EvalML Documentation

Release 0.14.1

Alteryx Innovation Labs

Sep 29, 2020

CONTENTS

Combined with Featuretools and Compose, EvalML can be used to create end-to-end supervised machine learning solutions.

CHAPTER

ONE

INSTALL

EvalML is available for Python 3.6+. It can be installed with pip or conda.

1.1 Pip with all dependencies

To install evalml with pip, run the following command:

```
pip install evalml
```

1.2 Pip with core dependencies

EvalML includes several optional dependencies. The `xgboost` and `catboost` packages support pipelines built around those modeling libraries. The `plotly` and `ipywidgets` packages support plotting functionality in automl searches. These dependencies are recommended, and are included with EvalML by default but are not required in order to install and use EvalML.

EvalML's core dependencies are listed in `core-requirements.txt` in the source code, and optional requirements are isted in `requirements.txt`.

To install EvalML with only the core required dependencies, download the EvalML source from pypi to access the requirements files. Then run the following:

```
pip install evalml --no-dependencies
pip install -r core-requirements.txt
```

1.3 Conda with all dependencies

To install evalml with conda run the following command:

```
conda install -c conda-forge evalml
```

1.4 Conda with core dependencies

To install evalml with only core dependencies run the following command:

```
conda install -c conda-forge evalml-core
```

1.5 Windows

The XGBoost library may not be pip-installable in some Windows environments. If you are encountering installation issues, please try installing XGBoost from Github before installing EvalML or install evalml with conda.

1.6 Mac

In order to run on Mac, LightGBM requires the `OpenMP` libray to be installed, which can be done with HomeBrew by running

```
brew install libomp
```

CHAPTER

TWO

START

In this guide, we'll show how you can use EvalML to automatically find the best pipeline for predicting whether a patient has breast cancer. Along the way, we'll highlight EvalML's built-in tools and features for understanding and interacting with the search process.

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

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

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

EvalML has many options to configure the pipeline search. At the minimum, we need to define an objective function. For simplicity, we will use the F1 score in this example. However, the real power of EvalML is in using domain-specific *objective functions* or *building your own*.

Below EvalML utilizes Bayesian optimization (EvalML's default optimizer) to search and find the best pipeline defined by the given objective.

EvalML provides a number of parameters to control the search process. `max_iterations` is one of the parameters which controls the stopping criterion for the AutoML search. It indicates the maximum number of pipelines to train and evaluate. In this example, `max_iterations` is set to 5.

** Graphing methods, like AutoMLSearch, on Jupyter Notebook and Jupyter Lab require ipywidgets to be installed.

** If graphing on Jupyter Lab, jupyterlab-plotly required. To download this, make sure you have npm installed.

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

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

[4]:
```
X_train, X_holdout, y_train, y_holdout = evalml.preprocessing.split_data(X, y, test_
↪size=.2)
```

When we call `search()`, the search for the best pipeline will begin. There is no need to wrangle with missing data or categorical variables as EvalML includes various preprocessing steps (like imputation, one-hot encoding, feature selection) to ensure you're getting the best results. As long as your data is in a single table, EvalML can handle it. If not, you can reduce your data to a single table by utilizing Featuretools and its Entity Sets.

You can find more information on pipeline components and how to integrate your own custom pipelines into EvalML *here*.

[5]:
```
automl.search(X_train, y_train)
```

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

Optimizing for F1.
Greater score is better.

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

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

```
(1/5) Mode Baseline Binary Classification P... Elapsed:00:00
        Starting cross validation
        Finished cross validation - mean F1: 0.000
(2/5) LightGBM Classifier w/ Imputer          Elapsed:00:00
        Starting cross validation
        Finished cross validation - mean F1: 0.944
(3/5) Extra Trees Classifier w/ Imputer       Elapsed:00:00
        Starting cross validation
        Finished cross validation - mean F1: 0.945
(4/5) Elastic Net Classifier w/ Imputer + S... Elapsed:00:01
        Starting cross validation
        Finished cross validation - mean F1: 0.632
(5/5) CatBoost Classifier w/ Imputer          Elapsed:00:02
        Starting cross validation
        Finished cross validation - mean F1: 0.946

Search finished after 00:02
Best pipeline: CatBoost Classifier w/ Imputer
Best pipeline F1: 0.945968
```

After the search is finished we can view all of the pipelines searched, ranked by score. Internally, EvalML performs cross validation to score the pipelines. If it notices a high variance across cross validation folds, it will warn you. EvalML also provides additional *data checks* to analyze your data to assist you in producing the best performing pipeline.

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

[6]:

	id	pipeline_name	score	validation_score	percent_better_than_baseline	high_variance_cv
0	4	CatBoost Classifier w/ Imputer	0.945968	0.929825	NaN	False
1	2	Extra Trees Classifier w/ Imputer	0.944722	0.946429	NaN	False
2	1	LightGBM Classifier w/ Imputer	0.943624	0.931034	NaN	False
3	3	Elastic Net Classifier w/ Imputer + Standard S...	0.632259	0.480000	NaN	True
4	0	Mode Baseline Binary Classification Pipeline	0.000000	0.000000	NaN	False

(continues on next page)

(continued from previous page)

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

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

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

```
*********************************************************
* Elastic Net Classifier w/ Imputer + Standard Scaler *
*********************************************************

Problem Type: binary
Model Family: Linear

Pipeline Steps
==============
1. Imputer
         * categorical_impute_strategy : most_frequent
         * numeric_impute_strategy : mean
         * categorical_fill_value : None
         * numeric_fill_value : None
2. Standard Scaler
3. Elastic Net Classifier
         * alpha : 0.5
         * l1_ratio : 0.5
         * n_jobs : -1
         * max_iter : 1000
         * penalty : elasticnet
         * loss : log

Training
========
Training for binary problems.
Total training time (including CV): 0.2 seconds

Cross Validation
----------
High variance within cross validation scores. Model may not perform as estimated on␣
↪unseen data.
              F1  MCC Binary  Log Loss Binary   AUC  Precision  Balanced Accuracy␣
↪Binary  Accuracy Binary # Training # Testing
0           0.480       0.473            0.518 0.986      1.000                    0.
↪658            0.743    303.000   152.000
1           0.719       0.667            0.479 0.988      1.000                    0.
↪781            0.836    303.000   152.000
2           0.698       0.649            0.490 0.985      1.000                    0.
↪768            0.828    304.000   151.000
mean        0.632       0.596            0.496 0.986      1.000                    0.
↪735            0.802          -         -
std         0.132       0.107            0.020 0.002      0.000                    0.
↪068            0.051          -         -
```

(continues on next page)

(continued from previous page)

```
coef of var 0.209       0.179            0.040 0.002      0.000                    0.
↪092            0.064          -          -
```

We can also view the pipeline parameters directly:

[8]:
```
pipeline = automl.get_pipeline(3)
print(pipeline.parameters)
```

```
{'Imputer': {'categorical_impute_strategy': 'most_frequent',␣
↪'numeric_impute_strategy': 'mean', 'categorical_fill_value': None,␣
↪'numeric_fill_value': None}, 'Elastic Net Classifier': {'alpha': 0.5, 'l1_ratio': 0.
↪5, 'n_jobs': -1, 'max_iter': 1000, 'penalty': 'elasticnet', 'loss': 'log'}}
```

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

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

[9]:
```
OrderedDict([('F1', 0.9397590361445782)])
```

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

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

[10]:

CHAPTER

THREE

TUTORIALS

Below are examples of how to apply EvalML to a variety of problems:

3.1 Building a Fraud Prediction Model with EvalML

In this demo, we will build an optimized fraud prediction model using EvalML. To optimize the pipeline, we will set up an objective function to minimize the percentage of total transaction value lost to fraud. At the end of this demo, we also show you how introducing the right objective during the training is over 4x better than using a generic machine learning metric like AUC.

[1]:
```
import evalml
from evalml import AutoMLSearch
from evalml.objectives import FraudCost
```

3.1.1 Configure "Cost of Fraud"

To optimize the pipelines toward the specific business needs of this model, you can set your own assumptions for the cost of fraud. These parameters are

- `retry_percentage` - what percentage of customers will retry a transaction if it is declined?
- `interchange_fee` - how much of each successful transaction do you collect?
- `fraud_payout_percentage` - the percentage of fraud will you be unable to collect
- `amount_col` - the column in the data the represents the transaction amount

Using these parameters, EvalML determines attempt to build a pipeline that will minimize the financial loss due to fraud.

[2]:
```
fraud_objective = FraudCost(retry_percentage=.5,
                            interchange_fee=.02,
                            fraud_payout_percentage=.75,
                            amount_col='amount')
```

3.1.2 Search for best pipeline

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

[3]:
```
X, y = evalml.demos.load_fraud(n_rows=2500)
```

```
             Number of Features
Boolean                       1
Categorical                   6
Numeric                       5

Number of training examples: 2500
Targets
False    85.92%
True     14.08%
Name: fraud, dtype: object
```

EvalML natively supports one-hot encoding. Here we keep 1 out of the 6 categorical columns to decrease computation time.

[4]:
```
X = X.drop(['datetime', 'expiration_date', 'country', 'region', 'provider'], axis=1)

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

print(X.dtypes)
```

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

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

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

automl.search(X_train, y_train)
```

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

Optimizing for Fraud Cost.
Lower score is better.

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

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

(continues on next page)

(continued from previous page)

```
             'type'...
```

```
(1/5) Mode Baseline Binary Classification P... Elapsed:00:00
        Starting cross validation
        Finished cross validation - mean Fraud Cost: 0.023
(2/5) LightGBM Classifier w/ Imputer + One ... Elapsed:00:00
        Starting cross validation
        Finished cross validation - mean Fraud Cost: 0.015
(3/5) Extra Trees Classifier w/ Imputer + O... Elapsed:00:01
        Starting cross validation
        Finished cross validation - mean Fraud Cost: 0.002
(4/5) Elastic Net Classifier w/ Imputer + O... Elapsed:00:03
        Starting cross validation
        Finished cross validation - mean Fraud Cost: 0.002
(5/5) CatBoost Classifier w/ Imputer           Elapsed:00:04
        Starting cross validation
        Finished cross validation - mean Fraud Cost: 0.002

Search finished after 00:05
Best pipeline: Extra Trees Classifier w/ Imputer + One Hot Encoder
Best pipeline Fraud Cost: 0.002316
```

View rankings and select pipeline

Once the fitting process is done, we can see all of the pipelines that were searched, ranked by their score on the fraud detection objective we defined

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

[6]:
```
   id                                    pipeline_name     score  \
0   2  Extra Trees Classifier w/ Imputer + One Hot En...  0.002316
1   3  Elastic Net Classifier w/ Imputer + One Hot En...  0.002316
2   4                    CatBoost Classifier w/ Imputer  0.002316
3   1   LightGBM Classifier w/ Imputer + One Hot Encoder  0.015253
4   0       Mode Baseline Binary Classification Pipeline  0.022830

   validation_score  percent_better_than_baseline  high_variance_cv  \
0          0.002169                     89.855612             False
1          0.002169                     89.855612             False
2          0.002169                     89.855612             False
3          0.005673                     33.191293              True
4          0.028406                      0.000000              True

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

to select the best pipeline we can run

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

Describe pipeline

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

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

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

Problem Type: binary
Model Family: Linear

Pipeline Steps
==============
1. Imputer
         * categorical_impute_strategy : most_frequent
         * numeric_impute_strategy : mean
         * categorical_fill_value : None
         * numeric_fill_value : None
2. One Hot Encoder
         * top_n : 10
         * categories : None
         * drop : None
         * handle_unknown : ignore
         * handle_missing : error
3. Standard Scaler
4. Elastic Net Classifier
         * alpha : 0.5
         * l1_ratio : 0.5
         * n_jobs : -1
         * max_iter : 1000
         * penalty : elasticnet
         * loss : log

Training
========
Training for binary problems.
Objective to optimize binary classification pipeline thresholds for: <evalml.
↪objectives.fraud_cost.FraudCost object at 0x7fc74d246a58>
Total training time (including CV): 1.0 seconds

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

3.1.3 Evaluate on hold out

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

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

```
[9]: GeneratedPipeline(parameters={'Imputer':{'categorical_impute_strategy': 'most_frequent
     ↪', 'numeric_impute_strategy': 'mean', 'categorical_fill_value': None, 'numeric_fill_
     ↪value': None}, 'One Hot Encoder':{'top_n': 10, 'categories': None, 'drop': None,
     ↪'handle_unknown': 'ignore', 'handle_missing': 'error'}, 'Extra Trees Classifier':{
     ↪'n_estimators': 100, 'max_features': 'auto', 'max_depth': 6, 'min_samples_split': 2,
     ↪ 'min_weight_fraction_leaf': 0.0, 'n_jobs': -1},})
```

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

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

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

3.1.4 Why optimize for a problem-specific objective?

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

```
[11]: automl_auc = AutoMLSearch(problem_type='binary',
                                objective='auc',
                                additional_objectives=['f1', 'precision'],
                                max_iterations=5,
                                optimize_thresholds=True)

      automl_auc.search(X_train, y_train)
```

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

Optimizing for AUC.
Greater score is better.

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

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

```
(1/5) Mode Baseline Binary Classification P... Elapsed:00:00
        Starting cross validation
        Finished cross validation - mean AUC: 0.500
(2/5) LightGBM Classifier w/ Imputer + One ... Elapsed:00:00
        Starting cross validation
        Finished cross validation - mean AUC: 0.861
(3/5) Extra Trees Classifier w/ Imputer + O... Elapsed:00:00
        Starting cross validation
        Finished cross validation - mean AUC: 0.822
(4/5) Elastic Net Classifier w/ Imputer + O... Elapsed:00:02
```

(continues on next page)

(continued from previous page)

```
        Starting cross validation
        Finished cross validation - mean AUC: 0.500
(5/5) CatBoost Classifier w/ Imputer           Elapsed:00:02
        Starting cross validation
        Finished cross validation - mean AUC: 0.844

Search finished after 00:03
Best pipeline: LightGBM Classifier w/ Imputer + One Hot Encoder
Best pipeline AUC: 0.861385
```

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

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

[12]:
```
   id                                     pipeline_name     score  \
0   1   LightGBM Classifier w/ Imputer + One Hot Encoder  0.861385
1   4                     CatBoost Classifier w/ Imputer  0.843946
2   2  Extra Trees Classifier w/ Imputer + One Hot En...  0.821698
3   0       Mode Baseline Binary Classification Pipeline  0.500000
4   3  Elastic Net Classifier w/ Imputer + One Hot En...  0.500000

   validation_score  percent_better_than_baseline  high_variance_cv  \
0          0.844306                     72.277000             False
1          0.840565                     68.789178             False
2          0.806227                     64.339583             False
3          0.500000                      0.000000             False
4          0.500000                      0.000000             False

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

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

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

[13]:
```
GeneratedPipeline(parameters={'Imputer':{'categorical_impute_strategy': 'most_frequent
↪', 'numeric_impute_strategy': 'mean', 'categorical_fill_value': None, 'numeric_fill_
↪value': None}, 'One Hot Encoder':{'top_n': 10, 'categories': None, 'drop': None,
↪'handle_unknown': 'ignore', 'handle_missing': 'error'}, 'LightGBM Classifier':{
↪'boosting_type': 'gbdt', 'learning_rate': 0.1, 'n_estimators': 100, 'max_depth': 0,
↪'num_leaves': 31, 'min_child_samples': 20, 'n_jobs': -1},})
```

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

[14]:
```
OrderedDict([('AUC', 0.8441860465116278),
             ('Fraud Cost', 0.004348876090615287)])
```

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

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

When we optimize for AUC, we can see that the AUC score from this pipeline is better than the AUC score from the pipeline optimized for fraud cost. However, the losses due to fraud are over 3% of the total transaction amount when optimized for AUC and under 1% when optimized for fraud cost. As a result, we lose more than 2% of the total transaction amount by not optimizing for fraud cost specifically.

This happens because optimizing for AUC does not take into account the user-specified `retry_percentage`, `interchange_fee`, `fraud_payout_percentage` values. Thus, the best pipelines may produce the highest AUC but may not actually reduce the amount loss due to your specific type fraud.

This example highlights how performance in the real world can diverge greatly from machine learning metrics.

3.2 Building a Lead Scoring Model with EvalML

In this demo, we will build an optimized lead scoring model using EvalML. To optimize the pipeline, we will set up an objective function to maximize the revenue generated with true positives while taking into account the cost of false positives. At the end of this demo, we also show you how introducing the right objective during the training is over 6x better than using a generic machine learning metric like AUC.

[1]:
```
import evalml
from evalml import AutoMLSearch
from evalml.objectives import LeadScoring
```

3.2.1 Configure LeadScoring

To optimize the pipelines toward the specific business needs of this model, you can set your own assumptions for how much value is gained through true positives and the cost associated with false positives. These parameters are

- `true_positive` - dollar amount to be gained with a successful lead
- `false_positive` - dollar amount to be lost with an unsuccessful lead

Using these parameters, EvalML builds a pileline that will maximize the amount of revenue per lead generated.

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

3.2.2 Dataset

We will be utilizing a dataset detailing a customer's job, country, state, zip, online action, the dollar amount of that action and whether they were a successful lead.

[3]:
```
from urllib.request import urlopen
import pandas as pd

customers_data = urlopen('https://featurelabs-static.s3.amazonaws.com/lead_scoring_ml_
↪apps/customers.csv')
interactions_data = urlopen('https://featurelabs-static.s3.amazonaws.com/lead_scoring_
↪ml_apps/interactions.csv')
leads_data = urlopen('https://featurelabs-static.s3.amazonaws.com/lead_scoring_ml_
↪apps/previous_leads.csv')
customers = pd.read_csv(customers_data)
```

(continues on next page)

(continued from previous page)

```
interactions = pd.read_csv(interactions_data)
leads = pd.read_csv(leads_data)

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

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

display(X.head())
```

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

3.2.3 Search for best pipeline

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

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

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

print(X.dtypes)
```

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

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

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

automl.search(X_train, y_train)
```

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

Optimizing for Lead Scoring.
```

(continues on next page)

(continued from previous page)

```
Greater score is better.

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

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

```
(1/5) Mode Baseline Binary Classification P... Elapsed:00:00
        Starting cross validation
        Finished cross validation - mean Lead Scoring: 0.000
(2/5) LightGBM Classifier w/ Imputer + One ... Elapsed:00:01
        Starting cross validation
        Finished cross validation - mean Lead Scoring: 35.962
(3/5) Extra Trees Classifier w/ Imputer + O... Elapsed:00:02
        Starting cross validation
        Finished cross validation - mean Lead Scoring: 42.269
(4/5) Elastic Net Classifier w/ Imputer + O... Elapsed:00:05
        Starting cross validation
        Finished cross validation - mean Lead Scoring: 42.140
(5/5) CatBoost Classifier w/ Imputer           Elapsed:00:06
        Starting cross validation
        Finished cross validation - mean Lead Scoring: 42.140

