verbose=False*)
Initialize self. See help(type(self)) for accurate signature.

### evalml.objectives.MCC.fit

MCC.**fit**(*y_predicted*, *y_true*, *extra_cols=None*)
Learn the objective function based on the predictions from a model.

If needs_fitting is false, this method won't be called

**Parameters**

- **y_predicted** (*list*) – the predictions from the model. If needs_proba is True, it is the probability estimates
- **y_true** (*list*) – the ground truth for the predictions.
- **extra_cols** (*pd.DataFrame*) – any extra columns that are needed from training data to fit. Only provided if uses_extra_columns is True.

**Returns** self

### evalml.objectives.MCC.predict

MCC.**predict**(*y_predicted*, *extra_cols=None*)
Apply the learned objective function to the output of a model.

If needs_fitting is false, this method won't be called

**Parameters** **y_predicted** – the prediction to transform to final prediction

**Returns** predictions

### evalml.objectives.MCC.score

MCC.**score**(*y_predicted*, *y_true*)
Calculate score from applying fitted objective to predicted values

If a higher score is better than a lower score, set greater_is_better attribute to True

**Parameters**

- **y_predicted** (*list*) – the predictions from the model. If needs_proba is True, it is the probability estimates
- **y_true** (*list*) – the ground truth for the predictions.
- **extra_cols** (*pd.DataFrame*) – any extra columns that are needed from training data to fit. Only provided if uses_extra_columns is True.

**Returns** score

#### evalml.objectives.MCC.supports_problem_type

**classmethod** MCC.**supports_problem_type**(*problem_type*)
Checks if objective supports given ProblemType

**Parameters** **problem_type** (*str or ProblemType*) – problem type to check

**Returns** whether objective supports ProblemType

**Return type** bool

### evalml.objectives.ROC

**class** evalml.objectives.**ROC**(*verbose=False*)
Receiver Operating Characteristic score for binary classification.

#### Methods

| | |
|---|---|
| *__init__* | Initialize self. |
| *fit* | Learn the objective function based on the predictions from a model. |
| *predict* | Apply the learned objective function to the output of a model. |
| *score* | Calculate score from applying fitted objective to predicted values |
| *supports_problem_type* | Checks if objective supports given ProblemType |

#### evalml.objectives.ROC.__init__

ROC.**__init__**(*verbose=False*)
Initialize self. See help(type(self)) for accurate signature.

#### evalml.objectives.ROC.fit

ROC.**fit**(*y_predicted*, *y_true*, *extra_cols=None*)
Learn the objective function based on the predictions from a model.

If needs_fitting is false, this method won't be called

**Parameters**

- **y_predicted** (*list*) – the predictions from the model. If needs_proba is True, it is the probability estimates
- **y_true** (*list*) – the ground truth for the predictions.
- **extra_cols** (*pd.DataFrame*) – any extra columns that are needed from training data to fit. Only provided if uses_extra_columns is True.

**Returns** self

### evalml.objectives.ROC.predict

ROC.**predict**(*y_predicted*, *extra_cols=None*)

Apply the learned objective function to the output of a model.

If needs_fitting is false, this method won't be called

**Parameters** **y_predicted** – the prediction to transform to final prediction

**Returns** predictions

### evalml.objectives.ROC.score

ROC.**score**(*y_predicted*, *y_true*)

Calculate score from applying fitted objective to predicted values

If a higher score is better than a lower score, set greater_is_better attribute to True

**Parameters**

- **y_predicted** (*list*) – the predictions from the model. If needs_proba is True, it is the probability estimates
- **y_true** (*list*) – the ground truth for the predictions.
- **extra_cols** (*pd.DataFrame*) – any extra columns that are needed from training data to fit. Only provided if uses_extra_columns is True.