Search finished after 00:07
Best pipeline: Extra Trees Classifier w/ Imputer + One Hot Encoder
Best pipeline Lead Scoring: 42.269377
```

View rankings and select pipeline

Once the fitting process is done, we can see all of the pipelines that were searched, ranked by their score on the lead scoring objective we defined

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

[6]:
```
   id                                     pipeline_name      score  \
0   2  Extra Trees Classifier w/ Imputer + One Hot En...  42.269377
1   3  Elastic Net Classifier w/ Imputer + One Hot En...  42.140250
2   4                    CatBoost Classifier w/ Imputer  42.140250
3   1   LightGBM Classifier w/ Imputer + One Hot Encoder  35.961746
4   0       Mode Baseline Binary Classification Pipeline   0.000000

   validation_score  percent_better_than_baseline  high_variance_cv  \
0         42.200000                           NaN             False
1         42.129032                           NaN             False
2         42.129032                           NaN             False
3         31.890323                           NaN             False
4          0.000000                           NaN             False

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

(continues on next page)

(continued from previous page)

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

to select the best pipeline we can run

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

Describe pipeline

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

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

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

Problem Type: binary
Model Family: Extra Trees

Pipeline Steps
==============
1. Imputer
         * categorical_impute_strategy : most_frequent
         * numeric_impute_strategy : mean
         * categorical_fill_value : None
         * numeric_fill_value : None
2. One Hot Encoder
         * top_n : 10
         * categories : None
         * drop : None
         * handle_unknown : ignore
         * handle_missing : error
3. Extra Trees Classifier
         * n_estimators : 100
         * max_features : auto
         * max_depth : 6
         * min_samples_split : 2
         * min_weight_fraction_leaf : 0.0
         * n_jobs : -1

Training
========
Training for binary problems.
Objective to optimize binary classification pipeline thresholds for: <evalml.
↪objectives.lead_scoring.LeadScoring object at 0x7f680519e978>
Total training time (including CV): 2.3 seconds

Cross Validation
----------
             Lead Scoring   AUC # Training # Testing
0                  42.200 0.743   2479.000  1550.000
1                  42.006 0.684   2479.000  1550.000
2                  42.602 0.708   2480.000  1549.000
mean               42.269 0.711          -         -
```

(continues on next page)

(continued from previous page)

```
std                  0.304 0.030          -          -
coef of var          0.007 0.042          -          -
```

3.2.4 Evaluate on hold out

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

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

[9]:
```
GeneratedPipeline(parameters={'Imputer':{'categorical_impute_strategy': 'most_frequent
↪', 'numeric_impute_strategy': 'mean', 'categorical_fill_value': None, 'numeric_fill_
↪value': None}, 'One Hot Encoder':{'top_n': 10, 'categories': None, 'drop': None,
↪'handle_unknown': 'ignore', 'handle_missing': 'error'}, 'Extra Trees Classifier':{
↪'n_estimators': 100, 'max_features': 'auto', 'max_depth': 6, 'min_samples_split': 2,
↪ 'min_weight_fraction_leaf': 0.0, 'n_jobs': -1},})
```

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

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

[10]:
```
OrderedDict([('AUC', 0.6895814445451798),
             ('Lead Scoring', -0.017196904557179708)])
```

3.2.5 Why optimize for a problem-specific objective?

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

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

automl_auc.search(X_train, y_train)
```

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

Optimizing for AUC.
Greater score is better.

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

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

```
(1/5) Mode Baseline Binary Classification P... Elapsed:00:00
        Starting cross validation
        Finished cross validation - mean AUC: 0.500
(2/5) LightGBM Classifier w/ Imputer + One ... Elapsed:00:00
        Starting cross validation
        Finished cross validation - mean AUC: 0.687
(3/5) Extra Trees Classifier w/ Imputer + O... Elapsed:00:00
        Starting cross validation
        Finished cross validation - mean AUC: 0.716
(4/5) Elastic Net Classifier w/ Imputer + O... Elapsed:00:01
        Starting cross validation
        Finished cross validation - mean AUC: 0.500
(5/5) CatBoost Classifier w/ Imputer           Elapsed:00:02
        Starting cross validation
        Finished cross validation - mean AUC: 0.582

Search finished after 00:02
Best pipeline: Extra Trees Classifier w/ Imputer + One Hot Encoder
Best pipeline AUC: 0.716259
```

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

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

[12]:
```
   id                                     pipeline_name     score  \
0   2  Extra Trees Classifier w/ Imputer + One Hot En...  0.716259
1   1   LightGBM Classifier w/ Imputer + One Hot Encoder  0.686930
2   4                     CatBoost Classifier w/ Imputer  0.581861
3   0       Mode Baseline Binary Classification Pipeline  0.500000
4   3  Elastic Net Classifier w/ Imputer + One Hot En...  0.500000

   validation_score  percent_better_than_baseline  high_variance_cv  \
0          0.737951                     43.251821             False
1          0.685421                     37.385993             False
2          0.558533                     16.372274             False
3          0.500000                      0.000000             False
4          0.500000                      0.000000             False

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

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

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

[13]:
```
GeneratedPipeline(parameters={'Imputer':{'categorical_impute_strategy': 'most_frequent
↪', 'numeric_impute_strategy': 'mean', 'categorical_fill_value': None, 'numeric_fill_
↪value': None}, 'One Hot Encoder':{'top_n': 10, 'categories': None, 'drop': None,
↪'handle_unknown': 'ignore', 'handle_missing': 'error'}, 'Extra Trees Classifier':{
↪'n_estimators': 100, 'max_features': 'auto', 'max_depth': 6, 'min_samples_split': 2,
↪ 'min_weight_fraction_leaf': 0.0, 'n_jobs': -1},})
```

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

[14]:
```
OrderedDict([('AUC', 0.6895814445451798),
             ('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.

3.3 Using the Cost-Benefit Matrix Objective

The Cost-Benefit Matrix (`CostBenefitMatrix`) objective is an objective that assigns costs to each of the quadrants of a confusion matrix to quantify the cost of being correct or incorrect.

3.3.1 Confusion Matrix

Confusion matrices are tables that summarize the number of correct and incorrectly-classified predictions, broken down by each class. They allow us to quickly understand the performance of a classification model and where the model gets "confused" when it is making predictions. For the binary classification problem, there are four possible combinations of prediction and actual target values possible:

- true positives (correct positive assignments)
- true negatives (correct negative assignments)
- false positives (incorrect positive assignments)
- false negatives (incorrect negative assignments)

An example of how to calculate a confusion matrix can be found *here*.

3.3.2 Cost-Benefit Matrix

Although the confusion matrix is an incredibly useful visual for understanding our model, each prediction that is correctly or incorrectly classified is treated equally. For example, for detecting breast cancer, the confusion matrix does not take into consideration that it could be much more costly to incorrectly classify a malignant tumor as benign than it is to incorrectly classify a benign tumor as malignant. This is where the cost-benefit matrix shines: it uses the cost of each of the four possible outcomes to weigh each outcome differently. By scoring using the cost-benefit matrix, we can measure the score of the model by a concrete unit that is more closely related to the goal of the model. In the below example, we will show how the cost-benefit matrix objective can be used, and how it can give us better real-world impact when compared to using other standard machine learning objectives.

3.3.3 Customer Churn Example

Data

In this example, we will be using a customer churn data set taken from Kaggle.

This dataset includes records of over 7000 customers, and includes customer account information, demographic information, services they signed up for, and whether or not the customer "churned" or left within the last month.

The target we want to predict is whether the customer churned ("Yes") or did not churn ("No"). In the dataset, approximately 73.5% of customers did not churn, and 26.5% did. We will refer to the customers who churned as the "positive" class and the customers who did not churn as the "negative" class.

[1]:
```
from evalml.demos.churn import load_churn
X, y = load_churn()
```

```
             Number of Features
Categorical                  16
Numeric                       3

Number of training examples: 7043
Targets
No     73.46%
Yes    26.54%
Name: Churn, dtype: object
```

In this example, let's say that correctly identifying customers who will churn (true positive case) will give us a net profit of $400, because it allows us to intervene, incentivize the customer to stay, and sign a new contract. Incorrectly classifying customers who were not going to churn as customers who will churn (false positive case) will cost $100 to represent the marketing and effort used to try to retain the user. Not identifying customers who will churn (false negative case) will cost us $200 to represent the lost in revenue from losing a customer. Finally, correctly identifying customers who will not churn (true negative case) will not cost us anything ($0), as nothing needs to be done for that customer.

We can represent these values in our `CostBenefitMatrix` objective, where a negative value represents a cost and a positive value represents a profit–note that this means that the greater the score, the more profit we will make.

[2]:
```
from evalml.objectives import CostBenefitMatrix
cost_benefit_matrix = CostBenefitMatrix(true_positive=400, true_negative=0,
                        false_positive=-100, false_negative=-200)
```

AutoML Search with Log Loss

First, let us run AutoML search to train pipelines using the default objective for binary classification (log loss).

[3]:
```
from evalml import AutoMLSearch
automl = AutoMLSearch(problem_type='binary', objective='log loss binary')
automl.search(X, y)

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

```
Using default limit of max_iterations=5.

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

(continues on next page)

(continued from previous page)

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

Optimizing for Log Loss Binary.
Lower score is better.

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

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

```
(1/5) Mode Baseline Binary Classification P... Elapsed:00:00
        Starting cross validation
        Finished cross validation - mean Log Loss Binary: 9.166
(2/5) LightGBM Classifier w/ Imputer + One ... Elapsed:00:00
        Starting cross validation
        Finished cross validation - mean Log Loss Binary: 0.439
(3/5) Extra Trees Classifier w/ Imputer + O... Elapsed:00:01
        Starting cross validation
        Finished cross validation - mean Log Loss Binary: 0.432
(4/5) Elastic Net Classifier w/ Imputer + O... Elapsed:00:04
        Starting cross validation
        Finished cross validation - mean Log Loss Binary: 0.579
(5/5) CatBoost Classifier w/ Imputer           Elapsed:00:05
        Starting cross validation
        Finished cross validation - mean Log Loss Binary: 0.600

Search finished after 00:06
Best pipeline: Extra Trees Classifier w/ Imputer + One Hot Encoder
Best pipeline Log Loss Binary: 0.432203
```

[3]:
```
OrderedDict([('Log Loss Binary', 0.4205318376942898)])
```

When we train our pipelines using log loss as our primary objective, we try to find pipelines that minimize log loss. However, our ultimate goal in training models is to find a model that gives us the most profit. When we score our pipeline on the cost benefit matrix (using the costs outlined above) to determine the profit we would earn from the predictions made by this model, we get approximately 14.851, indicating $14.85 in profit per customer. Across our 7043 customers, we get a total of approximately $104,588.55 in profit.

[4]:
```
ll_pipeline.score(X, y, [cost_benefit_matrix])
```

[4]:
```
OrderedDict([('Cost Benefit Matrix', 14.851625727672868)])
```

AutoML Search with Cost-Benefit Matrix

Let's try rerunning our AutoML search, but this time using the cost-benefit matrix as our primary objective to optimize.

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

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

```
Using default limit of max_iterations=5.

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

Optimizing for Cost Benefit Matrix.
Greater score is better.

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

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

```
(1/5) Mode Baseline Binary Classification P... Elapsed:00:00
        Starting cross validation
        Finished cross validation - mean Cost Benefit Matrix: -53.074
(2/5) LightGBM Classifier w/ Imputer + One ... Elapsed:00:00
        Starting cross validation
        Finished cross validation - mean Cost Benefit Matrix: 23.086
(3/5) Extra Trees Classifier w/ Imputer + O... Elapsed:00:01
        Starting cross validation
        Finished cross validation - mean Cost Benefit Matrix: 13.801
(4/5) Elastic Net Classifier w/ Imputer + O... Elapsed:00:03
        Starting cross validation
        Finished cross validation - mean Cost Benefit Matrix: -53.074
(5/5) CatBoost Classifier w/ Imputer           Elapsed:00:05
        Starting cross validation
        Finished cross validation - mean Cost Benefit Matrix: 16.087

Search finished after 00:05
Best pipeline: LightGBM Classifier w/ Imputer + One Hot Encoder
Best pipeline Cost Benefit Matrix: 23.086445
```

[5]:
```
GeneratedPipeline(parameters={'Imputer':{'categorical_impute_strategy': 'most_frequent
↪', 'numeric_impute_strategy': 'mean', 'categorical_fill_value': None, 'numeric_fill_
↪value': None}, 'One Hot Encoder':{'top_n': 10, 'categories': None, 'drop': None,
↪'handle_unknown': 'ignore', 'handle_missing': 'error'}, 'LightGBM Classifier':{
↪'boosting_type': 'gbdt', 'learning_rate': 0.1, 'n_estimators': 100, 'max_depth': 0,
↪'num_leaves': 31, 'min_child_samples': 20, 'n_jobs': -1},})
```

Now, if we calculate the cost-benefit matrix score on our best pipeline, we see that with this pipeline optimized for our cost-benefit matrix objective, we are able to generate $42.24 in profit per customer. Across our 7043 customers, we get a total of approximately $297,496.32 in profit–much more than the profit returned from our previous pipeline! Custom objectives like `CostBenefitMatrix` are just one example of how using EvalML can help find pipelines that can perform better on real-world problems, rather than on arbitrary standard statistical metrics.

[6]:
```
cbm_pipeline.score(X, y, [cost_benefit_matrix])
```

[6]:
```
OrderedDict([('Cost Benefit Matrix', 47.87732500354963)])
```

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

[7]:
```
from evalml.model_understanding.graphs import graph_confusion_matrix

# pipeline trained with log loss
y_pred = ll_pipeline.predict(X)
graph_confusion_matrix(y, y_pred)
```

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

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

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

CHAPTER

FOUR

USER GUIDE

These guides include in-depth descriptions and explanations of EvalML's features.

4.1 Automated Machine Learning (AutoML) Search

4.1.1 Background

Machine Learning

Machine learning (ML) is the process of constructing a mathematical model of a system based on a sample dataset collected from that system.

One of the main goals of training an ML model is to teach the model to separate the signal present in the data from the noise inherent in system and in the data collection process. If this is done effectively, the model can then be used to make accurate predictions about the system when presented with new, similar data. Additionally, introspecting on an ML model can reveal key information about the system being modeled, such as which inputs and transformations of the inputs are most useful to the ML model for learning the signal in the data, and are therefore the most predictive.

There are a variety of ML problem types. Supervised learning describes the case where the collected data contains an output value to be modeled and a set of inputs with which to train the model. EvalML focuses on training supervised learning models.

EvalML supports three common supervised ML problem types. The first is regression, where the target value to model is a continuous numeric value. Next are binary and multiclass classification, where the target value to model consists of two or more discrete values or categories. The choice of which supervised ML problem type is most appropriate depends on domain expertise and on how the model will be evaluated and used.

AutoML and Search

AutoML is the process of automating the construction, training and evaluation of ML models. Given a data and some configuration, AutoML searches for the most effective and accurate ML model or models to fit the dataset. During the search, AutoML will explore different combinations of model type, model parameters and model architecture.

An effective AutoML solution offers several advantages over constructing and tuning ML models by hand. AutoML can assist with many of the difficult aspects of ML, such as avoiding overfitting and underfitting, imbalanced data, detecting data leakage and other potential issues with the problem setup, and automatically applying best-practice data cleaning, feature engineering, feature selection and various modeling techniques. AutoML can also leverage search algorithms to optimally sweep the hyperparameter search space, resulting in model performance which would be difficult to achieve by manual training.

4.1.2 AutoML in EvalML

EvalML supports all of the above and more.

In its simplest usage, the AutoML search interface requires only the input data, the target data and a `problem_type` specifying what kind of supervised ML problem to model.

** Graphing methods, like AutoMLSearch, on Jupyter Notebook and Jupyter Lab require ipywidgets to be installed.

** If graphing on Jupyter Lab, jupyterlab-plotly required. To download this, make sure you have npm installed.

[1]:
```
import evalml

X, y = evalml.demos.load_breast_cancer()

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

```
Using default limit of max_iterations=5.

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

Optimizing for Log Loss Binary.
Lower score is better.

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

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

```
(1/5) Mode Baseline Binary Classification P... Elapsed:00:00
        Starting cross validation
        Finished cross validation - mean Log Loss Binary: 12.868
(2/5) LightGBM Classifier w/ Imputer           Elapsed:00:00
        Starting cross validation
        Finished cross validation - mean Log Loss Binary: 0.138
(3/5) Extra Trees Classifier w/ Imputer        Elapsed:00:00
        Starting cross validation
        Finished cross validation - mean Log Loss Binary: 0.146
(4/5) Elastic Net Classifier w/ Imputer + S... Elapsed:00:02
        Starting cross validation
        Finished cross validation - mean Log Loss Binary: 0.504
(5/5) CatBoost Classifier w/ Imputer           Elapsed:00:02
        Starting cross validation
        Finished cross validation - mean Log Loss Binary: 0.390

Search finished after 00:02
Best pipeline: LightGBM Classifier w/ Imputer
Best pipeline Log Loss Binary: 0.137999
```

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

There are a number of mechanisms to control the AutoML search time. One way is to set the maximum number of

candidate models to be evaluated during AutoML using `max_iterations`. By default, AutoML will search a fixed number of iterations and parameter pairs (`max_iterations=5`). The first pipeline to be evaluated will always be a baseline model representing a trivial solution.

The AutoML interface supports a variety of other parameters. For a comprehensive list, please *refer to the API reference.*

Detecting Problem Type

EvalML includes a simple method, `detect_problem_type`, to help determine the problem type given the target data.

This function can return the predicted problem type as a ProblemType enum, choosing from ProblemType.BINARY, ProblemType.MULTICLASS, and ProblemType.REGRESSION. If the target data is invalid (for instance when there is only 1 unique label), the function will throw an error instead.

[2]:
```
import pandas as pd
from evalml.problem_types import detect_problem_type

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

[2]:
```
<ProblemTypes.BINARY: 'binary'>
```

Objective parameter

AutoMLSearch takes in an `objective` parameter to determine which `objective` to optimize for. By default, this parameter is set to `auto`, which allows AutoML to choose `LogLossBinary` for binary classification problems, `LogLossMulticlass` for multiclass classification problems, and `R2` for regression problems.

It should be noted that the `objective` parameter is only used in ranking and helping choose the pipelines to iterate over, but is not used to optimize each individual pipeline during fit-time.

To get the default objective for each problem type, you can use the `get_default_primary_search_objective` function.

[3]:
```
from evalml.automl import get_default_primary_search_objective

binary_objective = get_default_primary_search_objective("binary")
multiclass_objective = get_default_primary_search_objective("multiclass")
regression_objective = get_default_primary_search_objective("regression")

print(binary_objective.name)
print(multiclass_objective.name)
print(regression_objective.name)
```

```
Log Loss Binary
Log Loss Multiclass
R2
```

Using custom pipelines

EvalML's AutoML algorithm generates a set of pipelines to search with. To provide a custom set instead, set allowed_pipelines to a list of *custom pipeline* classes. Note: this will prevent AutoML from generating other pipelines to search over.

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

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

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

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

Stopping the search early

To stop the search early, hit `Ctrl-C`. This will bring up a prompt asking for confirmation. Responding with `y` will immediately stop the search. Responding with `n` will continue the search.

4.1.3 View Rankings

A summary of all the pipelines built can be returned as a pandas DataFrame which is sorted by score. The score column contains the average score across all cross-validation folds while the validation_score column is computed from the first cross-validation fold.

[5]:
```
automl.rankings
```

[5]:
```
   id                                     pipeline_name      score  \
0   1                   LightGBM Classifier w/ Imputer   0.137999
1   2                 Extra Trees Classifier w/ Imputer   0.146066
2   4                    CatBoost Classifier w/ Imputer   0.390101
3   3  Elastic Net Classifier w/ Imputer + Standard S...   0.503740
4   0       Mode Baseline Binary Classification Pipeline  12.868443

   validation_score  percent_better_than_baseline  high_variance_cv  \
0          0.176639                     98.927620              True
1          0.156833                     98.864930             False
2          0.397696                     96.968547             False
3          0.510650                     96.085460             False
4         12.906595                      0.000000             False

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

4.1.4 Describe Pipeline

Each pipeline is given an `id`. We can get more information about any particular pipeline using that `id`. Here, we will get more information about the pipeline with `id = 1`.

[6]:
```
automl.describe_pipeline(1)
```

```
*********************************
* LightGBM Classifier w/ Imputer *
*********************************

Problem Type: binary
Model Family: LightGBM

Pipeline Steps
==============
1. Imputer
         * categorical_impute_strategy : most_frequent
         * numeric_impute_strategy : mean
         * categorical_fill_value : None
         * numeric_fill_value : None
2. LightGBM Classifier
         * boosting_type : gbdt
         * learning_rate : 0.1
         * n_estimators : 100
         * max_depth : 0
         * num_leaves : 31
         * min_child_samples : 20
         * n_jobs : -1

Training
========
Training for binary problems.
Total training time (including CV): 0.5 seconds

Cross Validation
----------
High variance within cross validation scores. Model may not perform as estimated on␣
↪unseen data.
             Log Loss Binary  MCC Binary   AUC  Precision    F1  Balanced Accuracy␣
↪Binary  Accuracy Binary # Training # Testing
0                      0.177       0.899 0.991      0.943 0.936                     0.
↪948            0.953    379.000   190.000
1                      0.104       0.955 0.995      1.000 0.971                     0.
↪972            0.979    379.000   190.000
2                      0.133       0.909 0.986      0.943 0.943                     0.
↪955            0.958    380.000   189.000
mean                   0.138       0.921 0.991      0.962 0.950                     0.
↪958            0.963          -         -
std                    0.036       0.030 0.005      0.033 0.018                     0.
↪012            0.014          -         -
coef of var            0.264       0.033 0.005      0.034 0.019                     0.
↪013            0.015          -         -
```

4.1.5 Get Pipeline

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

[7]:
```
pipeline = automl.get_pipeline(1)
print(pipeline.name)
print(pipeline.parameters)
```

```
LightGBM Classifier w/ Imputer
{'Imputer': {'categorical_impute_strategy': 'most_frequent',␣
↪'numeric_impute_strategy': 'mean', 'categorical_fill_value': None,␣
↪'numeric_fill_value': None}, 'LightGBM Classifier': {'boosting_type': 'gbdt',␣
↪'learning_rate': 0.1, 'n_estimators': 100, 'max_depth': 0, 'num_leaves': 31,␣
↪'min_child_samples': 20, 'n_jobs': -1}}
```

Get best pipeline

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

[8]:
```
best_pipeline = automl.best_pipeline
print(best_pipeline.name)
print(best_pipeline.parameters)
```

```
LightGBM Classifier w/ Imputer
{'Imputer': {'categorical_impute_strategy': 'most_frequent',␣
↪'numeric_impute_strategy': 'mean', 'categorical_fill_value': None,␣
↪'numeric_fill_value': None}, 'LightGBM Classifier': {'boosting_type': 'gbdt',␣
↪'learning_rate': 0.1, 'n_estimators': 100, 'max_depth': 0, 'num_leaves': 31,␣
↪'min_child_samples': 20, 'n_jobs': -1}}
```

4.1.6 Access raw results

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

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

[9]:
```
{'pipeline_results': {0: {'id': 0,
   'pipeline_name': 'Mode Baseline Binary Classification Pipeline',
   'pipeline_class': evalml.pipelines.classification.baseline_binary.
↪ModeBaselineBinaryPipeline,
   'pipeline_summary': 'Baseline Classifier',
   'parameters': {'Baseline Classifier': {'strategy': 'mode'}},
   'score': 12.868443394958925,
   'high_variance_cv': False,
   'training_time': 0.049742698669433594,
   'cv_data': [{'all_objective_scores': OrderedDict([('Log Loss Binary',
                 12.906595389677152),
                ('MCC Binary', 0.0),
                ('AUC', 0.5),
                ('Precision', 0.0),
                ('F1', 0.0),
                ('Balanced Accuracy Binary', 0.5),
                ('Accuracy Binary', 0.6263157894736842),
                ('# Training', 379),
                ('# Testing', 190)]),
     'score': 12.906595389677152,
     'binary_classification_threshold': 0.5},
    {'all_objective_scores': OrderedDict([('Log Loss Binary',
                 12.906595389677149),
                ('MCC Binary', 0.0),
                ('AUC', 0.5),
                ('Precision', 0.0),
```

(continues on next page)

(continued from previous page)

```
                ('F1', 0.0),
                ('Balanced Accuracy Binary', 0.5),
                ('Accuracy Binary', 0.6263157894736842),
                ('# Training', 379),
                ('# Testing', 190)]),
   'score': 12.906595389677149,
   'binary_classification_threshold': 0.5},
  {'all_objective_scores': OrderedDict([('Log Loss Binary',
                12.792139405522475),
                ('MCC Binary', 0.0),
                ('AUC', 0.5),
                ('Precision', 0.0),
                ('F1', 0.0),
                ('Balanced Accuracy Binary', 0.5),
                ('Accuracy Binary', 0.6296296296296297),
                ('# Training', 380),
                ('# Testing', 189)]),
   'score': 12.792139405522475,
   'binary_classification_threshold': 0.5}],
 'percent_better_than_baseline': 0,
 'validation_score': 12.906595389677152},
1: {'id': 1,
 'pipeline_name': 'LightGBM Classifier w/ Imputer',
 'pipeline_class': evalml.pipelines.utils.make_pipeline.<locals>.GeneratedPipeline,
 'pipeline_summary': 'LightGBM Classifier w/ Imputer',
 'parameters': {'Imputer': {'categorical_impute_strategy': 'most_frequent',
   'numeric_impute_strategy': 'mean',
   'categorical_fill_value': None,
   'numeric_fill_value': None},
  'LightGBM Classifier': {'boosting_type': 'gbdt',
   'learning_rate': 0.1,
   'n_estimators': 100,
   'max_depth': 0,
   'num_leaves': 31,
   'min_child_samples': 20,
   'n_jobs': -1}},
 'score': 0.13799860263727523,
 'high_variance_cv': True,
 'training_time': 0.5125904083251953,
 'cv_data': [{'all_objective_scores': OrderedDict([('Log Loss Binary',
                0.17663855005522583),
                ('MCC Binary', 0.8985734479173947),
                ('AUC', 0.9911232098473193),
                ('Precision', 0.9428571428571428),
                ('F1', 0.9361702127659575),
                ('Balanced Accuracy Binary', 0.9479820097052906),
                ('Accuracy Binary', 0.9526315789473684),
                ('# Training', 379),
                ('# Testing', 190)]),
   'score': 0.17663855005522583,
   'binary_classification_threshold': 0.5},
  {'all_objective_scores': OrderedDict([('Log Loss Binary',
                0.1042038854081209),
                ('MCC Binary', 0.9554966130892879),
                ('AUC', 0.9952657119185703),
                ('Precision', 1.0),
                ('F1', 0.9710144927536231),
```

(continues on next page)

(continued from previous page)

```
                ('Balanced Accuracy Binary', 0.971830985915493),
                ('Accuracy Binary', 0.9789473684210527),
                ('# Training', 379),
                ('# Testing', 190)]),
   'score': 0.1042038854081209,
   'binary_classification_threshold': 0.5},
  {'all_objective_scores': OrderedDict([('Log Loss Binary',
                0.1331533724484789),
                ('MCC Binary', 0.9092436974789916),
                ('AUC', 0.9861944777911165),
                ('Precision', 0.9428571428571428),
                ('F1', 0.9428571428571428),
                ('Balanced Accuracy Binary', 0.9546218487394957),
                ('Accuracy Binary', 0.9576719576719577),
                ('# Training', 380),
                ('# Testing', 189)]),
   'score': 0.1331533724484789,
   'binary_classification_threshold': 0.5}],
 'percent_better_than_baseline': 98.9276200826952,
 'validation_score': 0.17663855005522583},
2: {'id': 2,
 'pipeline_name': 'Extra Trees Classifier w/ Imputer',
 'pipeline_class': evalml.pipelines.utils.make_pipeline.<locals>.GeneratedPipeline,
 'pipeline_summary': 'Extra Trees Classifier w/ Imputer',
 'parameters': {'Imputer': {'categorical_impute_strategy': 'most_frequent',
   'numeric_impute_strategy': 'mean',
   'categorical_fill_value': None,
   'numeric_fill_value': None},
  'Extra Trees Classifier': {'n_estimators': 100,
   'max_features': 'auto',
   'max_depth': 6,
   'min_samples_split': 2,
   'min_weight_fraction_leaf': 0.0,
   'n_jobs': -1}},
 'score': 0.14606590090087035,
 'high_variance_cv': False,
 'training_time': 1.167860984802246,
 'cv_data': [{'all_objective_scores': OrderedDict([('Log Loss Binary',
                0.15683251263377887),
                ('MCC Binary', 0.8778182058245897),
                ('AUC', 0.989821280624926),
                ('Precision', 0.9838709677419355),
                ('F1', 0.9172932330827067),
                ('Balanced Accuracy Binary', 0.9253757841164635),
                ('Accuracy Binary', 0.9421052631578948),
                ('# Training', 379),
                ('# Testing', 190)]),
   'score': 0.15683251263377887,
   'binary_classification_threshold': 0.5},
  {'all_objective_scores': OrderedDict([('Log Loss Binary',
                0.1314416667412096),
                ('MCC Binary', 0.9010215251215669),
                ('AUC', 0.995384069120606),
                ('Precision', 1.0),
                ('F1', 0.9323308270676691),
                ('Balanced Accuracy Binary', 0.9366197183098591),
                ('Accuracy Binary', 0.9526315789473684),
```

(continues on next page)

(continued from previous page)

```
                ('# Training', 379),
                ('# Testing', 190)]),
   'score': 0.1314416667412096,
   'binary_classification_threshold': 0.5},
  {'all_objective_scores': OrderedDict([('Log Loss Binary',
                0.14992352332762263),
                ('MCC Binary', 0.9092436974789916),
                ('AUC', 0.9879951980792318),
                ('Precision', 0.9428571428571428),
                ('F1', 0.9428571428571428),
                ('Balanced Accuracy Binary', 0.9546218487394957),
                ('Accuracy Binary', 0.9576719576719577),
                ('# Training', 380),
                ('# Testing', 189)]),
   'score': 0.14992352332762263,
   'binary_classification_threshold': 0.5}],
 'percent_better_than_baseline': 98.8649295301863,
 'validation_score': 0.15683251263377887},
3: {'id': 3,
 'pipeline_name': 'Elastic Net Classifier w/ Imputer + Standard Scaler',
 'pipeline_class': evalml.pipelines.utils.make_pipeline.<locals>.GeneratedPipeline,
 'pipeline_summary': 'Elastic Net Classifier w/ Imputer + Standard Scaler',
 'parameters': {'Imputer': {'categorical_impute_strategy': 'most_frequent',
   'numeric_impute_strategy': 'mean',
   'categorical_fill_value': None,
   'numeric_fill_value': None},
  'Elastic Net Classifier': {'alpha': 0.5,
   'l1_ratio': 0.5,
   'n_jobs': -1,
   'max_iter': 1000,
   'penalty': 'elasticnet',
   'loss': 'log'}},
 'score': 0.5037404207981783,
 'high_variance_cv': False,
 'training_time': 0.16781187057495117,
 'cv_data': [{'all_objective_scores': OrderedDict([('Log Loss Binary',
                0.5106495939683122),
                ('MCC Binary', 0.5269050806054695),
                ('AUC', 0.9815362764824239),
                ('Precision', 1.0),
                ('F1', 0.5510204081632654),
                ('Balanced Accuracy Binary', 0.6901408450704225),
                ('Accuracy Binary', 0.7684210526315789),
                ('# Training', 379),
                ('# Testing', 190)]),
   'score': 0.5106495939683122,
   'binary_classification_threshold': 0.5},
  {'all_objective_scores': OrderedDict([('Log Loss Binary',
                0.5088061082724736),
                ('MCC Binary', 0.5269050806054695),
                ('AUC', 0.9914782814534264),
                ('Precision', 1.0),
                ('F1', 0.5510204081632654),
                ('Balanced Accuracy Binary', 0.6901408450704225),
                ('Accuracy Binary', 0.7684210526315789),
                ('# Training', 379),
                ('# Testing', 190)]),
```

(continues on next page)

(continued from previous page)

```
    'score': 0.5088061082724736,
    'binary_classification_threshold': 0.5},
   {'all_objective_scores': OrderedDict([('Log Loss Binary',
                 0.49176556015374906),
                ('MCC Binary', 0.6324555320336759),
                ('AUC', 0.9854741896758704),
                ('Precision', 1.0),
                ('F1', 0.6792452830188679),
                ('Balanced Accuracy Binary', 0.7571428571428571),
                ('Accuracy Binary', 0.8201058201058201),
                ('# Training', 380),
                ('# Testing', 189)]),
    'score': 0.49176556015374906,
    'binary_classification_threshold': 0.5}],
  'percent_better_than_baseline': 96.0854595591918,
  'validation_score': 0.5106495939683122},
 4: {'id': 4,
  'pipeline_name': 'CatBoost Classifier w/ Imputer',
  'pipeline_class': evalml.pipelines.utils.make_pipeline.<locals>.GeneratedPipeline,
  'pipeline_summary': 'CatBoost Classifier w/ Imputer',
  'parameters': {'Imputer': {'categorical_impute_strategy': 'most_frequent',
    'numeric_impute_strategy': 'mean',
    'categorical_fill_value': None,
    'numeric_fill_value': None},
   'CatBoost Classifier': {'n_estimators': 10,
    'eta': 0.03,
    'max_depth': 6,
    'bootstrap_type': None,
    'silent': True,
    'allow_writing_files': False}},
  'score': 0.3901007526814435,
  'high_variance_cv': False,
  'training_time': 0.45626187324523926,
  'cv_data': [{'all_objective_scores': OrderedDict([('Log Loss Binary',
                 0.3976961654004939),
                ('MCC Binary', 0.8425009463611858),
                ('AUC', 0.9835483489170316),
                ('Precision', 0.9523809523809523),
                ('F1', 0.8955223880597014),
                ('Balanced Accuracy Binary', 0.9099301692507988),
                ('Accuracy Binary', 0.9263157894736842),
                ('# Training', 379),
                ('# Testing', 190)]),
    'score': 0.3976961654004939,
    'binary_classification_threshold': 0.5},
   {'all_objective_scores': OrderedDict([('Log Loss Binary',
                 0.39058702666377215),
                ('MCC Binary', 0.9216584956231404),
                ('AUC', 0.9958574979287491),
                ('Precision', 0.9848484848484849),
                ('F1', 0.948905109489051),
                ('Balanced Accuracy Binary', 0.9535447982009706),
                ('Accuracy Binary', 0.9631578947368421),
                ('# Training', 379),
                ('# Testing', 190)]),
    'score': 0.39058702666377215,
    'binary_classification_threshold': 0.5},
```

(continues on next page)

(continued from previous page)

```
    {'all_objective_scores': OrderedDict([('Log Loss Binary',
                   0.38201906598006447),
                  ('MCC Binary', 0.9218075091290715),
                  ('AUC', 0.9885954381752702),
                  ('Precision', 0.9315068493150684),
                  ('F1', 0.9510489510489512),
                  ('Balanced Accuracy Binary', 0.9647058823529412),
                  ('Accuracy Binary', 0.9629629629629629),
                  ('# Training', 380),
                  ('# Testing', 189)]),
     'score': 0.38201906598006447,
     'binary_classification_threshold': 0.5}],
   'percent_better_than_baseline': 96.9685474714505,
   'validation_score': 0.3976961654004939}},
 'search_order': [0, 1, 2, 3, 4]}
```

4.2 Objectives

4.2.1 Overview

One of the key choices to make when training an ML model is what metric to choose by which to measure the efficacy of the model at learning the signal. Such metrics are useful for comparing how well the trained models generalize to new similar data.

This choice of metric is a key component of AutoML because it defines the cost function the AutoML search will seek to optimize. In EvalML, these metrics are called **objectives**. AutoML will seek to minimize (or maximize) the objective score as it explores more pipelines and parameters and will use the feedback from scoring pipelines to tune the available hyperparameters and continue the search. Therefore, it is critical to have an objective function that represents how the model will be applied in the intended domain of use.

EvalML supports a variety of objectives from traditional supervised ML including mean squared error for regression problems and cross entropy or area under the ROC curve for classification problems. EvalML also allows the user to define a custom objective using their domain expertise, so that AutoML can search for models which provide the most value for the user's problem.

4.2.2 Core Objectives

Use the `get_core_objectives` method to get a list of which objectives are included with EvalML for each problem type:

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

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

```
MCC Binary
Log Loss Binary
AUC
Precision
F1
Balanced Accuracy Binary
Accuracy Binary
```

EvalML defines a base objective class for each problem type: `RegressionObjective`, `BinaryClassificationObjective` and `MulticlassClassificationObjective`. All EvalML objectives are a subclass of one of these.

Binary Classification Objectives and Thresholds

All binary classification objectives have a `threshold` property. Some binary classification objectives like log loss and AUC are unaffected by the choice of binary classification threshold, because they score based on predicted probabilities or examine a range of threshold values. These metrics are defined with `score_needs_proba` set to False. For all other binary classification objectives, we can compute the optimal binary classification threshold from the predicted probabilities and the target.

[2]:
```
from evalml.pipelines import BinaryClassificationPipeline
from evalml.demos import load_fraud
from evalml.objectives import F1

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

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

y_pred_proba = pipeline.predict_proba(X)[True]
pipeline.threshold = objective.optimize_threshold(y_pred_proba, y)
print(pipeline.threshold)
print(pipeline.score(X, y, objectives=[objective]))
```

```
             Number of Features
Boolean                       1
Categorical                   6
Numeric                       5

Number of training examples: 100
Targets
False    91.00%
True      9.00%
Name: fraud, dtype: object
None
OrderedDict([('F1', 0.5)])
0.2506657906442736
OrderedDict([('F1', 1.0)])
```

4.2.3 Custom Objectives

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

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

Defining a Custom Objective Function

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

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

Example: Fraud Detection

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

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


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

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

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

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

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

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

(continues on next page)

(continued from previous page)

```
        self.interchange_fee = interchange_fee
        self.fraud_payout_percentage = fraud_payout_percentage
        self.amount_col = amount_col

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

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

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

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

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

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

            Arguments:
                y_predicted (pd.Series): predicted fraud labels
                y_true (pd.Series): true fraud labels
                X (pd.DataFrame): dataframe with transaction amounts

            Returns:
                float: amount lost to fraud per transaction
        """
        if not isinstance(X, pd.DataFrame):
            X = pd.DataFrame(X)

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

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

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

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

(continues on next page)

(continued from previous page)

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

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

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

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

        loss_per_total_processed = loss / transaction_amount.sum()

        return loss_per_total_processed
```

4.3 Components

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

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

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

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

4.3.1 Transformers

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

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

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

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

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

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

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

display(X)
```

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

Below is a list of all transformers included with EvalML:

[3]:
```
from evalml.pipelines.components.utils import all_components, Estimator, Transformer
for component in all_components():
    if issubclass(component, Transformer):
        print(f"Transformer: {component.name}")
```

```
Transformer: Text Featurization Component
Transformer: LSA Transformer
Transformer: Drop Null Columns Transformer
Transformer: DateTime Featurization Component
Transformer: Select Columns Transformer
Transformer: Drop Columns Transformer
Transformer: Standard Scaler
Transformer: Imputer
Transformer: Per Column Imputer
Transformer: Simple Imputer
Transformer: RF Regressor Select From Model
Transformer: RF Classifier Select From Model
Transformer: One Hot Encoder
```

4.3.2 Estimators

Each estimator wraps an ML algorithm. Estimators subclass the `Estimator` class, and define a `fit` method to learn information from training data and a `predict` method for generating predictions from new data. Classification estimators should also define a `predict_proba` method for generating predicted probabilities.