**Returns** score

### evalml.objectives.ROC.supports_problem_type

**classmethod** ROC.**supports_problem_type**(*problem_type*)

Checks if objective supports given ProblemType

**Parameters** **problem_type** (*str or ProblemType*) – problem type to check

**Returns** whether objective supports ProblemType

**Return type** bool

## evalml.objectives.ConfusionMatrix

**class** evalml.objectives.**ConfusionMatrix**(*verbose=False*)

Confusion matrix for classification problems

### Methods

| | |
|---|---|
| *__init__* | Initialize self. |
| *fit* | Learn the objective function based on the predictions from a model. |
| *predict* | Apply the learned objective function to the output of a model. |

Continued on next page

Table 54 – continued from previous page

| | |
|---|---|
| *score* | Calculate score from applying fitted objective to predicted values |
| *supports_problem_type* | Checks if objective supports given ProblemType |

### evalml.objectives.ConfusionMatrix.__init__

ConfusionMatrix.**__init__**(*verbose=False*)

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

### evalml.objectives.ConfusionMatrix.fit

ConfusionMatrix.**fit**(*y_predicted*, *y_true*, *extra_cols=None*)

Learn the objective function based on the predictions from a model.

If needs_fitting is false, this method won't be called

**Parameters**

- **y_predicted** (*list*) – the predictions from the model. If needs_proba is True, it is the probability estimates
- **y_true** (*list*) – the ground truth for the predictions.
- **extra_cols** (*pd.DataFrame*) – any extra columns that are needed from training data to fit. Only provided if uses_extra_columns is True.

**Returns** self

### evalml.objectives.ConfusionMatrix.predict

ConfusionMatrix.**predict**(*y_predicted*, *extra_cols=None*)

Apply the learned objective function to the output of a model.

If needs_fitting is false, this method won't be called

**Parameters** **y_predicted** – the prediction to transform to final prediction

**Returns** predictions

### evalml.objectives.ConfusionMatrix.score

ConfusionMatrix.**score**(*y_predicted*, *y_true*)

Calculate score from applying fitted objective to predicted values

If a higher score is better than a lower score, set greater_is_better attribute to True

**Parameters**

- **y_predicted** (*list*) – the predictions from the model. If needs_proba is True, it is the probability estimates
- **y_true** (*list*) – the ground truth for the predictions.
- **extra_cols** (*pd.DataFrame*) – any extra columns that are needed from training data to fit. Only provided if uses_extra_columns is True.

**Returns** score

#### evalml.objectives.ConfusionMatrix.supports_problem_type

**classmethod** ConfusionMatrix.**supports_problem_type**(*problem_type*)
Checks if objective supports given ProblemType

**Parameters** **problem_type** (*str or ProblemType*) – problem type to check

**Returns** whether objective supports ProblemType

**Return type** bool

### Regression

| | |
|---|---|
| *R2* | Coefficient of determination for regression |
| *MAE* | Mean absolute error for regression |
| *MSE* | Mean squared error for regression |
| *MSLE* | Mean squared log error for regression |
| *MedianAE* | Median absolute error for regression |
| *MaxError* | Maximum residual error for regression |
| *ExpVariance* | Explained variance score for regression |

#### evalml.objectives.R2

**class** evalml.objectives.**R2**(*verbose=False*)
Coefficient of determination for regression

#### Methods

| | |
|---|---|
| *__init__* | Initialize self. |
| *fit* | Learn the objective function based on the predictions from a model. |
| *predict* | Apply the learned objective function to the output of a model. |
| *score* | Calculate score from applying fitted objective to predicted values |
| *supports_problem_type* | Checks if objective supports given ProblemType |

#### evalml.objectives.R2.__init__

R2.**__init__**(*verbose=False*)
Initialize self. See help(type(self)) for accurate signature.

#### evalml.objectives.R2.fit

R2.**fit**(*y_predicted*, *y_true*, *extra_cols=None*)
Learn the objective function based on the predictions from a model.

If needs_fitting is false, this method won't be called

**Parameters**

- **y_predicted** (*list*) – the predictions from the model. If needs_proba is True, it is the probability estimates
- **y_true** (*list*) – the ground truth for the predictions.
- **extra_cols** (*pd.DataFrame*) – any extra columns that are needed from training data to fit. Only provided if uses_extra_columns is True.