Estimator classes each define a `model_family` attribute indicating what type of model is used.

Here's an example of using the *LogisticRegressionClassifier* estimator to fit and predict on a simple dataset:

[4]:
```
from evalml.pipelines.components import LogisticRegressionClassifier

clf = LogisticRegressionClassifier()

X = X
y = [1, 0]

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

[4]:
```
0    0
1    0
dtype: int64
```

Below is a list of all estimators included with EvalML:

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

```
Estimator: Decision Tree Regressor.
Estimator: Baseline Regressor
Estimator: Extra Trees Regressor
Estimator: XGBoost Regressor
Estimator: CatBoost Regressor
Estimator: Random Forest Regressor
Estimator: Linear Regressor
Estimator: Elastic Net Regressor
Estimator: Decision Tree Classifier.
Estimator: LightGBM Classifier
Estimator: Baseline Classifier
Estimator: Extra Trees Classifier
Estimator: Elastic Net Classifier
Estimator: CatBoost Classifier
Estimator: XGBoost Classifier
Estimator: Random Forest Classifier
Estimator: Logistic Regression Classifier
```

4.3.3 Defining Custom Components

EvalML allows you to easily create your own custom components by following the steps below.

Custom Transformers

Your transformer must inherit from the correct subclass. In this case *Transformer* for components that transform data. Next we will use EvalML's *DropNullColumns* as an example.

[6]:
```
import pandas as pd
from evalml.pipelines.components import Transformer

class DropNullColumns(Transformer):
    """Transformer to drop features whose percentage of NaN values exceeds a␣
↪specified threshold"""
    name = "Drop Null Columns Transformer"
    hyperparameter_ranges = {}

    def __init__(self, pct_null_threshold=1.0, random_state=0, **kwargs):
        """Initalizes an transformer to drop features whose percentage of NaN values␣
↪exceeds a specified threshold.

        Arguments:
            pct_null_threshold(float): The percentage of NaN values in an input␣
↪feature to drop.
                Must be a value between [0, 1] inclusive. If equal to 0.0, will drop␣
↪columns with any null values.
                If equal to 1.0, will drop columns with all null values. Defaults to␣
↪0.95.
        """
        if pct_null_threshold < 0 or pct_null_threshold > 1:
            raise ValueError("pct_null_threshold must be a float between 0 and 1,␣
↪inclusive.")
```

(continues on next page)

(continued from previous page)

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

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

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

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

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

Required fields

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

Required methods

Likewise, there are select methods you need to override as `Transformer` is an abstract base class:

- `__init__()` - the `__init__()` method of your transformer will need to call `super().__init__()` and pass three parameters in: a `parameters` dictionary holding the parameters to the component, the `component_obj`, and the `random_state` value. You can see that `component_obj` is set to `None` above and we will discuss `component_obj` in depth later on.
- `fit()` - the `fit()` method is responsible for fitting your component on training data.
- `transform()` - after fitting a component, the `transform()` method will take in new data and transform accordingly. Note: a component must call `fit()` before `transform()`.

You can also call or override `fit_transform()` that combines `fit()` and `transform()` into one method.

Custom Estimators

Your estimator must inherit from the correct subclass. In this case *Estimator* for components that predict new target values. Next we will use EvalML's *BaselineRegressor* as an example.

[7]:
```
import numpy as np
import pandas as pd

from evalml.model_family import ModelFamily
from evalml.pipelines.components.estimators import Estimator
from evalml.problem_types import ProblemTypes


class BaselineRegressor(Estimator):
    """Regressor that predicts using the specified strategy.

    This is useful as a simple baseline regressor to compare with other regressors.
    """
    name = "Baseline Regressor"
    hyperparameter_ranges = {}
    model_family = ModelFamily.BASELINE
    supported_problem_types = [ProblemTypes.REGRESSION]

    def __init__(self, strategy="mean", random_state=0, **kwargs):
        """Baseline regressor that uses a simple strategy to make predictions.

        Arguments:
            strategy (str): method used to predict. Valid options are "mean", "median
↪". Defaults to "mean".
            random_state (int, np.random.RandomState): seed for the random number␣
↪generator

        """
        if strategy not in ["mean", "median"]:
            raise ValueError("'strategy' parameter must equal either 'mean' or 'median
↪'")
        parameters = {"strategy": strategy}
        parameters.update(kwargs)

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

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

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

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

(continues on next page)

(continued from previous page)

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

    @property
    def feature_importance(self):
        """Returns importance associated with each feature. Since baseline regressors
↪do not use input features to calculate predictions, returns an array of zeroes.

        Returns:
            np.array (float): an array of zeroes

        """
        return np.zeros(self._num_features)
```

Required fields

- `name` indicating a human-readable name.
- `model_family` - EvalML *model_family* that this component belongs to
- `supported_problem_types` - list of EvalML *problem_types* that this component supports

Model families and problem types include:

[8]:
```
from evalml.model_family import ModelFamily
from evalml.problem_types import ProblemTypes

print("Model Families:\n", [m.value for m in ModelFamily])
print("Problem Types:\n", [p.value for p in ProblemTypes])
```

```
Model Families:
 ['random_forest', 'xgboost', 'lightgbm', 'linear_model', 'catboost', 'extra_trees',
↪'decision_tree', 'baseline', 'none']
Problem Types:
 ['binary', 'multiclass', 'regression']
```

Required methods

- `__init__()` - the `__init__()` method of your estimator will need to call `super().__init__()` and pass three parameters in: a `parameters` dictionary holding the parameters to the component, the `component_obj`, and the `random_state` value.
- `fit()` - the `fit()` method is responsible for fitting your component on training data.
- `predict()` - after fitting a component, the `predict()` method will take in new data and predict new target values. Note: a component must call `fit()` before `predict()`.
- `feature_importance` - `feature_importance` is a Python property that returns a list of importances associated with each feature.

If your estimator handles classification problems it also requires an additonal method:

- `predict_proba()` - this method predicts probability estimates for classification labels

Components Wrapping Third-Party Objects

The `component_obj` parameter is used for wrapping third-party objects and using them in component implementation. If you're using a `component_obj` you will need to define `__init__()` and pass in the relevant object that has also implemented the required methods mentioned above. However, if the `component_obj` does not follow EvalML component conventions, you may need to override methods as needed. Below is an example of EvalML's *LinearRegressor*.

[9]:
```
from sklearn.linear_model import LinearRegression as SKLinearRegression

from evalml.model_family import ModelFamily
from evalml.pipelines.components.estimators import Estimator
from evalml.problem_types import ProblemTypes

class LinearRegressor(Estimator):
    """Linear Regressor."""
    name = "Linear Regressor"
    model_family = ModelFamily.LINEAR_MODEL
    supported_problem_types = [ProblemTypes.REGRESSION]

    def __init__(self, fit_intercept=True, normalize=False, n_jobs=-1, random_state=0,
↪ **kwargs):
        parameters = {
            'fit_intercept': fit_intercept,
            'normalize': normalize,
            'n_jobs': n_jobs
        }
        parameters.update(kwargs)
        linear_regressor = SKLinearRegression(**parameters)
        super().__init__(parameters=parameters,
                         component_obj=linear_regressor,
                         random_state=random_state)

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

Hyperparameter Ranges for AutoML

`hyperparameter_ranges` is a dictionary mapping the parameter name (str) to an allowed range (SkOpt Space) for that parameter. Both lists and `skopt.space.Categorical` values are accepted for categorical spaces.

AutoML will perform a search over the allowed ranges for each parameter to select models which produce optimal performance within those ranges. AutoML gets the allowed ranges for each component from the component's `hyperparameter_ranges` class attribute. Any component parameter you add an entry for in `hyperparameter_ranges` will be included in the AutoML search. If parameters are omitted, AutoML will use the default value in all pipelines.

[10]:
```
from sklearn.linear_model import LinearRegression as SKLinearRegression

from evalml.model_family import ModelFamily
from evalml.pipelines.components.estimators import Estimator
from evalml.problem_types import ProblemTypes

class LinearRegressor(Estimator):
    """Linear Regressor."""
```

(continues on next page)

(continued from previous page)

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

    def __init__(self, fit_intercept=True, normalize=False, n_jobs=-1, random_state=0,
↪ **kwargs):
        parameters = {
            'fit_intercept': fit_intercept,
            'normalize': normalize,
            'n_jobs': n_jobs
        }
        parameters.update(kwargs)
        linear_regressor = SKLinearRegression(**parameters)
        super().__init__(parameters=parameters,
                         component_obj=linear_regressor,
                         random_state=random_state)

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

Expectations for Custom Classification Components

EvalML expects the following from custom classification component implementations:

- Classification targets will range from 0 to n-1 and are integers.
- For classification estimators, the order of predict_proba's columns must match the order of the target, and the column names must be integers ranging from 0 to n-1

4.4 Pipelines

EvalML pipelines represent a sequence of operations to be applied to data, where each operation is either a data transformation or an ML modeling algorithm.

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

Each component and pipeline class supports a set of parameters which configure its behavior. The AutoML search process seeks to find the combination of pipeline structure and pipeline parameters which perform the best on the data.

4.4.1 Class Definition

Pipeline definitions must inherit from the proper pipeline base class, `RegressionPipeline`, `BinaryClassificationPipeline` or `MulticlassClassificationPipeline`. They must also include a `component_graph` list as a class variable containing the sequence of components to be fit and evaluated. The `component_graph` list is used to determine the ordered list of components that should be instantiated when a pipeline instance is created. Each component in `component_graph` can be provided as a reference to the component class for custom components, and as either a string name or as a reference to the component class for components defined in EvalML.

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

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

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

[2]:
```
from evalml.pipelines.components import Transformer

class NewTransformer(Transformer):
    name = 'New Transformer'
    hyperparameter_ranges = {
        "parameter_1":['a', 'b', 'c']
    }

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

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

4.4.2 Pipeline Usage

All pipelines define the following methods:

- `fit` fits each component on the provided training data, in order.
- `predict` computes the predictions of the component graph on the provided data.
- `score` computes the value of *an objective* on the provided data.

[3]:
```
from evalml.demos import load_wine
X, y = load_wine()

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

```
0      class_0
1      class_0
2      class_0
3      class_0
4      class_0
       ...
173    class_2
174    class_2
175    class_2
176    class_2
177    class_2
Length: 178, dtype: object
OrderedDict([('Log Loss Multiclass', 0.04132737017536148)])
```

4.4.3 Custom Name

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

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

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

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

print(CustomPipeline.name)
```

```
A custom pipeline name
```

4.4.4 Override Component Hyperparameter Ranges

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

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

[5]:
```
from skopt.space import Categorical
class CustomPipeline(MulticlassClassificationPipeline):
    component_graph = ['Imputer', 'One Hot Encoder', 'Standard Scaler', 'Logistic␣
↪Regression Classifier']

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

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

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

```
Without custom hyperparameters:
{'Imputer': {'categorical_impute_strategy': ['most_frequent'],␣
↪'numeric_impute_strategy': ['mean', 'median', 'most_frequent']}, 'One Hot Encoder':␣
↪{}, 'Standard Scaler': {}, 'Logistic Regression Classifier': {'penalty': ['l2'],␣
↪'C': Real(low=0.01, high=10, prior='uniform', transform='identity')}}
```

(continues on next page)

(continued from previous page)

```
With custom hyperparameters:
{'Imputer': {'categorical_impute_strategy': ['most_frequent'],␣
↪'numeric_impute_strategy': ['mean', 'median', 'most_frequent']}, 'One Hot Encoder':␣
↪{}, 'Standard Scaler': {}, 'Logistic Regression Classifier': {'penalty': ['l2'],␣
↪'C': Real(low=0.01, high=10, prior='uniform', transform='identity')}}
```

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

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

[6]:
```
CustomPipeline(parameters={'Imputer':{'categorical_impute_strategy': 'most_frequent',
↪'numeric_impute_strategy': 'mean', 'categorical_fill_value': None, 'numeric_fill_
↪value': None}, 'One Hot Encoder':{'top_n': 10, 'categories': None, 'drop': None,
↪'handle_unknown': 'ignore', 'handle_missing': 'error'}, 'Logistic Regression␣
↪Classifier':{'penalty': 'l2', 'C': 1.0, 'n_jobs': -1, 'multi_class': 'auto', 'solver
↪': 'lbfgs'},})
```

4.4.5 Pipeline Parameters

You can also pass in custom parameters, which will then be used when instantiating each component in `component_graph`. The parameters dictionary needs to be in the format of a two-layered dictionary where the key-value pairs are the component name and corresponding component parameters dictionary. The component parameters dictionary consists of (parameter name, parameter values) key-value pairs.

An example will be shown below. The API reference for component parameters can also be found *here*.

[7]:
```
parameters = {
        'Imputer': {
            "categorical_impute_strategy": "most_frequent",
            "numeric_impute_strategy": "median"
        },
        'Logistic Regression Classifier': {
            'penalty': 'l2',
            'C': 1.0,
        }
    }

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

4.4.6 Pipeline Description

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

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

[8]:

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

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

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

Problem Type: multiclass
Model Family: Linear

Pipeline Steps
==============
1. Imputer
         * categorical_impute_strategy : most_frequent
         * numeric_impute_strategy : median
         * categorical_fill_value : None
         * numeric_fill_value : None
2. One Hot Encoder
         * top_n : 10
         * categories : None
         * drop : None
         * handle_unknown : ignore
         * handle_missing : error
3. Standard Scaler
4. Logistic Regression Classifier
         * penalty : l2
         * C : 1.0
         * n_jobs : -1
         * multi_class : auto
         * solver : lbfgs
```

4.4.7 Component Graph

You can use the pipeline's `component_graph` attribute to access a component at a specific index:

[10]:
```
first_component = cp.component_graph[0]
print (first_component.name)
```

```
Imputer
```

Alternatively, you can use `pipeline.get_component(name)` and provide the component name instead (API reference *here*):

[11]:
```
cp.get_component('Imputer')
```

[11]:
```
Imputer(categorical_impute_strategy='most_frequent', numeric_impute_strategy='median',
↪ categorical_fill_value=None, numeric_fill_value=None)
```

4.4.8 Pipeline Estimator

EvalML enforces that the last component of a pipeline is an estimator. You can access this estimator directly by using either `pipeline.component_graph[-1]` or `pipeline.estimator`.

[12]:
```
cp.component_graph[-1]
```

[12]:
```
LogisticRegressionClassifier(penalty='l2', C=1.0, n_jobs=-1, multi_class='auto',␣
↪solver='lbfgs')
```

```
[13]: cp.estimator
```

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

4.4.9 Input Feature Names

After a pipeline is fitted, you can access a pipeline's `input_feature_names` attribute to obtain a dictionary containing a list of feature names passed to each component of the pipeline. This could be especially useful for debugging where a feature might have been dropped or detecting unexpected behavior.

```
[14]: pipeline.input_feature_names
```

```
[14]: {'Imputer': ['alcohol',
    'malic_acid',
    'ash',
    'alcalinity_of_ash',
    'magnesium',
    'total_phenols',
    'flavanoids',
    'nonflavanoid_phenols',
    'proanthocyanins',
    'color_intensity',
    'hue',
    'od280/od315_of_diluted_wines',
    'proline'],
   'Random Forest Classifier': ['alcohol',
    'malic_acid',
    'ash',
    'alcalinity_of_ash',
    'magnesium',
    'total_phenols',
    'flavanoids',
    'nonflavanoid_phenols',
    'proanthocyanins',
    'color_intensity',
    'hue',
    'od280/od315_of_diluted_wines',
    'proline']}
```

4.5 Model Understanding

Simply examining a model's performance metrics is not enough to select a model and promote it for use in a production setting. While developing an ML algorithm, it is important to understand how the model behaves on the data, to examine the key factors influencing its predictions and to consider where it may be deficient. Determination of what "success" may mean for an ML project depends first and foremost on the user's domain expertise.

EvalML includes a variety of tools for understanding models, from graphing utilities to methods for explaining predictions.

** Graphing methods on Jupyter Notebook and Jupyter Lab require ipywidgets to be installed.

** If graphing on Jupyter Lab, jupyterlab-plotly required. To download this, make sure you have npm installed.

4.5.1 Graphing Utilities

First, let's train a pipeline on some data.

[1]:
```
import evalml

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

X, y = evalml.demos.load_breast_cancer()

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

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

Feature Importance

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

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

[2]:

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

We can also create a bar plot of the feature importances

[3]:
```
pipeline.graph_feature_importance()
```

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

Permutation Importance

We can also compute and plot the permutation importance of the pipeline.

[4]:
```
from evalml.model_understanding.graphs import calculate_permutation_importance
calculate_permutation_importance(pipeline, X, y, 'log loss binary')
```

[4]:

	feature	importance
0	worst perimeter	0.078033
1	worst radius	0.074341
2	worst concave points	0.068313
3	worst area	0.067733
4	mean concave points	0.041261
5	worst concavity	0.037533
6	mean concavity	0.036664
7	area error	0.035838
8	mean perimeter	0.025783
9	mean area	0.025203
10	worst texture	0.016211
11	perimeter error	0.011738
12	mean texture	0.011716
13	radius error	0.010910
14	mean radius	0.010775
15	worst compactness	0.008322
16	worst smoothness	0.008281
17	mean smoothness	0.005707
18	worst symmetry	0.004454
19	worst fractal dimension	0.003889
20	concavity error	0.003858
21	compactness error	0.003572
22	concave points error	0.003449
23	mean compactness	0.003173
24	smoothness error	0.003172
25	fractal dimension error	0.002618
26	texture error	0.002533
27	mean fractal dimension	0.002228
28	symmetry error	0.002126
29	mean symmetry	0.001786

[5]:
```
from evalml.model_understanding.graphs import graph_permutation_importance
graph_permutation_importance(pipeline, X, y, 'log loss binary')
```

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

Partial Dependence Plots

We can calculate the partial dependence plots for a feature.

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

[6]:
```
    feature_values  partial_dependence
0         9.498540            0.371141
1         9.610488            0.371141
2         9.722436            0.371141
3         9.834384            0.371141
4         9.946332            0.371141
..             ...                 ...
95       20.133608            0.399560
96       20.245556            0.399560
97       20.357504            0.399560
98       20.469452            0.399560
99       20.581400            0.399560

[100 rows x 2 columns]
```

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

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

Confusion Matrix

For binary or multiclass classification, we can view a confusion matrix of the classifier's predictions. In the DataFrame output of `confusion_matrix()`, the column header represents the predicted labels while row header represents the actual labels.

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

[8]:
```
              benign  malignant
benign      1.000000   0.000000
malignant   0.009434   0.990566
```

[9]:
```
from evalml.model_understanding.graphs import graph_confusion_matrix
y_pred = pipeline.predict(X)
graph_confusion_matrix(y, y_pred)
```

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

Precision-Recall Curve

For binary classification, we can view the precision-recall curve of the pipeline.

[10]:
```
from evalml.model_understanding.graphs import graph_precision_recall_curve
# get the predicted probabilities associated with the "true" label
y_encoded = y.map({'benign': 0, 'malignant': 1})
```

(continues on next page)

(continued from previous page)

```
y_pred_proba = pipeline.predict_proba(X)["malignant"]
graph_precision_recall_curve(y_encoded, y_pred_proba)
```

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

ROC Curve

For binary and multiclass classification, we can view the Receiver Operating Characteristic (ROC) curve of the pipeline.

[11]:
```
from evalml.model_understanding.graphs import graph_roc_curve
# get the predicted probabilities associated with the "malignant" label
y_pred_proba = pipeline.predict_proba(X)["malignant"]
graph_roc_curve(y_encoded, y_pred_proba)
```

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

The ROC curve can also be generated for multiclass classification problems. For multiclass problems, the graph will show a one-vs-many ROC curve for each class.

[12]:
```
class RFMulticlassClassificationPipeline(evalml.pipelines.
↪MulticlassClassificationPipeline):
    component_graph = ['Simple Imputer', 'Random Forest Classifier']

X_multi, y_multi = evalml.demos.load_wine()

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

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

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

Binary Objective Score vs. Threshold Graph

Some binary classification objectives (objectives that have `score_needs_proba` set to False) are sensitive to a decision threshold. For those objectives, we can obtain and graph the scores for thresholds from zero to one, calculated at evenly-spaced intervals determined by `steps`.

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

[13]:

	threshold	score
0	0.00	0.542894
1	0.01	0.750442
2	0.02	0.815385
3	0.03	0.848000

(continues on next page)

(continued from previous page)

```
4          0.04  0.874227
..          ...       ...
96         0.96  0.854054
97         0.97  0.835165
98         0.98  0.805634
99         0.99  0.722892
100        1.00  0.000000

[101 rows x 2 columns]
```

[14]:
```
from evalml.model_understanding.graphs import graph_binary_objective_vs_threshold
graph_binary_objective_vs_threshold(pipeline, X, y, 'f1', steps=100)
```

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

4.5.2 Explaining Predictions

Explaining Individual Predictions

We can explain why the model made an individual prediction with the *explain_prediction* function. This will use the Shapley Additive Explanations (SHAP) algorithms to identify the top features that explain the predicted value.

This function can explain both classification and regression models - all you need to do is provide the pipeline, the input features (must correspond to one row of the input data) and the training data. The function will return a table that you can print summarizing the top 3 most positive and negative contributing features to the predicted value.

In the example below, we explain the prediction for the third data point in the data set. We see that the `worst concave points` feature increased the estimated probability that the tumor is malignant by 20% while the `worst radius` feature decreased the probability the tumor is malignant by 5%.

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

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

```
    Feature Name       Feature Value   Contribution to Prediction   SHAP Value
=========================================================================================
worst concave points       0.26                    ++                    0.20
mean concave points        0.11                    +                     0.11
   mean concavity          0.24                    +                     0.08
     worst area           567.70                   -                    -0.03
  worst perimeter          98.87                   -                    -0.05
    worst radius           14.91                   -                    -0.05
```

The interpretation of the table is the same for regression problems - but the SHAP value now corresponds to the change in the estimated value of the dependent variable rather than a change in probability. For multiclass classification problems, a table will be output for each possible class.

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

Explaining Multiple Predictions

When debugging machine learning models, it is often useful to analyze the best and worst predictions the model made. The *explain_predictions_best_worst* function can help us with this.

This function will display the output of *explain_prediction* for the best 2 and worst 2 predictions. By default, the best and worst predictions are determined by the absolute error for regression problems and cross entropy for classification problems.

We can specify our own ranking function by passing in a function to the `metric` parameter. This function will be called on `y_true` and `y_pred`. By convention, lower scores are better.

At the top of each table, we can see the predicted probabilities, target value, and error on that prediction. For a regression problem, we would see the predicted value instead of predicted probabilities.

[16]:
```
from evalml.model_understanding.prediction_explanations import explain_predictions_
↪best_worst

report = explain_predictions_best_worst(pipeline=pipeline, input_features=X, y_true=y,
                                        include_shap_values=True, num_to_explain=2)

print(report)
```

```
RFBinary Classification Pipeline

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

        Best 1 of 2

                Predicted Probabilities: [benign: 0.0, malignant: 1.0]
                Predicted Value: malignant
                Target Value: malignant
                Cross Entropy: 0.0

                     Feature Name         Feature Value        Contribution to␣
↪SHAP Value
                                                                 Prediction
                ␣
↪=============================================================================
                    worst perimeter          155.30                    +␣
↪   0.10
                     worst radius             23.14                    +␣
↪   0.08
                 worst concave points         0.17                    +␣
↪   0.08
                worst fractal dimension       0.09                    -␣
↪  -0.00
                   compactness error          0.04                    -␣
↪  -0.00
                    worst symmetry            0.22                    -␣
↪  -0.00


        Best 2 of 2

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

(continues on next page)

(continued from previous page)

```
                Cross Entropy: 0.0

                    Feature Name       Feature Value   Contribution to Prediction
↪SHAP Value
                ⌊
↪================================================================================
                  worst perimeter         166.10                     +
↪ 0.10
                     worst radius          25.45                     +
↪ 0.08
                worst concave points       0.22                     +
↪ 0.08
                 compactness error         0.03                     -
↪-0.00
                 worst compactness         0.21                     -
↪-0.00
                   worst symmetry          0.21                     -
↪-0.00


        Worst 1 of 2

                Predicted Probabilities: [benign: 0.552, malignant: 0.448]
                Predicted Value: benign
                Target Value: malignant
                Cross Entropy: 0.802

                    Feature Name       Feature Value   Contribution to Prediction
↪SHAP Value
                ⌊
↪================================================================================
                  smoothness error         0.00                     +
↪ 0.04
                     mean texture          21.58                     +
↪ 0.03
                    worst texture          30.25                     +
↪ 0.02
                worst concave points       0.11                     -
↪-0.02
                     worst radius          15.93                     -
↪-0.03
                mean concave points        0.02                     -
↪-0.03


        Worst 2 of 2

                Predicted Probabilities: [benign: 0.788, malignant: 0.212]
                Predicted Value: benign
                Target Value: malignant
                Cross Entropy: 1.55

                    Feature Name       Feature Value   Contribution to Prediction
↪SHAP Value
                ⌊
↪================================================================================
                    worst texture          33.37                     +
↪ 0.05
```

(continues on next page)

(continued from previous page)

```
                    mean texture           22.47                    +                ␣
↪ 0.03
                  symmetry error           0.02                     +                ␣
↪ 0.01
                worst concave points       0.09                     -                ␣
↪-0.04
                    worst radius           14.49                    -                ␣
↪-0.05
                  worst perimeter          92.04                    -                ␣
↪-0.06
```

We use a custom metric (hinge loss) for selecting the best and worst predictions. See this example:

```
import numpy as np

def hinge_loss(y_true, y_pred_proba):

    probabilities = np.clip(y_pred_proba.iloc[:, 1], 0.001, 0.999)
    y_true[y_true == 0] = -1

    return np.clip(1 - y_true * np.log(probabilities / (1 - probabilities)), a_min=0,␣
↪a_max=None)

report = explain_predictions_best_worst(pipeline=pipeline, input_features=X, y_true=y,
                                        include_shap_values=True, num_to_explain=5,␣
↪metric=hinge_loss)

print(report)
```

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

[17]:
```
from evalml.model_understanding.prediction_explanations import explain_predictions

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

```
RFBinary Classification Pipeline

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

        1 of 3

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

(continues on next page)

(continued from previous page)

```
                compactness error         0.05                    -
↪-0.00
                  worst texture           17.33                   -
↪-0.03
                   mean texture           10.38                   -
↪-0.05


       2 of 3

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


       3 of 3

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

Changing Output Formats

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

[18]:
```
import json
report = explain_predictions_best_worst(pipeline=pipeline, input_features=X, y_true=y,
                                        num_to_explain=1, include_shap_values=True,␣
↪output_format="dict")
print(json.dumps(report, indent=2))
```


```
{
  "explanations": [
    {
      "rank": {
        "prefix": "best",
        "index": 1
      },
      "predicted_values": {
        "probabilities": {
          "benign": 0.0,
          "malignant": 1.0
        },
        "predicted_value": "malignant",
        "target_value": "malignant",
        "error_name": "Cross Entropy",
        "error_value": 9.95074382629983e-05
      },
      "explanations": [
        {
          "feature_names": [
            "worst perimeter",
            "worst radius",
            "worst concave points",
            "worst fractal dimension",
            "compactness error",
            "worst symmetry"
          ],
          "feature_values": [
            155.3,
            23.14,
            0.1721,
            0.093,
            0.03634,
            0.216
          ],
          "qualitative_explanation": [
            "+",
            "+",
            "+",
            "-",
            "-",
            "-"
          ],
          "quantitative_explanation": [
            0.09988982304983156,
            0.08240174808629956,
            0.07868368954615064,
            -0.001381925705526203,
            -0.0022100542079298295,
            -0.00455357441134733
          ],
          "class_name": "malignant"
```


(continues on next page)

(continued from previous page)


```
        }
      ]
    },
    {
      "rank": {
        "prefix": "worst",
        "index": 1
      },
      "predicted_values": {
        "probabilities": {
          "benign": 0.788,
          "malignant": 0.212
        },
        "predicted_value": "benign",
        "target_value": "malignant",
        "error_name": "Cross Entropy",
        "error_value": 1.5499050281608746
      },
      "explanations": [
        {
          "feature_names": [
            "worst texture",
            "mean texture",
            "symmetry error",
            "worst concave points",
            "worst radius",
            "worst perimeter"
          ],
          "feature_values": [
            33.37,
            22.47,
            0.01647,
            0.09331,
            14.49,
            92.04
          ],
          "qualitative_explanation": [
            "+",
            "+",
            "+",
            "-",
            "-",
            "-"
          ],
          "quantitative_explanation": [
            0.05245422607466413,
            0.03035933540832274,
            0.013117759717201452,
            -0.04174884967530769,
            -0.0491285663898271,
            -0.05666940833106337
          ],
          "class_name": "malignant"
        }
      ]
    }
  ]
```


(continues on next page)

(continued from previous page)

```
}
```

4.6 Data Checks

EvalML provides data checks to help guide you in achieving the highest performing model. These utility functions help deal with problems such as overfitting, abnormal data, and missing data. These data checks can be found under `evalml/data_checks`. Below we will cover examples such as abnormal and missing data data checks.

4.6.1 Missing Data

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

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

from evalml.data_checks import HighlyNullDataCheck

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

null_check = HighlyNullDataCheck(pct_null_threshold=0.8)

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

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

4.6.2 Abnormal Data

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

ID Columns

ID columns in your dataset provide little to no benefit to a machine learning pipeline as the pipeline cannot extrapolate useful information from unique identifiers. Thus, `IDColumnsDataCheck()` reminds you if these columns exists. In the given example, 'user_number' and 'id' columns are both identified as potentially being unique identifiers that should be removed.

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

X = pd.DataFrame([[0, 53, 6325, 5],[1, 90, 6325, 10],[2, 90, 18, 20]], columns=['user_
↪number', 'cost', 'revenue', 'id'])
id_col_check = IDColumnsDataCheck(id_threshold=0.9)

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

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

4.6.3 Outliers

Outliers are observations that differ significantly from other observations in the same sample. Many machine learning pipelines suffer in performance if outliers are not dropped from the training set as they are not representative of the data. `OutliersDataCheck()` uses Isolation Forests to notify you if a sample can be considered an outlier.

Below we generate a random dataset with some outliers.

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

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

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

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

outliers_check = OutliersDataCheck()

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

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

4.6.4 Writing Your Own Data Check

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

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

class ZeroVarianceDataCheck(DataCheck):
    def validate(self, X, y):
        if not isinstance(X, pd.DataFrame):
```

(continues on next page)

(continued from previous page)