**Returns** self

### evalml.objectives.R2.predict

R2.**predict**(*y_predicted*, *extra_cols=None*)

Apply the learned objective function to the output of a model.

If needs_fitting is false, this method won't be called

**Parameters** **y_predicted** – the prediction to transform to final prediction

**Returns** predictions

### evalml.objectives.R2.score

R2.**score**(*y_predicted*, *y_true*)

Calculate score from applying fitted objective to predicted values

If a higher score is better than a lower score, set greater_is_better attribute to True

**Parameters**

- **y_predicted** (*list*) – the predictions from the model. If needs_proba is True, it is the probability estimates
- **y_true** (*list*) – the ground truth for the predictions.
- **extra_cols** (*pd.DataFrame*) – any extra columns that are needed from training data to fit. Only provided if uses_extra_columns is True.

**Returns** score

### evalml.objectives.R2.supports_problem_type

**classmethod** R2.**supports_problem_type**(*problem_type*)

Checks if objective supports given ProblemType

**Parameters** **problem_type** (*str or ProblemType*) – problem type to check

**Returns** whether objective supports ProblemType

**Return type** bool

## evalml.objectives.MAE

**class** evalml.objectives.**MAE**(*verbose=False*)

Mean absolute error for regression

### Methods

| | |
|---|---|
| *__init__* | Initialize self. |
| *fit* | Learn the objective function based on the predictions from a model. |
| *predict* | Apply the learned objective function to the output of a model. |
| *score* | Calculate score from applying fitted objective to predicted values |
| *supports_problem_type* | Checks if objective supports given ProblemType |

#### evalml.objectives.MAE.__init__

MAE.**__init__**(*verbose=False*)
Initialize self. See help(type(self)) for accurate signature.

#### evalml.objectives.MAE.fit

MAE.**fit**(*y_predicted*, *y_true*, *extra_cols=None*)
Learn the objective function based on the predictions from a model.

If needs_fitting is false, this method won't be called

**Parameters**

- **y_predicted** (*list*) – the predictions from the model. If needs_proba is True, it is the probability estimates
- **y_true** (*list*) – the ground truth for the predictions.
- **extra_cols** (*pd.DataFrame*) – any extra columns that are needed from training data to fit. Only provided if uses_extra_columns is True.

**Returns** self

#### evalml.objectives.MAE.predict

MAE.**predict**(*y_predicted*, *extra_cols=None*)
Apply the learned objective function to the output of a model.

If needs_fitting is false, this method won't be called

**Parameters** **y_predicted** – the prediction to transform to final prediction

**Returns** predictions

#### evalml.objectives.MAE.score

MAE.**score**(*y_predicted*, *y_true*)
Calculate score from applying fitted objective to predicted values

If a higher score is better than a lower score, set greater_is_better attribute to True

**Parameters**

- **y_predicted** (*list*) – the predictions from the model. If needs_proba is True, it is the probability estimates
- **y_true** (*list*) – the ground truth for the predictions.
- **extra_cols** (*pd.DataFrame*) – any extra columns that are needed from training data to fit. Only provided if uses_extra_columns is True.

**Returns** score

#### evalml.objectives.MAE.supports_problem_type

**classmethod** MAE.**supports_problem_type**(*problem_type*)

Checks if objective supports given ProblemType

**Parameters** **problem_type** (*str or ProblemType*) – problem type to check

**Returns** whether objective supports ProblemType

**Return type** bool

### evalml.objectives.MSE

**class** evalml.objectives.**MSE**(*verbose=False*)

Mean squared error for regression

#### Methods

| | |
|---|---|
| *__init__* | Initialize self. |
| *fit* | Learn the objective function based on the predictions from a model. |
| *predict* | Apply the learned objective function to the output of a model. |
| *score* | Calculate score from applying fitted objective to predicted values |
| *supports_problem_type* | Checks if objective supports given ProblemType |

#### evalml.objectives.MSE.__init__

MSE.**__init__**(*verbose=False*)

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

#### evalml.objectives.MSE.fit

MSE.**fit**(*y_predicted*, *y_true*, *extra_cols=None*)

Learn the objective function based on the predictions from a model.