```
            X = pd.DataFrame(X)
        warning_msg = "Column '{}' has zero variance"
        return [DataCheckError(warning_msg.format(column), self.name) for column in X.
↪columns if len(X[column].unique()) == 1]
```

4.7 Utilities

4.7.1 Configuring Logging

EvalML uses the standard python logging package. By default, EvalML will log `INFO`-level logs and higher (warnings, errors and critical) to stdout, and will log everything to `evalml_debug.log` in the current working directory.

If you want to change the location of the logfile, before import, set the `EVALML_LOG_FILE` environment variable to specify a filename within an existing directory in which you have write permission. If you want to disable logging to the logfile, set `EVALML_LOG_FILE` to be empty. If the environment variable is set to an invalid location, EvalML will print a warning message to stdout and will not create a log file.

4.8 FAQ

4.8.1 Q: What is the difference between EvalML and other AutoML libraries?

EvalML optimizes machine learning pipelines on *custom practical objectives* instead of vague machine learning loss functions so that it will find the best pipelines for your specific needs. Furthermore, EvalML *pipelines* are able to take in all kinds of data (missing values, categorical, etc.) as long as the data are in a single table. EvalML also allows you to build your own pipelines with existing or custom components so you can have more control over the AutoML process. Moreover, EvalML also provides you with support in the form of *data checks* to ensure that you are aware of potential issues your data may cause with machine learning algorithms.

4.8.2 Q: How does EvalML handle missing values?

EvalML contains imputation components in its pipelines so that missing values are taken care of. EvalML optimizes over different types of imputation to search for the best possible pipeline. You can find more information about components *here* and in the API reference *here*.

4.8.3 Q: How does EvalML handle categorical encoding?

EvalML provides a *one-hot-encoding component* in its pipelines for categorical variables. EvalML plans to support other encoders in the future.

4.8.4 Q: How does EvalML handle feature selection?

EvalML currently utilizes scikit-learn's SelectFromModel with a Random Forest classifier/regressor to handle feature selection. EvalML plans on supporting more feature selectors in the future. You can find more information in the API reference *here*.

4.8.5 Q: How is feature importance calculated?

Feature importance depends on the estimator used. Variable coefficients are used for regression-based estimators (Logistic Regression and Linear Regression) and Gini importance is used for tree-based estimators (Random Forest and XGBoost).

4.8.6 Q: How does hyperparameter tuning work?

EvalML tunes hyperparameters for its pipelines through Bayesian optimization. In the future we plan to support more optimization techniques such as random search.

4.8.7 Q: Can I create my own objective metric?

Yes you can! You can *create your own custom objective* so that EvalML optimizes the best model for your needs.

4.8.8 Q: How does EvalML avoid overfitting?

EvalML provides *data checks* to combat overfitting. Such data checks include detecting label leakage, unstable pipelines, hold-out datasets and cross validation. EvalML defaults to using Stratified K-Fold cross-validation for classification problems and K-Fold cross-validation for regression problems but allows you to utilize your own cross-validation methods as well.

4.8.9 Q: Can I create my own pipeline for EvalML?

Yes! EvalML allows you to create *custom pipelines* using modular components. This allows you to customize EvalML pipelines for your own needs or for AutoML.

4.8.10 Q: Does EvalML work with X algorithm?

EvalML is constantly improving and adding new components and will allow your own algorithms to be used as components in our pipelines.

CHAPTER

FIVE

API REFERENCE

5.1 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.
load_churn	Load credit card fraud dataset.

5.1.1 evalml.demos.load_fraud

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

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

Parameters

- **n_rows** (*int*) – Number of rows from the dataset to return
- **verbose** (*bool*) – Whether to print information about features and labels

Returns X, y

Return type pd.DataFrame, pd.Series

5.1.2 evalml.demos.load_wine

evalml.demos.**load_wine**()

Load wine dataset. Multiclass problem

Returns X, y

Return type pd.DataFrame, pd.Series

5.1.3 evalml.demos.load_breast_cancer

evalml.demos.**load_breast_cancer**()

Load breast cancer dataset. Binary classification problem.

Returns X, y

Return type pd.DataFrame, pd.Series

5.1.4 evalml.demos.load_diabetes

evalml.demos.**load_diabetes**()

Load diabetes dataset. Regression problem

Returns X, y

Return type pd.DataFrame, pd.Series

5.1.5 evalml.demos.load_churn

evalml.demos.**load_churn**(*n_rows=None*, *verbose=True*)

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

Parameters

- **n_rows** (*int*) – Number of rows from the dataset to return
- **verbose** (*bool*) – Whether to print information about features and labels

Returns X, y

Return type pd.DataFrame, pd.Series

5.2 Preprocessing

Utilities to preprocess data before using evalml.

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

5.2.1 evalml.preprocessing.drop_nan_target_rows

evalml.preprocessing.**drop_nan_target_rows**(*X*, *y*)

Drops rows in X and y when row in the target y has a value of NaN.

Parameters

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

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

Return type pd.DataFrame

5.2.2 evalml.preprocessing.target_distribution

evalml.preprocessing.**target_distribution**(*targets*)

Get the target distributions.

Parameters **targets** (*pd.Series*) – Target data

Returns Target data and their frequency distribution as percentages.

Return type pd.Series

5.2.3 evalml.preprocessing.load_data

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

Load features and target from file.

Parameters

- **path** (*str*) – Path to file or a http/ftp/s3 URL
- **index** (*str*) – Column for index
- **target** (*str*) – Column for target
- **n_rows** (*int*) – Number of rows to return
- **drop** (*list*) – List of columns to drop
- **verbose** (*bool*) – If True, prints information about features and target

Returns features and target

Return type pd.DataFrame, pd.Series

5.2.4 evalml.preprocessing.number_of_features

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

Get the number of features for specific dtypes.

Parameters **dtypes** (*pd.Series*) – dtypes to get the number of features for

Returns dtypes and the number of features for each input type

Return type pd.Series

5.2.5 evalml.preprocessing.split_data

evalml.preprocessing.**split_data**(*X*, *y*, *regression=False*, *test_size=0.2*, *random_state=None*)

Splits data into train and test sets.

Parameters

- **X** (*pd.DataFrame or np.array*) – Data of shape [n_samples, n_features]
- **y** (*pd.Series*) – Target data of length [n_samples]
- **regression** (*bool*) – If true, do not use stratified split
- **test_size** (*float*) – Percent of train set to holdout for testing
- **random_state** (*int, np.random.RandomState*) – Seed for the random number generator

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

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

5.3 AutoML

5.3.1 AutoML Search Classes

AutoMLSearch	Automated Pipeline search.

evalml.automl.AutoMLSearch

evalml.automl.automl_search.AutoMLSearch

class evalml.automl.**AutoMLSearch**(*problem_type=None*, *objective='auto'*, *max_pipelines=None*, *max_iterations=None*, *max_time=None*, *patience=None*, *tolerance=None*, *data_split=None*, *allowed_pipelines=None*, *allowed_model_families=None*, *start_iteration_callback=None*, *add_result_callback=None*, *additional_objectives=None*, *random_state=0*, *n_jobs=-1*, *tuner_class=None*, *verbose=True*, *optimize_thresholds=False*, *_max_batches=None*)

Automated Pipeline search.

Methods

__init__	Automated pipeline search
add_to_rankings	Fits and evaluates a given pipeline then adds the results to the automl rankings with the requirement that automl search has been run.
describe_pipeline	Describe a pipeline
get_pipeline	Given the ID of a pipeline training result, returns an untrained instance of the specified pipeline initialized with the parameters used to train that pipeline during automl search.
load	Loads AutoML object at file path
save	Saves AutoML object at file path
search	Find the best pipeline for the data set.

evalml.automl.AutoMLSearch.__init__

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

Automated pipeline search

Parameters

- **problem_type** (*str or* ProblemTypes) – Choice of 'regression', 'binary', or 'multiclass', depending on the desired problem type.
- **objective** (*str,* ObjectiveBase) – The objective to optimize for. Used to propose and rank pipelines, but not for optimizing each pipeline during fit-time. When set to 'auto', chooses:
  - LogLossBinary for binary classification problems,
  - LogLossMulticlass for multiclass classification problems, and
  - R2 for regression problems.
- **max_pipelines** (*int*) – Will be deprecated in the next release. 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_iterations** (*int*) – Maximum number of iterations to search. If max_iterations and max_time is not set, then max_iterations will default to max_iterations of 5.
- **max_time** (*int, str*) – Maximum time to search for pipelines. This will not start a new pipeline search after the duration has elapsed. If it is an integer, then the time will be in seconds. For strings, time can be specified as seconds, minutes, or hours.
- **patience** (*int*) – Number of iterations without improvement to stop search early. Must be positive. If None, early stopping is disabled. Defaults to None.
- **tolerance** (*float*) – Minimum percentage difference to qualify as score improvement for early stopping. Only applicable if patience is not None. Defaults to None.
- **allowed_pipelines** (*list(class)*) – A list of PipelineBase subclasses indicating the pipelines allowed in the search. The default of None indicates all pipelines for this problem type are allowed. Setting this field will cause allowed_model_families to be ignored.
- **allowed_model_families** (*list(str,* ModelFamily*)*) – The model families to search. The default of None searches over all model families. Run evalml.pipelines.components.utils.allowed_model_families("binary") to see options. Change *binary* to *multiclass* or *regression* depending on the problem type. Note that if allowed_pipelines is provided, this parameter will be ignored.
- **data_split** (*sklearn.model_selection.BaseCrossValidator*) – data splitting method to use. Defaults to StratifiedKFold.
- **tuner_class** – the tuner class to use. Defaults to scikit-optimize tuner
- **start_iteration_callback** (*callable*) – function called before each pipeline training iteration. Passed three parameters: pipeline_class, parameters, and the AutoMLSearch object.

- **add_result_callback** (*callable*) – function called after each pipeline training iteration. Passed three parameters: A dictionary containing the training results for the new pipeline, an untrained_pipeline containing the parameters used during training, and the AutoMLSearch object.
- **additional_objectives** (*list*) – Custom set of objectives to score on. Will override default objectives for problem type if not empty.
- **random_state** (*int, np.random.RandomState*) – The random seed/state. Defaults to 0.
- **n_jobs** (*int or None*) – Non-negative integer describing level of parallelism used for pipelines. None and 1 are equivalent. If set to -1, all CPUs are used. For n_jobs below -1, (n_cpus + 1 + n_jobs) are used.
- **verbose** (*boolean*) – If True, turn verbosity on. Defaults to True
- **_max_batches** (*int*) – The maximum number of batches of pipelines to search. Parameters max_time, and max_iterations have precedence over stopping the search.

evalml.automl.AutoMLSearch.add_to_rankings

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

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

Parameters

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

evalml.automl.AutoMLSearch.describe_pipeline

AutoMLSearch.**describe_pipeline**(*pipeline_id*, *return_dict=False*)

Describe a pipeline

Parameters

- **pipeline_id** (*int*) – pipeline to describe
- **return_dict** (*bool*) – If True, return dictionary of information about pipeline. Defaults to False.

Returns Description of specified pipeline. Includes information such as type of pipeline components, problem, training time, cross validation, etc.

evalml.automl.AutoMLSearch.get_pipeline

AutoMLSearch.**get_pipeline**(*pipeline_id*, *random_state=0*)

Given the ID of a pipeline training result, returns an untrained instance of the specified pipeline initialized with the parameters used to train that pipeline during automl search.

Parameters

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

Returns untrained pipeline instance associated with the provided ID

Return type *PipelineBase*

evalml.automl.AutoMLSearch.load

static AutoMLSearch.**load**(*file_path*)

Loads AutoML object at file path

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

Returns AutoSearchBase object

evalml.automl.AutoMLSearch.save

AutoMLSearch.**save**(*file_path*, *pickle_protocol=4*)

Saves AutoML object at file path

Parameters

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

Returns None

evalml.automl.AutoMLSearch.search

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

Find the best pipeline for the data set.

Parameters

- **X** (*pd.DataFrame*) – the input training data of shape [n_samples, n_features]
- **y** (*pd.Series*) – the target training data of length [n_samples]
- **feature_types** (*list, optional*) – list of feature types, either numerical or categorical. Categorical features will automatically be encoded
- **show_iteration_plot** (*boolean, True*) – Shows an iteration vs. score plot in Jupyter notebook. Disabled by default in non-Jupyter enviroments.
- **data_checks** (DataChecks, *list(Datacheck), str, None*) – A collection of data checks to run before automl search. If data checks produce any errors, an exception will be thrown before the search begins. If "disabled" or None, no data checks will be done. If set to "auto", DefaultDataChecks will be done. Default value is set to "auto".

Returns self

Attributes

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

5.3.2 AutoML Utils

get_default_primary_search_objective	Get the default primary search objective for a problem type.

evalml.automl.get_default_primary_search_objective

evalml.automl.**get_default_primary_search_objective**(*problem_type*)

Get the default primary search objective for a problem type.

Parameters **problem_type** (*str or ProblemType*) – problem type of interest.

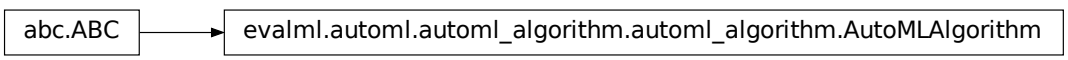
Returns primary objective instance for the problem type.

Return type *ObjectiveBase*

5.3.3 AutoML Algorithm Classes

AutoMLAlgorithm	Base class for the automl algorithms which power evalml.
IterativeAlgorithm	An automl algorithm which first fits a base round of pipelines with default parameters, then does a round of parameter tuning on each pipeline in order of performance.

evalml.automl.automl_algorithm.AutoMLAlgorithm



class evalml.automl.automl_algorithm.**AutoMLAlgorithm**(*allowed_pipelines=None*, *max_iterations=None*, *tuner_class=None*, *random_state=0*)

Base class for the automl algorithms which power evalml.

Methods

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

evalml.automl.automl_algorithm.AutoMLAlgorithm.__init__

AutoMLAlgorithm.**__init__**(*allowed_pipelines=None*, *max_iterations=None*, *tuner_class=None*, *random_state=0*)

This class represents an automated machine learning (AutoML) algorithm. It encapsulates the decision-making logic behind an automl search, by both deciding which pipelines to evaluate next and by deciding what set of parameters to configure the pipeline with.

To use this interface, you must define a next_batch method which returns the next group of pipelines to evaluate on the training data. That method may access state and results recorded from the previous batches, although that information is not tracked in a general way in this base class. Overriding add_result is a convenient way to record pipeline evaluation info if necessary.

Parameters

- **allowed_pipelines** (*list(class)*) – A list of PipelineBase subclasses indicating the pipelines allowed in the search. The default of None indicates all pipelines for this problem type are allowed.
- **max_iterations** (*int*) – The maximum number of iterations to be evaluated.
- **tuner_class** (*class*) – A subclass of Tuner, to be used to find parameters for each pipeline. The default of None indicates the SKOptTuner will be used.
- **random_state** (*int, np.random.RandomState*) – The random seed/state. Defaults to 0.

evalml.automl.automl_algorithm.AutoMLAlgorithm.add_result

AutoMLAlgorithm.**add_result**(*score_to_minimize*, *pipeline*)

Register results from evaluating a pipeline

Parameters

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

evalml.automl.automl_algorithm.AutoMLAlgorithm.next_batch

AutoMLAlgorithm.**next_batch**()

Get the next batch of pipelines to evaluate

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

Return type list(*PipelineBase*)

Attributes

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

evalml.automl.automl_algorithm.IterativeAlgorithm

class evalml.automl.automl_algorithm.**IterativeAlgorithm**(*allowed_pipelines=None*, *max_iterations=None*, *tuner_class=None*, *random_state=0*, *pipelines_per_batch=5*, *n_jobs=-1*, *number_features=None*)

An automl algorithm which first fits a base round of pipelines with default parameters, then does a round of parameter tuning on each pipeline in order of performance.

Methods

__init__	An automl algorithm which first fits a base round of pipelines with default parameters, then does a round of parameter tuning on each pipeline in order of performance.
add_result	Register results from evaluating a pipeline
next_batch	Get the next batch of pipelines to evaluate

evalml.automl.automl_algorithm.IterativeAlgorithm.__init__

IterativeAlgorithm.**__init__**(*allowed_pipelines=None*, *max_iterations=None*, *tuner_class=None*, *random_state=0*, *pipelines_per_batch=5*, *n_jobs=-1*, *number_features=None*)

An automl algorithm which first fits a base round of pipelines with default parameters, then does a round of parameter tuning on each pipeline in order of performance.

Parameters

- **allowed_pipelines** (*list (class)*) – A list of PipelineBase subclasses indicating the pipelines allowed in the search. The default of None indicates all pipelines for this problem type are allowed.
- **max_iterations** (*int*) – The maximum number of iterations to be evaluated.
- **tuner_class** (*class*) – A subclass of Tuner, to be used to find parameters for each pipeline. The default of None indicates the SKOptTuner will be used.
- **random_state** (*int, np.random.RandomState*) – The random seed/state. Defaults to 0.
- **pipelines_per_batch** (*int*) – the number of pipelines to be evaluated in each batch, after the first batch.
- **n_jobs** (*int or None*) – Non-negative integer describing level of parallelism used for pipelines.
- **number_features** (*int*) – The number of columns in the input features.

evalml.automl.automl_algorithm.IterativeAlgorithm.add_result

IterativeAlgorithm.**add_result**(*score_to_minimize*, *pipeline*)

Register results from evaluating a pipeline

Parameters

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

evalml.automl.automl_algorithm.IterativeAlgorithm.next_batch

IterativeAlgorithm.**next_batch**()

Get the next batch of pipelines to evaluate

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

Return type list(*PipelineBase*)

Attributes

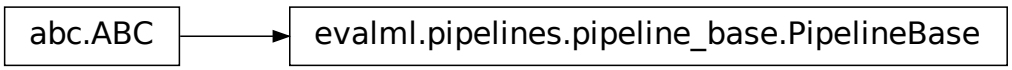
batch_number	Returns the number of batches which have been recommended so far.
pipeline_number	Returns the number of pipelines which have been recommended so far.

5.4 Pipelines

5.4.1 Pipeline Base Classes

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

evalml.pipelines.PipelineBase



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

Base class for all pipelines.

name = 'Pipeline Base'

custom_name = None

problem_type = None

Instance attributes

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

Methods:

__init__	Machine learning pipeline made out of transformers and a estimator.
clone	Constructs a new pipeline with the same parameters and components.
compute_estimator_features	Transforms the data by applying all pre-processing components.
describe	Outputs pipeline details including component parameters

Continued on next page

Table 14 – continued from previous page

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

evalml.pipelines.PipelineBase.__init__

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

Machine learning pipeline made out of transformers and a estimator.

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

Parameters

- **parameters** (*dict*) – Dictionary with component names as keys and dictionary of that component's parameters as values. An empty dictionary { } implies using all default values for component parameters.
- **random_state** (*int, np.random.RandomState*) – The random seed/state. Defaults to 0.

evalml.pipelines.PipelineBase.clone

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

Constructs a new pipeline with the same parameters and components.

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

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

evalml.pipelines.PipelineBase.compute_estimator_features

PipelineBase.**compute_estimator_features**(*X*)

Transforms the data by applying all pre-processing components.

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

Returns pd.DataFrame - New transformed features.

evalml.pipelines.PipelineBase.describe

PipelineBase.**describe**()

Outputs pipeline details including component parameters

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

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

Return type dict

evalml.pipelines.PipelineBase.fit

PipelineBase.**fit**(*X*, *y*)
Build a model

Parameters

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

Returns self

evalml.pipelines.PipelineBase.get_component

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

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

Returns Component to return

Return type Component

evalml.pipelines.PipelineBase.graph

PipelineBase.**graph**(*filepath=None*)
Generate an image representing the pipeline graph

Parameters **filepath** (*str, optional*) – Path to where the graph should be saved. If set to None (as by default), the graph will not be saved.

Returns Graph object that can be directly displayed in Jupyter notebooks.

Return type graphviz.Digraph

evalml.pipelines.PipelineBase.graph_feature_importance

PipelineBase.**graph_feature_importance**(*importance_threshold=0*)
Generate a bar graph of the pipeline's feature importance

Parameters **importance_threshold** (*float, optional*) – If provided, graph features with a permutation importance whose absolute value is larger than importance_threshold. Defaults to zero.

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

evalml.pipelines.PipelineBase.load

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

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

Returns PipelineBase object

evalml.pipelines.PipelineBase.predict

PipelineBase.**predict**(*X*, *objective=None*)
Make predictions using selected features.

Parameters

- **X** (*pd.DataFrame or np.array*) – Data of shape [n_samples, n_features]
- **objective** (*Object or string*) – The objective to use to make predictions

Returns Predicted values.

Return type pd.Series

evalml.pipelines.PipelineBase.save

PipelineBase.**save**(*file_path*, *pickle_protocol=4*)
Saves pipeline at file path

Parameters

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

Returns None

evalml.pipelines.PipelineBase.score

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

Parameters

- **X** (*pd.DataFrame or np.array*) – Data of shape [n_samples, n_features]
- **y** (*pd.Series*) – Target data of length [n_samples]
- **objectives** (*list*) – Non-empty list of objectives to score on

Returns Ordered dictionary of objective scores

Return type dict

evalml.pipelines.ClassificationPipeline

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

Pipeline subclass for all classification pipelines.

name = 'Classification Pipeline'

custom_name = None

problem_type = None

Instance attributes

classes_	Gets the class names for the problem.
feature_importance	Return an attribute of instance, which is of type owner.
parameters	Returns parameter dictionary for this pipeline

Methods:

__init__	Machine learning classification pipeline made out of transformers and a classifier.
clone	Constructs a new pipeline with the same parameters and components.
compute_estimator_features	Transforms the data by applying all pre-processing components.
describe	Outputs pipeline details including component parameters
fit	Build a classification model. For string and categorical targets, classes are sorted
get_component	Returns component by name
graph	Generate an image representing the pipeline graph
graph_feature_importance	Generate a bar graph of the pipeline's feature importance
load	Loads pipeline at file path
predict	Make predictions using selected features.
predict_proba	Make probability estimates for labels.
save	Saves pipeline at file path
score	Evaluate model performance on objectives

evalml.pipelines.ClassificationPipeline.__init__

ClassificationPipeline.**__init__**(*parameters*, *random_state=0*)

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

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

Parameters

- **parameters** (*dict*) – Dictionary with component names as keys and dictionary of that component's parameters as values. An empty dictionary { } implies using all default values for component parameters.
- **random_state** (*int, np.random.RandomState*) – The random seed/state. Defaults to 0.

evalml.pipelines.ClassificationPipeline.clone

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

Constructs a new pipeline with the same parameters and components.

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

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

evalml.pipelines.ClassificationPipeline.compute_estimator_features

ClassificationPipeline.**compute_estimator_features**(*X*)

Transforms the data by applying all pre-processing components.

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

Returns pd.DataFrame - New transformed features.

evalml.pipelines.ClassificationPipeline.describe

ClassificationPipeline.**describe**()

Outputs pipeline details including component parameters

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

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

Return type dict

evalml.pipelines.ClassificationPipeline.fit

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

Build a classification model. For string and categorical targets, classes are sorted by sorted(set(y)) and then are mapped to values between 0 and n_classes-1.

Parameters

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

Returns self

evalml.pipelines.ClassificationPipeline.get_component

ClassificationPipeline.**get_component**(*name*)

Returns component by name

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

Returns Component to return

Return type Component

evalml.pipelines.ClassificationPipeline.graph

ClassificationPipeline.**graph**(*filepath=None*)

Generate an image representing the pipeline graph

Parameters **filepath** (*str, optional*) – Path to where the graph should be saved. If set to None (as by default), the graph will not be saved.

Returns Graph object that can be directly displayed in Jupyter notebooks.

Return type graphviz.Digraph

evalml.pipelines.ClassificationPipeline.graph_feature_importance

ClassificationPipeline.**graph_feature_importance**(*importance_threshold=0*)

Generate a bar graph of the pipeline's feature importance

Parameters **importance_threshold** (*float, optional*) – If provided, graph features with a permutation importance whose absolute value is larger than importance_threshold. Defaults to zero.

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

evalml.pipelines.ClassificationPipeline.load

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

Loads pipeline at file path

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

Returns PipelineBase object

evalml.pipelines.ClassificationPipeline.predict

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

Make predictions using selected features.

Parameters

- **X** (*pd.DataFrame or np.array*) – Data of shape [n_samples, n_features]
- **objective** (*Object or string*) – The objective to use to make predictions

Returns Estimated labels

Return type pd.Series

evalml.pipelines.ClassificationPipeline.predict_proba

ClassificationPipeline.**predict_proba**(*X*)

Make probability estimates for labels.

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

Returns Probability estimates

Return type pd.DataFrame

evalml.pipelines.ClassificationPipeline.save

ClassificationPipeline.**save**(*file_path*, *pickle_protocol=4*)

Saves pipeline at file path

Parameters

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

Returns None

evalml.pipelines.ClassificationPipeline.score

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

Evaluate model performance on objectives

Parameters

- **X** (*pd.DataFrame or np.array*) – Data of shape [n_samples, n_features]
- **y** (*pd.Series*) – True labels of length [n_samples]
- **objectives** (*list*) – List of objectives to score

Returns Ordered dictionary of objective scores

Return type dict

evalml.pipelines.BinaryClassificationPipeline

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

Pipeline subclass for all binary classification pipelines.

name = 'Binary Classification Pipeline'

custom_name = None

problem_type = 'binary'

Instance attributes

classes_	Gets the class names for the problem.
feature_importance	Return an attribute of instance, which is of type owner.
parameters	Returns parameter dictionary for this pipeline
threshold	Threshold used to make a prediction.

Methods:

__init__	Machine learning classification pipeline made out of transformers and a classifier.
clone	Constructs a new pipeline with the same parameters and components.
compute_estimator_features	Transforms the data by applying all pre-processing components.
describe	Outputs pipeline details including component parameters
fit	Build a classification model. For string and categorical targets, classes are sorted
get_component	Returns component by name
graph	Generate an image representing the pipeline graph
graph_feature_importance	Generate a bar graph of the pipeline's feature importance
load	Loads pipeline at file path
predict	Make predictions using selected features.
predict_proba	Make probability estimates for labels.
save	Saves pipeline at file path
score	Evaluate model performance on objectives

evalml.pipelines.BinaryClassificationPipeline.__init__

BinaryClassificationPipeline.**__init__**(*parameters*, *random_state=0*)

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

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

Parameters

- **parameters** (*dict*) – Dictionary with component names as keys and dictionary of that component's parameters as values. An empty dictionary {} implies using all default values for component parameters.
- **random_state** (*int, np.random.RandomState*) – The random seed/state. Defaults to 0.

evalml.pipelines.BinaryClassificationPipeline.clone

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

Constructs a new pipeline with the same parameters and components.

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

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

evalml.pipelines.BinaryClassificationPipeline.compute_estimator_features

BinaryClassificationPipeline.**compute_estimator_features**(*X*)

Transforms the data by applying all pre-processing components.

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

Returns pd.DataFrame - New transformed features.

evalml.pipelines.BinaryClassificationPipeline.describe

BinaryClassificationPipeline.**describe**()

Outputs pipeline details including component parameters

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

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

Return type dict

evalml.pipelines.BinaryClassificationPipeline.fit

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

Build a classification model. For string and categorical targets, classes are sorted by sorted(set(y)) and then are mapped to values between 0 and n_classes-1.

Parameters

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

Returns self

evalml.pipelines.BinaryClassificationPipeline.get_component

BinaryClassificationPipeline.**get_component**(*name*)

Returns component by name

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

Returns Component to return

Return type Component

evalml.pipelines.BinaryClassificationPipeline.graph

BinaryClassificationPipeline.**graph**(*filepath=None*)

Generate an image representing the pipeline graph

Parameters **filepath** (*str, optional*) – Path to where the graph should be saved. If set to None (as by default), the graph will not be saved.

Returns Graph object that can be directly displayed in Jupyter notebooks.

Return type graphviz.Digraph

evalml.pipelines.BinaryClassificationPipeline.graph_feature_importance

BinaryClassificationPipeline.**graph_feature_importance**(*importance_threshold=0*)

Generate a bar graph of the pipeline's feature importance

Parameters **importance_threshold** (*float, optional*) – If provided, graph features with a permutation importance whose absolute value is larger than importance_threshold. Defaults to zero.

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

evalml.pipelines.BinaryClassificationPipeline.load

static BinaryClassificationPipeline.**load**(*file_path*)

Loads pipeline at file path

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

Returns PipelineBase object

evalml.pipelines.BinaryClassificationPipeline.predict

BinaryClassificationPipeline.**predict**(*X*, *objective=None*)

Make predictions using selected features.

Parameters

- **X** (*pd.DataFrame or np.array*) – Data of shape [n_samples, n_features]
- **objective** (*Object or string*) – The objective to use to make predictions

Returns Estimated labels

Return type pd.Series

evalml.pipelines.BinaryClassificationPipeline.predict_proba

BinaryClassificationPipeline.**predict_proba**(*X*)

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

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

Returns probability estimates

Return type pd.DataFrame

evalml.pipelines.BinaryClassificationPipeline.save

BinaryClassificationPipeline.**save**(*file_path*, *pickle_protocol=4*)

Saves pipeline at file path

Parameters

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

Returns None

evalml.pipelines.BinaryClassificationPipeline.score

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

Evaluate model performance on objectives

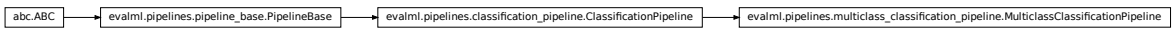
Parameters

- **X** (*pd.DataFrame or np.array*) – Data of shape [n_samples, n_features]
- **y** (*pd.Series*) – True labels of length [n_samples]
- **objectives** (*list*) – List of objectives to score

Returns Ordered dictionary of objective scores

Return type dict

evalml.pipelines.MulticlassClassificationPipeline



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

Pipeline subclass for all multiclass classification pipelines.

name = 'Multiclass Classification Pipeline'

custom_name = None

problem_type = 'multiclass'

Instance attributes

classes_	Gets the class names for the problem.
feature_importance	Return an attribute of instance, which is of type owner.
parameters	Returns parameter dictionary for this pipeline

Methods:

__init__	Machine learning classification pipeline made out of transformers and a classifier.
clone	Constructs a new pipeline with the same parameters and components.
compute_estimator_features	Transforms the data by applying all pre-processing components.
describe	Outputs pipeline details including component parameters
fit	Build a classification model. For string and categorical targets, classes are sorted
get_component	Returns component by name
graph	Generate an image representing the pipeline graph
graph_feature_importance	Generate a bar graph of the pipeline's feature importance
load	Loads pipeline at file path
predict	Make predictions using selected features.
predict_proba	Make probability estimates for labels.
save	Saves pipeline at file path
score	Evaluate model performance on objectives

evalml.pipelines.MulticlassClassificationPipeline.__init__

MulticlassClassificationPipeline.**__init__**(*parameters*, *random_state=0*)

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

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

Parameters

- **parameters** (*dict*) – Dictionary with component names as keys and dictionary of that component's parameters as values. An empty dictionary { } implies using all default values for component parameters.
- **random_state** (*int, np.random.RandomState*) – The random seed/state. Defaults to 0.

evalml.pipelines.MulticlassClassificationPipeline.clone

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

Constructs a new pipeline with the same parameters and components.

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

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

evalml.pipelines.MulticlassClassificationPipeline.compute_estimator_features

MulticlassClassificationPipeline.**compute_estimator_features**(*X*)

Transforms the data by applying all pre-processing components.

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

Returns pd.DataFrame - New transformed features.

evalml.pipelines.MulticlassClassificationPipeline.describe

MulticlassClassificationPipeline.**describe**()

Outputs pipeline details including component parameters

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

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

Return type dict

evalml.pipelines.MulticlassClassificationPipeline.fit

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

Build a classification model. For string and categorical targets, classes are sorted by sorted(set(y)) and then are mapped to values between 0 and n_classes-1.

Parameters

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

Returns self

evalml.pipelines.MulticlassClassificationPipeline.get_component

MulticlassClassificationPipeline.**get_component**(*name*)

Returns component by name

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

Returns Component to return

Return type Component

evalml.pipelines.MulticlassClassificationPipeline.graph

MulticlassClassificationPipeline.**graph**(*filepath=None*)

Generate an image representing the pipeline graph

Parameters **filepath** (*str, optional*) – Path to where the graph should be saved. If set to None (as by default), the graph will not be saved.

Returns Graph object that can be directly displayed in Jupyter notebooks.

Return type graphviz.Digraph

evalml.pipelines.MulticlassClassificationPipeline.graph_feature_importance

MulticlassClassificationPipeline.**graph_feature_importance**(*importance_threshold=0*)
Generate a bar graph of the pipeline's feature importance

Parameters **importance_threshold** (*float, optional*) – If provided, graph features with a permutation importance whose absolute value is larger than importance_threshold. Defaults to zero.

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

evalml.pipelines.MulticlassClassificationPipeline.load

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

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

Returns PipelineBase object

evalml.pipelines.MulticlassClassificationPipeline.predict

MulticlassClassificationPipeline.**predict**(*X*, *objective=None*)
Make predictions using selected features.

Parameters

- **X** (*pd.DataFrame or np.array*) – Data of shape [n_samples, n_features]
- **objective** (*Object or string*) – The objective to use to make predictions

Returns Estimated labels

Return type pd.Series

evalml.pipelines.MulticlassClassificationPipeline.predict_proba

MulticlassClassificationPipeline.**predict_proba**(*X*)
Make probability estimates for labels.

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

Returns Probability estimates

Return type pd.DataFrame

evalml.pipelines.MulticlassClassificationPipeline.save

MulticlassClassificationPipeline.**save**(*file_path*, *pickle_protocol=4*)
Saves pipeline at file path

Parameters

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

Returns None

evalml.pipelines.MulticlassClassificationPipeline.score

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

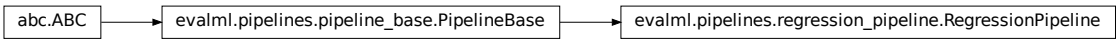
Parameters

- **X** (*pd.DataFrame or np.array*) – Data of shape [n_samples, n_features]
- **y** (*pd.Series*) – True labels of length [n_samples]
- **objectives** (*list*) – List of objectives to score

Returns Ordered dictionary of objective scores

Return type dict

evalml.pipelines.RegressionPipeline



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

name = 'Regression Pipeline'

custom_name = None

problem_type = 'regression'

Instance attributes

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

Methods:

__init__	Machine learning pipeline made out of transformers and a estimator.
clone	Constructs a new pipeline with the same parameters and components.
compute_estimator_features	Transforms the data by applying all pre-processing components.
describe	Outputs pipeline details including component parameters
fit	Build a regression model.
get_component	Returns component by name

Continued on next page

Table 22 – continued from previous page

graph	Generate an image representing the pipeline graph
graph_feature_importance	Generate a bar graph of the pipeline's feature importance
load	Loads pipeline at file path
predict	Make predictions using selected features.
save	Saves pipeline at file path
score	Evaluate model performance on current and additional objectives

evalml.pipelines.RegressionPipeline.__init__

RegressionPipeline.**__init__**(*parameters*, *random_state=0*)

Machine learning pipeline made out of transformers and a estimator.

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

Parameters

- **parameters** (*dict*) – Dictionary with component names as keys and dictionary of that component's parameters as values. An empty dictionary { } implies using all default values for component parameters.
- **random_state** (*int, np.random.RandomState*) – The random seed/state. Defaults to 0.

evalml.pipelines.RegressionPipeline.clone

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

Constructs a new pipeline with the same parameters and components.

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

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

evalml.pipelines.RegressionPipeline.compute_estimator_features

RegressionPipeline.**compute_estimator_features**(*X*)

Transforms the data by applying all pre-processing components.

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

Returns pd.DataFrame - New transformed features.

evalml.pipelines.RegressionPipeline.describe

RegressionPipeline.**describe**()

Outputs pipeline details including component parameters

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

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

Return type dict

evalml.pipelines.RegressionPipeline.fit

RegressionPipeline.**fit**(*X*, *y*)
Build a regression model.

Parameters

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

Returns self

evalml.pipelines.RegressionPipeline.get_component

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

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

Returns Component to return

Return type Component

evalml.pipelines.RegressionPipeline.graph

RegressionPipeline.**graph**(*filepath=None*)
Generate an image representing the pipeline graph

Parameters **filepath** (*str, optional*) – Path to where the graph should be saved. If set to None (as by default), the graph will not be saved.

Returns Graph object that can be directly displayed in Jupyter notebooks.

Return type graphviz.Digraph

evalml.pipelines.RegressionPipeline.graph_feature_importance

RegressionPipeline.**graph_feature_importance**(*importance_threshold=0*)
Generate a bar graph of the pipeline's feature importance

Parameters **importance_threshold** (*float, optional*) – If provided, graph features with a permutation importance whose absolute value is larger than importance_threshold. Defaults to zero.

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

evalml.pipelines.RegressionPipeline.load

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

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

Returns PipelineBase object

evalml.pipelines.RegressionPipeline.predict

RegressionPipeline.**predict**(*X*, *objective=None*)

Make predictions using selected features.

Parameters

- **X** (*pd.DataFrame or np.array*) – Data of shape [n_samples, n_features]
- **objective** (*Object or string*) – The objective to use to make predictions

Returns Predicted values.

Return type pd.Series

evalml.pipelines.RegressionPipeline.save

RegressionPipeline.**save**(*file_path*, *pickle_protocol=4*)

Saves pipeline at file path

Parameters

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

Returns None

evalml.pipelines.RegressionPipeline.score

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

Evaluate model performance on current and additional objectives

Parameters

- **X** (*pd.DataFrame or np.array*) – Data of shape [n_samples, n_features]
- **y** (*pd.Series*) – True values of length [n_samples]
- **objectives** (*list*) – Non-empty list of objectives to score on

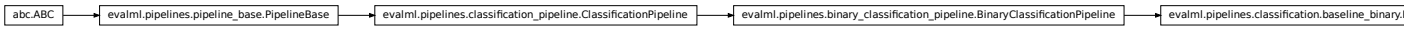
Returns Ordered dictionary of objective scores

Return type dict

5.4.2 Classification Pipelines

BaselineBinaryPipeline	Baseline Pipeline for binary classification.
BaselineMulticlassPipeline	Baseline Pipeline for multiclass classification.
ModeBaselineBinaryPipeline	Mode Baseline Pipeline for binary classification.
ModeBaselineMulticlassPipeline	Mode Baseline Pipeline for multiclass classification.

evalml.pipelines.BaselineBinaryPipeline



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

Baseline Pipeline for binary classification.

name = 'Baseline Classification Pipeline'

custom_name = 'Baseline Classification Pipeline'

summary = 'Baseline Classifier'

component_graph = ['Baseline Classifier']

problem_type = 'binary'

model_family = 'baseline'

hyperparameters = {'Baseline Classifier': {}}

custom_hyperparameters = None

default_parameters = {'Baseline Classifier': {'strategy': 'mode'}}

Instance attributes

classes_	Gets the class names for the problem.
feature_importance	Return an attribute of instance, which is of type owner.
parameters	Returns parameter dictionary for this pipeline
threshold	Threshold used to make a prediction.

Methods:

__init__	Machine learning classification pipeline made out of transformers and a classifier.
clone	Constructs a new pipeline with the same parameters and components.
compute_estimator_features	Transforms the data by applying all pre-processing components.
describe	Outputs pipeline details including component parameters
fit	Build a classification model. For string and categorical targets, classes are sorted
get_component	Returns component by name
graph	Generate an image representing the pipeline graph
graph_feature_importance	Generate a bar graph of the pipeline's feature importance
load	Loads pipeline at file path
predict	Make predictions using selected features.

Continued on next page

Table 25 – continued from previous page

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

evalml.pipelines.BaselineBinaryPipeline.__init__

BaselineBinaryPipeline.**__init__**(*parameters*, *random_state=0*)

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

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

Parameters

- **parameters** (*dict*) – Dictionary with component names as keys and dictionary of that component's parameters as values. An empty dictionary { } implies using all default values for component parameters.
- **random_state** (*int, np.random.RandomState*) – The random seed/state. Defaults to 0.

evalml.pipelines.BaselineBinaryPipeline.clone

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

Constructs a new pipeline with the same parameters and components.

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

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

evalml.pipelines.BaselineBinaryPipeline.compute_estimator_features

BaselineBinaryPipeline.**compute_estimator_features**(*X*)

Transforms the data by applying all pre-processing components.

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

Returns pd.DataFrame - New transformed features.

evalml.pipelines.BaselineBinaryPipeline.describe

BaselineBinaryPipeline.**describe**()

Outputs pipeline details including component parameters

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

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

Return type dict

evalml.pipelines.BaselineBinaryPipeline.fit

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

Build a classification model. For string and categorical targets, classes are sorted by sorted(set(y)) and then are mapped to values between 0 and n_classes-1.

Parameters

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

Returns self

evalml.pipelines.BaselineBinaryPipeline.get_component

BaselineBinaryPipeline.**get_component**(*name*)

Returns component by name

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

Returns Component to return

Return type Component

evalml.pipelines.BaselineBinaryPipeline.graph

BaselineBinaryPipeline.**graph**(*filepath=None*)

Generate an image representing the pipeline graph

Parameters **filepath** (*str, optional*) – Path to where the graph should be saved. If set to None (as by default), the graph will not be saved.