If needs_fitting is false, this method won't be called

**Parameters**

- **y_predicted** (*list*) – the predictions from the model. If needs_proba is True, it is the probability estimates

- **y_true** (*list*) – the ground truth for the predictions.
- **extra_cols** (*pd.DataFrame*) – any extra columns that are needed from training data to fit. Only provided if uses_extra_columns is True.

**Returns** self

#### evalml.objectives.MSE.predict

MSE.**predict**(*y_predicted*, *extra_cols=None*)

Apply the learned objective function to the output of a model.

If needs_fitting is false, this method won't be called

**Parameters** **y_predicted** – the prediction to transform to final prediction

**Returns** predictions

#### evalml.objectives.MSE.score

MSE.**score**(*y_predicted*, *y_true*)

Calculate score from applying fitted objective to predicted values

If a higher score is better than a lower score, set greater_is_better attribute to True

**Parameters**

- **y_predicted** (*list*) – the predictions from the model. If needs_proba is True, it is the probability estimates
- **y_true** (*list*) – the ground truth for the predictions.
- **extra_cols** (*pd.DataFrame*) – any extra columns that are needed from training data to fit. Only provided if uses_extra_columns is True.

**Returns** score

#### evalml.objectives.MSE.supports_problem_type

**classmethod** MSE.**supports_problem_type**(*problem_type*)

Checks if objective supports given ProblemType

**Parameters** **problem_type** (*str or ProblemType*) – problem type to check

**Returns** whether objective supports ProblemType

**Return type** bool

### evalml.objectives.MSLE

**class** evalml.objectives.**MSLE**(*verbose=False*)

Mean squared log error for regression

#### Methods

| | |
|---|---|
| *__init__* | Initialize self. |
| *fit* | Learn the objective function based on the predictions from a model. |
| *predict* | Apply the learned objective function to the output of a model. |
| *score* | Calculate score from applying fitted objective to predicted values |
| *supports_problem_type* | Checks if objective supports given ProblemType |

### evalml.objectives.MSLE.__init__

MSLE.**__init__**(*verbose=False*)

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

### evalml.objectives.MSLE.fit

MSLE.**fit**(*y_predicted*, *y_true*, *extra_cols=None*)

Learn the objective function based on the predictions from a model.

If needs_fitting is false, this method won't be called

**Parameters**

- **y_predicted** (*list*) – the predictions from the model. If needs_proba is True, it is the probability estimates
- **y_true** (*list*) – the ground truth for the predictions.
- **extra_cols** (*pd.DataFrame*) – any extra columns that are needed from training data to fit. Only provided if uses_extra_columns is True.

**Returns** self

### evalml.objectives.MSLE.predict

MSLE.**predict**(*y_predicted*, *extra_cols=None*)

Apply the learned objective function to the output of a model.

If needs_fitting is false, this method won't be called

**Parameters** **y_predicted** – the prediction to transform to final prediction

**Returns** predictions

### evalml.objectives.MSLE.score

MSLE.**score**(*y_predicted*, *y_true*)

Calculate score from applying fitted objective to predicted values

If a higher score is better than a lower score, set greater_is_better attribute to True

**Parameters**

- **y_predicted** (*list*) – the predictions from the model. If needs_proba is True, it is the probability estimates

- **y_true** (*list*) – the ground truth for the predictions.
- **extra_cols** (*pd.DataFrame*) – any extra columns that are needed from training data to fit. Only provided if uses_extra_columns is True.

**Returns** score

#### evalml.objectives.MSLE.supports_problem_type

**classmethod** MSLE.**supports_problem_type**(*problem_type*)

Checks if objective supports given ProblemType

**Parameters** **problem_type** (*str or ProblemType*) – problem type to check

**Returns** whether objective supports ProblemType

**Return type** bool

### evalml.objectives.MedianAE

**class** evalml.objectives.**MedianAE**(*verbose=False*)

Median absolute error for regression

#### Methods

| | |
|---|---|
| *__init__* | Initialize self. |
| *fit* | Learn the objective function based on the predictions from a model. |
| *predict* | Apply the learned objective function to the output of a model. |
| *score* | Calculate score from applying fitted objective to predicted values |
| *supports_problem_type* | Checks if objective supports given ProblemType |

#### evalml.objectives.MedianAE.__init__

MedianAE.**__init__**(*verbose=False*)

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

#### evalml.objectives.MedianAE.fit

MedianAE.**fit**(*y_predicted*, *y_true*, *extra_cols=None*)

Learn the objective function based on the predictions from a model.

If needs_fitting is false, this method won't be called

**Parameters**

- **y_predicted** (*list*) – the predictions from the model. If needs_proba is True, it is the probability estimates
- **y_true** (*list*) – the ground truth for the predictions.

- **extra_cols** (*pd.DataFrame*) – any extra columns that are needed from training data to fit. Only provided if uses_extra_columns is True.

**Returns** self

### evalml.objectives.MedianAE.predict

MedianAE.**predict**(*y_predicted*, *extra_cols=None*)

Apply the learned objective function to the output of a model.

If needs_fitting is false, this method won't be called

**Parameters** **y_predicted** – the prediction to transform to final prediction

**Returns** predictions

### evalml.objectives.MedianAE.score

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

Calculate score from applying fitted objective to predicted values

If a higher score is better than a lower score, set greater_is_better attribute to True

**Parameters**

- **y_predicted** (*list*) – the predictions from the model. If needs_proba is True, it is the probability estimates
- **y_true** (*list*) – the ground truth for the predictions.
- **extra_cols** (*pd.DataFrame*) – any extra columns that are needed from training data to fit. Only provided if uses_extra_columns is True.

**Returns** score

### evalml.objectives.MedianAE.supports_problem_type

**classmethod** MedianAE.**supports_problem_type**(*problem_type*)

Checks if objective supports given ProblemType

**Parameters** **problem_type** (*str or ProblemType*) – problem type to check

**Returns** whether objective supports ProblemType

**Return type** bool

## evalml.objectives.MaxError

**class** evalml.objectives.**MaxError**(*verbose=False*)

Maximum residual error for regression

### Methods

| | |
|---|---|
| *__init__* | Initialize self. |

Continued on next page

Table 61 – continued from previous page

| | |
|---|---|
| *fit* | Learn the objective function based on the predictions from a model. |
| *predict* | Apply the learned objective function to the output of a model. |
| *score* | Calculate score from applying fitted objective to predicted values |
| *supports_problem_type* | Checks if objective supports given ProblemType |

### evalml.objectives.MaxError.__init__

MaxError.**__init__**(*verbose=False*)

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

### evalml.objectives.MaxError.fit

MaxError.**fit**(*y_predicted*, *y_true*, *extra_cols=None*)

Learn the objective function based on the predictions from a model.

If needs_fitting is false, this method won't be called

**Parameters**

- **y_predicted** (*list*) – the predictions from the model. If needs_proba is True, it is the probability estimates
- **y_true** (*list*) – the ground truth for the predictions.
- **extra_cols** (*pd.DataFrame*) – any extra columns that are needed from training data to fit. Only provided if uses_extra_columns is True.

**Returns** self

### evalml.objectives.MaxError.predict

MaxError.**predict**(*y_predicted*, *extra_cols=None*)

Apply the learned objective function to the output of a model.

If needs_fitting is false, this method won't be called

**Parameters** **y_predicted** – the prediction to transform to final prediction

**Returns** predictions

### evalml.objectives.MaxError.score

MaxError.**score**(*y_predicted*, *y_true*)

Calculate score from applying fitted objective to predicted values

If a higher score is better than a lower score, set greater_is_better attribute to True

**Parameters**

- **y_predicted** (*list*) – the predictions from the model. If needs_proba is True, it is the probability estimates
- **y_true** (*list*) – the ground truth for the predictions.

- **extra_cols** (*pd.DataFrame*) – any extra columns that are needed from training data to fit. Only provided if uses_extra_columns is True.

**Returns** score

### evalml.objectives.MaxError.supports_problem_type

**classmethod** MaxError.**supports_problem_type**(*problem_type*)

Checks if objective supports given ProblemType

**Parameters** **problem_type** (*str or ProblemType*) – problem type to check

**Returns** whether objective supports ProblemType

**Return type** bool

## evalml.objectives.ExpVariance

**class** evalml.objectives.**ExpVariance**(*verbose=False*)

Explained variance score for regression

### Methods

| | |
|---|---|
| *__init__* | Initialize self. |
| *fit* | Learn the objective function based on the predictions from a model. |
| *predict* | Apply the learned objective function to the output of a model. |
| *score* | Calculate score from applying fitted objective to predicted values |
| *supports_problem_type* | Checks if objective supports given ProblemType |

### evalml.objectives.ExpVariance.__init__

ExpVariance.**__init__**(*verbose=False*)

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

### evalml.objectives.ExpVariance.fit

ExpVariance.**fit**(*y_predicted*, *y_true*, *extra_cols=None*)

Learn the objective function based on the predictions from a model.

If needs_fitting is false, this method won't be called

**Parameters**

- **y_predicted** (*list*) – the predictions from the model. If needs_proba is True, it is the probability estimates
- **y_true** (*list*) – the ground truth for the predictions.
- **extra_cols** (*pd.DataFrame*) – any extra columns that are needed from training data to fit. Only provided if uses_extra_columns is True.

**Returns** self

### evalml.objectives.ExpVariance.predict

ExpVariance.**predict**(*y_predicted*, *extra_cols=None*)

Apply the learned objective function to the output of a model.

If needs_fitting is false, this method won't be called

**Parameters** **y_predicted** – the prediction to transform to final prediction

**Returns** predictions

### evalml.objectives.ExpVariance.score

ExpVariance.**score**(*y_predicted*, *y_true*)

Calculate score from applying fitted objective to predicted values

If a higher score is better than a lower score, set greater_is_better attribute to True

**Parameters**

- **y_predicted** (*list*) – the predictions from the model. If needs_proba is True, it is the probability estimates
- **y_true** (*list*) – the ground truth for the predictions.
- **extra_cols** (*pd.DataFrame*) – any extra columns that are needed from training data to fit. Only provided if uses_extra_columns is True.

**Returns** score

### evalml.objectives.ExpVariance.supports_problem_type

**classmethod** ExpVariance.**supports_problem_type**(*problem_type*)

Checks if objective supports given ProblemType

**Parameters** **problem_type** (*str or ProblemType*) – problem type to check

**Returns** whether objective supports ProblemType

**Return type** bool

## 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_types** (*str or ProblemTypes*) – problem type that needs to be handled

**Returns** ProblemTypes

## Tuners

| | |
|---|---|
| *Tuner* | Defines API for Tuners |
| *SKOptTuner* | Bayesian Optimizer |

### evalml.tuners.Tuner

**class** evalml.tuners.**Tuner**(*space*, *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__* | Init Tuner |
| *add* | Register a set of hyperparameters with the score obtained from training a pipeline with those hyperparameters. |
| *propose* | Returns a set of hyperparameters to train a pipeline with, based off the search space dimensions and prior samples |

#### evalml.tuners.Tuner.__init__

Tuner.**__init__**(*space*, *random_state=0*)

Init Tuner

**Parameters**

- **space** (*dict*) – search space for hyperparameters
- **random_state** (*int*) – random state

**Returns** self

**Return type** *Tuner*

#### evalml.tuners.Tuner.add

Tuner.**add**(*parameters*, *score*)

Register a set of hyperparameters with the score obtained from training a pipeline with those hyperparameters.

**Parameters**

- **parameters** (*dict*) – hyperparameters
- **score** (*float*) – associated score

**Returns** None

#### evalml.tuners.Tuner.propose

Tuner.**propose**()

Returns a set of hyperparameters to train a pipeline with, based off the search space dimensions and prior samples

**Returns** proposed hyperparameters

**Return type** dict

### evalml.tuners.SKOptTuner

**class** evalml.tuners.**SKOptTuner**(*space*, *random_state=0*)

Bayesian Optimizer

#### Methods

| | |
|---|---|
| *__init__* | Init SkOptTuner |
| *add* | Add score to sample |
| *propose* | Returns hyperparameters based off search space and samples |

#### evalml.tuners.SKOptTuner.__init__

SKOptTuner.**__init__**(*space*, *random_state=0*)

Init SkOptTuner

**Parameters**

- **space** (*dict*) – search space for hyperparameters
- **random_state** (*int*) – random state

**Returns** self

**Return type** SKoptTuner

### evalml.tuners.SKOptTuner.add

SKOptTuner.**add**(*parameters*, *score*)

Add score to sample

**Parameters**

- **parameters** (*dict*) – hyperparameters
- **score** (*float*) – associated score

**Returns** None

### evalml.tuners.SKOptTuner.propose

SKOptTuner.**propose**()

Returns hyperparameters based off search space and samples

**Returns** proposed hyperparameters

**Return type** dict

## Guardrails

| | |
|---|---|
| *detect_highly_null* | Checks if there are any highly-null columns in a dataframe. |
| *detect_label_leakage* | Check if any of the features are highly correlated with the target. |
| *detect_outliers* | Checks if there are any outliers in a dataframe by using first Isolation Forest to obtain the anomaly score of each index and then using IQR to determine score anomalies. |
| *detect_id_columns* | Check if any of the features are ID columns. |

### evalml.guardrails.detect_highly_null

evalml.guardrails.**detect_highly_null**(*X*, *percent_threshold=0.95*)

Checks if there are any highly-null columns in a dataframe.

**Parameters**

- **X** (*pd.DataFrame*) – features
- **percent_threshold** (*float*) – Require that percentage of null values to be considered "highly-null", defaults to .95

**Returns** A dictionary of features with column name or index and their percentage of null values

### Example

```
>>> df = pd.DataFrame({
...     'lots_of_null': [None, None, None, None, 5],
...     'no_null': [1, 2, 3, 4, 5]
... })
>>> detect_highly_null(df, percent_threshold=0.8)
{'lots_of_null': 0.8}
```

### evalml.guardrails.detect_label_leakage

evalml.guardrails.**detect_label_leakage**(*X*, *y*, *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**

- **X** (*pd.DataFrame*) – The input features to check
- **y** (*pd.Series*) – the labels
- **threshold** (*float*) – the correlation threshold to be considered leakage. Defaults to .95

**Returns** leakage, dictionary of features with leakage and corresponding threshold

#### 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])
>>> detect_label_leakage(X, y, threshold=0.8)
{'leak': 0.8827072320669518}
```

### evalml.guardrails.detect_outliers

evalml.guardrails.**detect_outliers**(*X*, *random_state=0*)

Checks if there are any outliers in a dataframe by using first 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

**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]
... })
>>> detect_outliers(df)
[3]
```

### evalml.guardrails.detect_id_columns

evalml.guardrails.**detect_id_columns**(*X*, *threshold=1.0*)

Check if any of the features are 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]
... })
>>> detect_id_columns(df)
{'df_id': 1.0}
```

### 2.6.15 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 *guardrails* 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 are feature importances 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 *guardrails* to combat overfitting. Such guardrails 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

## A



## C



## D



## E



## F

## G

## R



## S

## T



## X