Returns Graph object that can be directly displayed in Jupyter notebooks.

Return type graphviz.Digraph

evalml.pipelines.BaselineBinaryPipeline.graph_feature_importance

BaselineBinaryPipeline.**graph_feature_importance**(*importance_threshold=0*)

Generate a bar graph of the pipeline's feature importance

Parameters **importance_threshold** (*float, optional*) – If provided, graph features with a permutation importance whose absolute value is larger than importance_threshold. Defaults to zero.

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

evalml.pipelines.BaselineBinaryPipeline.load

static BaselineBinaryPipeline.**load**(*file_path*)

Loads pipeline at file path

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

Returns PipelineBase object

evalml.pipelines.BaselineBinaryPipeline.predict

BaselineBinaryPipeline.**predict**(*X*, *objective=None*)

Make predictions using selected features.

Parameters

- **X** (*pd.DataFrame or np.array*) – Data of shape [n_samples, n_features]
- **objective** (*Object or string*) – The objective to use to make predictions

Returns Estimated labels

Return type pd.Series

evalml.pipelines.BaselineBinaryPipeline.predict_proba

BaselineBinaryPipeline.**predict_proba**(*X*)

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

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

Returns probability estimates

Return type pd.DataFrame

evalml.pipelines.BaselineBinaryPipeline.save

BaselineBinaryPipeline.**save**(*file_path*, *pickle_protocol=4*)

Saves pipeline at file path

Parameters

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

Returns None

evalml.pipelines.BaselineBinaryPipeline.score

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

Evaluate model performance on objectives

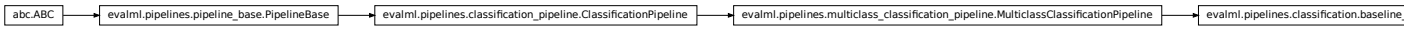
Parameters

- **X** (*pd.DataFrame or np.array*) – Data of shape [n_samples, n_features]
- **y** (*pd.Series*) – True labels of length [n_samples]
- **objectives** (*list*) – List of objectives to score

Returns Ordered dictionary of objective scores

Return type dict

evalml.pipelines.BaselineMulticlassPipeline



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

Baseline Pipeline for multiclass classification.

name = 'Baseline Multiclass Classification Pipeline'

custom_name = 'Baseline Multiclass Classification Pipeline'

summary = 'Baseline Classifier'

component_graph = ['Baseline Classifier']

problem_type = 'multiclass'

model_family = 'baseline'

hyperparameters = {'Baseline Classifier': {}}

custom_hyperparameters = None

default_parameters = {'Baseline Classifier': {'strategy': 'mode'}}

Instance attributes

classes_	Gets the class names for the problem.
feature_importance	Return an attribute of instance, which is of type owner.
parameters	Returns parameter dictionary for this pipeline

Methods:

__init__	Machine learning classification pipeline made out of transformers and a classifier.
clone	Constructs a new pipeline with the same parameters and components.
compute_estimator_features	Transforms the data by applying all pre-processing components.
describe	Outputs pipeline details including component parameters
fit	Build a classification model. For string and categorical targets, classes are sorted
get_component	Returns component by name
graph	Generate an image representing the pipeline graph
graph_feature_importance	Generate a bar graph of the pipeline's feature importance
load	Loads pipeline at file path
predict	Make predictions using selected features.
predict_proba	Make probability estimates for labels.

Continued on next page

Table 27 – continued from previous page

save	Saves pipeline at file path
score	Evaluate model performance on objectives

evalml.pipelines.BaselineMulticlassPipeline.__init__

BaselineMulticlassPipeline.**__init__**(*parameters*, *random_state=0*)

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

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

Parameters

- **parameters** (*dict*) – Dictionary with component names as keys and dictionary of that component's parameters as values. An empty dictionary { } implies using all default values for component parameters.
- **random_state** (*int, np.random.RandomState*) – The random seed/state. Defaults to 0.

evalml.pipelines.BaselineMulticlassPipeline.clone

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

Constructs a new pipeline with the same parameters and components.

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

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

evalml.pipelines.BaselineMulticlassPipeline.compute_estimator_features

BaselineMulticlassPipeline.**compute_estimator_features**(*X*)

Transforms the data by applying all pre-processing components.

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

Returns pd.DataFrame - New transformed features.

evalml.pipelines.BaselineMulticlassPipeline.describe

BaselineMulticlassPipeline.**describe**()

Outputs pipeline details including component parameters

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

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

Return type dict

evalml.pipelines.BaselineMulticlassPipeline.fit

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

Build a classification model. For string and categorical targets, classes are sorted by sorted(set(y)) and then are mapped to values between 0 and n_classes-1.

Parameters

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

Returns self

evalml.pipelines.BaselineMulticlassPipeline.get_component

BaselineMulticlassPipeline.**get_component**(*name*)

Returns component by name

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

Returns Component to return

Return type Component

evalml.pipelines.BaselineMulticlassPipeline.graph

BaselineMulticlassPipeline.**graph**(*filepath=None*)

Generate an image representing the pipeline graph

Parameters **filepath** (*str, optional*) – Path to where the graph should be saved. If set to None (as by default), the graph will not be saved.

Returns Graph object that can be directly displayed in Jupyter notebooks.

Return type graphviz.Digraph

evalml.pipelines.BaselineMulticlassPipeline.graph_feature_importance

BaselineMulticlassPipeline.**graph_feature_importance**(*importance_threshold=0*)

Generate a bar graph of the pipeline's feature importance

Parameters **importance_threshold** (*float, optional*) – If provided, graph features with a permutation importance whose absolute value is larger than importance_threshold. Defaults to zero.

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

evalml.pipelines.BaselineMulticlassPipeline.load

static BaselineMulticlassPipeline.**load**(*file_path*)

Loads pipeline at file path

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

Returns PipelineBase object

evalml.pipelines.BaselineMulticlassPipeline.predict

BaselineMulticlassPipeline.**predict**(*X*, *objective=None*)

Make predictions using selected features.

Parameters

- **X** (*pd.DataFrame or np.array*) – Data of shape [n_samples, n_features]
- **objective** (*Object or string*) – The objective to use to make predictions

Returns Estimated labels

Return type pd.Series

evalml.pipelines.BaselineMulticlassPipeline.predict_proba

BaselineMulticlassPipeline.**predict_proba**(*X*)

Make probability estimates for labels.

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

Returns Probability estimates

Return type pd.DataFrame

evalml.pipelines.BaselineMulticlassPipeline.save

BaselineMulticlassPipeline.**save**(*file_path*, *pickle_protocol=4*)

Saves pipeline at file path

Parameters

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

Returns None

evalml.pipelines.BaselineMulticlassPipeline.score

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

Evaluate model performance on objectives

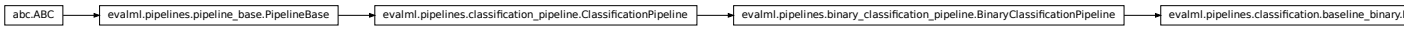
Parameters

- **X** (*pd.DataFrame or np.array*) – Data of shape [n_samples, n_features]
- **y** (*pd.Series*) – True labels of length [n_samples]
- **objectives** (*list*) – List of objectives to score

Returns Ordered dictionary of objective scores

Return type dict

evalml.pipelines.ModeBaselineBinaryPipeline



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

Mode Baseline Pipeline for binary classification.

name = 'Mode Baseline Binary Classification Pipeline'

custom_name = 'Mode Baseline Binary Classification Pipeline'

summary = 'Baseline Classifier'

component_graph = ['Baseline Classifier']

problem_type = 'binary'

model_family = 'baseline'

hyperparameters = {'Baseline Classifier': {}}

custom_hyperparameters = {'strategy': ['mode']}

default_parameters = {'Baseline Classifier': {'strategy': 'mode'}}

Instance attributes

classes_	Gets the class names for the problem.
feature_importance	Return an attribute of instance, which is of type owner.
parameters	Returns parameter dictionary for this pipeline
threshold	Threshold used to make a prediction.

Methods:

__init__	Machine learning classification pipeline made out of transformers and a classifier.
clone	Constructs a new pipeline with the same parameters and components.
compute_estimator_features	Transforms the data by applying all pre-processing components.
describe	Outputs pipeline details including component parameters
fit	Build a classification model. For string and categorical targets, classes are sorted
get_component	Returns component by name
graph	Generate an image representing the pipeline graph
graph_feature_importance	Generate a bar graph of the pipeline's feature importance
load	Loads pipeline at file path
predict	Make predictions using selected features.

Continued on next page

Table 29 – continued from previous page

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

evalml.pipelines.ModeBaselineBinaryPipeline.__init__

ModeBaselineBinaryPipeline.**__init__**(*parameters*, *random_state=0*)

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

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

Parameters

- **parameters** (*dict*) – Dictionary with component names as keys and dictionary of that component's parameters as values. An empty dictionary { } implies using all default values for component parameters.
- **random_state** (*int, np.random.RandomState*) – The random seed/state. Defaults to 0.

evalml.pipelines.ModeBaselineBinaryPipeline.clone

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

Constructs a new pipeline with the same parameters and components.

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

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

evalml.pipelines.ModeBaselineBinaryPipeline.compute_estimator_features

ModeBaselineBinaryPipeline.**compute_estimator_features**(*X*)

Transforms the data by applying all pre-processing components.

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

Returns pd.DataFrame - New transformed features.

evalml.pipelines.ModeBaselineBinaryPipeline.describe

ModeBaselineBinaryPipeline.**describe**()

Outputs pipeline details including component parameters

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

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

Return type dict

evalml.pipelines.ModeBaselineBinaryPipeline.fit

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

Build a classification model. For string and categorical targets, classes are sorted by sorted(set(y)) and then are mapped to values between 0 and n_classes-1.

Parameters

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

Returns self

evalml.pipelines.ModeBaselineBinaryPipeline.get_component

ModeBaselineBinaryPipeline.**get_component**(*name*)

Returns component by name

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

Returns Component to return

Return type Component

evalml.pipelines.ModeBaselineBinaryPipeline.graph

ModeBaselineBinaryPipeline.**graph**(*filepath=None*)

Generate an image representing the pipeline graph

Parameters **filepath** (*str, optional*) – Path to where the graph should be saved. If set to None (as by default), the graph will not be saved.

Returns Graph object that can be directly displayed in Jupyter notebooks.

Return type graphviz.Digraph

evalml.pipelines.ModeBaselineBinaryPipeline.graph_feature_importance

ModeBaselineBinaryPipeline.**graph_feature_importance**(*importance_threshold=0*)

Generate a bar graph of the pipeline's feature importance

Parameters **importance_threshold** (*float, optional*) – If provided, graph features with a permutation importance whose absolute value is larger than importance_threshold. Defaults to zero.

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

evalml.pipelines.ModeBaselineBinaryPipeline.load

static ModeBaselineBinaryPipeline.**load**(*file_path*)

Loads pipeline at file path

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

Returns PipelineBase object

evalml.pipelines.ModeBaselineBinaryPipeline.predict

ModeBaselineBinaryPipeline.**predict**(*X*, *objective=None*)

Make predictions using selected features.

Parameters

- **X** (*pd.DataFrame or np.array*) – Data of shape [n_samples, n_features]
- **objective** (*Object or string*) – The objective to use to make predictions

Returns Estimated labels

Return type pd.Series

evalml.pipelines.ModeBaselineBinaryPipeline.predict_proba

ModeBaselineBinaryPipeline.**predict_proba**(*X*)

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

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

Returns probability estimates

Return type pd.DataFrame

evalml.pipelines.ModeBaselineBinaryPipeline.save

ModeBaselineBinaryPipeline.**save**(*file_path*, *pickle_protocol=4*)

Saves pipeline at file path

Parameters

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

Returns None

evalml.pipelines.ModeBaselineBinaryPipeline.score

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

Evaluate model performance on objectives

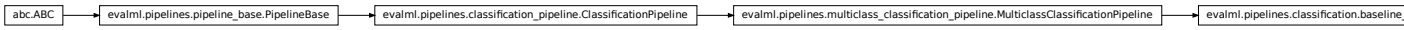
Parameters

- **X** (*pd.DataFrame or np.array*) – Data of shape [n_samples, n_features]
- **y** (*pd.Series*) – True labels of length [n_samples]
- **objectives** (*list*) – List of objectives to score

Returns Ordered dictionary of objective scores

Return type dict

evalml.pipelines.ModeBaselineMulticlassPipeline



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

Mode Baseline Pipeline for multiclass classification.

name = 'Mode Baseline Multiclass Classification Pipeline'

custom_name = 'Mode Baseline Multiclass Classification Pipeline'

summary = 'Baseline Classifier'

component_graph = ['Baseline Classifier']

problem_type = 'multiclass'

model_family = 'baseline'

hyperparameters = {'Baseline Classifier': {}}

custom_hyperparameters = {'strategy': ['mode']}

default_parameters = {'Baseline Classifier': {'strategy': 'mode'}}

Instance attributes

classes_	Gets the class names for the problem.
feature_importance	Return an attribute of instance, which is of type owner.
parameters	Returns parameter dictionary for this pipeline

Methods:

__init__	Machine learning classification pipeline made out of transformers and a classifier.
clone	Constructs a new pipeline with the same parameters and components.
compute_estimator_features	Transforms the data by applying all pre-processing components.
describe	Outputs pipeline details including component parameters
fit	Build a classification model. For string and categorical targets, classes are sorted
get_component	Returns component by name
graph	Generate an image representing the pipeline graph
graph_feature_importance	Generate a bar graph of the pipeline's feature importance
load	Loads pipeline at file path
predict	Make predictions using selected features.
predict_proba	Make probability estimates for labels.

Continued on next page

Table 31 – continued from previous page

save	Saves pipeline at file path
score	Evaluate model performance on objectives

evalml.pipelines.ModeBaselineMulticlassPipeline.__init__

ModeBaselineMulticlassPipeline.**__init__**(*parameters*, *random_state=0*)

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

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

Parameters

- **parameters** (*dict*) – Dictionary with component names as keys and dictionary of that component's parameters as values. An empty dictionary { } implies using all default values for component parameters.
- **random_state** (*int, np.random.RandomState*) – The random seed/state. Defaults to 0.

evalml.pipelines.ModeBaselineMulticlassPipeline.clone

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

Constructs a new pipeline with the same parameters and components.

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

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

evalml.pipelines.ModeBaselineMulticlassPipeline.compute_estimator_features

ModeBaselineMulticlassPipeline.**compute_estimator_features**(*X*)

Transforms the data by applying all pre-processing components.

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

Returns pd.DataFrame - New transformed features.

evalml.pipelines.ModeBaselineMulticlassPipeline.describe

ModeBaselineMulticlassPipeline.**describe**()

Outputs pipeline details including component parameters

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

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

Return type dict

evalml.pipelines.ModeBaselineMulticlassPipeline.fit

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

Build a classification model. For string and categorical targets, classes are sorted by sorted(set(y)) and then are mapped to values between 0 and n_classes-1.

Parameters

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

Returns self

evalml.pipelines.ModeBaselineMulticlassPipeline.get_component

ModeBaselineMulticlassPipeline.**get_component**(*name*)

Returns component by name

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

Returns Component to return

Return type Component

evalml.pipelines.ModeBaselineMulticlassPipeline.graph

ModeBaselineMulticlassPipeline.**graph**(*filepath=None*)

Generate an image representing the pipeline graph

Parameters **filepath** (*str, optional*) – Path to where the graph should be saved. If set to None (as by default), the graph will not be saved.

Returns Graph object that can be directly displayed in Jupyter notebooks.

Return type graphviz.Digraph

evalml.pipelines.ModeBaselineMulticlassPipeline.graph_feature_importance

ModeBaselineMulticlassPipeline.**graph_feature_importance**(*importance_threshold=0*)

Generate a bar graph of the pipeline's feature importance

Parameters **importance_threshold** (*float, optional*) – If provided, graph features with a permutation importance whose absolute value is larger than importance_threshold. Defaults to zero.

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

evalml.pipelines.ModeBaselineMulticlassPipeline.load

static ModeBaselineMulticlassPipeline.**load**(*file_path*)

Loads pipeline at file path

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

Returns PipelineBase object

evalml.pipelines.ModeBaselineMulticlassPipeline.predict

ModeBaselineMulticlassPipeline.**predict**(*X*, *objective=None*)

Make predictions using selected features.

Parameters

- **X** (*pd.DataFrame or np.array*) – Data of shape [n_samples, n_features]
- **objective** (*Object or string*) – The objective to use to make predictions

Returns Estimated labels

Return type pd.Series

evalml.pipelines.ModeBaselineMulticlassPipeline.predict_proba

ModeBaselineMulticlassPipeline.**predict_proba**(*X*)

Make probability estimates for labels.

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

Returns Probability estimates

Return type pd.DataFrame

evalml.pipelines.ModeBaselineMulticlassPipeline.save

ModeBaselineMulticlassPipeline.**save**(*file_path*, *pickle_protocol=4*)

Saves pipeline at file path

Parameters

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

Returns None

evalml.pipelines.ModeBaselineMulticlassPipeline.score

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

Evaluate model performance on objectives

Parameters

- **X** (*pd.DataFrame or np.array*) – Data of shape [n_samples, n_features]
- **y** (*pd.Series*) – True labels of length [n_samples]
- **objectives** (*list*) – List of objectives to score

Returns Ordered dictionary of objective scores

Return type dict

5.4.3 Regression Pipelines

BaselineRegressionPipeline	Baseline Pipeline for regression problems.
MeanBaselineRegressionPipeline	Baseline Pipeline for regression problems.

evalml.pipelines.BaselineRegressionPipeline

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

Baseline Pipeline for regression problems.

name = 'Baseline Regression Pipeline'

custom_name = None

summary = 'Baseline Regressor'

component_graph = ['Baseline Regressor']

problem_type = 'regression'

model_family = 'baseline'

hyperparameters = {'Baseline Regressor': {}}

custom_hyperparameters = None

default_parameters = {'Baseline Regressor': {'strategy': 'mean'}}

Instance attributes

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

Methods:

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

Continued on next page

Table 34 – continued from previous page

graph_feature_importance	Generate a bar graph of the pipeline's feature importance
load	Loads pipeline at file path
predict	Make predictions using selected features.
save	Saves pipeline at file path
score	Evaluate model performance on current and additional objectives

evalml.pipelines.BaselineRegressionPipeline.__init__

BaselineRegressionPipeline.**__init__**(*parameters*, *random_state=0*)

Machine learning pipeline made out of transformers and a estimator.

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

Parameters

- **parameters** (*dict*) – Dictionary with component names as keys and dictionary of that component's parameters as values. An empty dictionary {} implies using all default values for component parameters.
- **random_state** (*int, np.random.RandomState*) – The random seed/state. Defaults to 0.

evalml.pipelines.BaselineRegressionPipeline.clone

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

Constructs a new pipeline with the same parameters and components.

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

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

evalml.pipelines.BaselineRegressionPipeline.compute_estimator_features

BaselineRegressionPipeline.**compute_estimator_features**(*X*)

Transforms the data by applying all pre-processing components.

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

Returns pd.DataFrame - New transformed features.

evalml.pipelines.BaselineRegressionPipeline.describe

BaselineRegressionPipeline.**describe**()

Outputs pipeline details including component parameters

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

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

Return type dict

evalml.pipelines.BaselineRegressionPipeline.fit

BaselineRegressionPipeline.**fit**(*X*, *y*)
Build a regression model.

Parameters

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

Returns self

evalml.pipelines.BaselineRegressionPipeline.get_component

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

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

Returns Component to return

Return type Component

evalml.pipelines.BaselineRegressionPipeline.graph

BaselineRegressionPipeline.**graph**(*filepath=None*)
Generate an image representing the pipeline graph

Parameters **filepath** (*str, optional*) – Path to where the graph should be saved. If set to None (as by default), the graph will not be saved.

Returns Graph object that can be directly displayed in Jupyter notebooks.

Return type graphviz.Digraph

evalml.pipelines.BaselineRegressionPipeline.graph_feature_importance

BaselineRegressionPipeline.**graph_feature_importance**(*importance_threshold=0*)
Generate a bar graph of the pipeline's feature importance

Parameters **importance_threshold** (*float, optional*) – If provided, graph features with a permutation importance whose absolute value is larger than importance_threshold. Defaults to zero.

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

evalml.pipelines.BaselineRegressionPipeline.load

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

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

Returns PipelineBase object

evalml.pipelines.BaselineRegressionPipeline.predict

BaselineRegressionPipeline.**predict**(*X*, *objective=None*)

Make predictions using selected features.

Parameters

- **X** (*pd.DataFrame or np.array*) – Data of shape [n_samples, n_features]
- **objective** (*Object or string*) – The objective to use to make predictions

Returns Predicted values.

Return type pd.Series

evalml.pipelines.BaselineRegressionPipeline.save

BaselineRegressionPipeline.**save**(*file_path*, *pickle_protocol=4*)

Saves pipeline at file path

Parameters

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

Returns None

evalml.pipelines.BaselineRegressionPipeline.score

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

Evaluate model performance on current and additional objectives

Parameters

- **X** (*pd.DataFrame or np.array*) – Data of shape [n_samples, n_features]
- **y** (*pd.Series*) – True values of length [n_samples]
- **objectives** (*list*) – Non-empty list of objectives to score on

Returns Ordered dictionary of objective scores

Return type dict

evalml.pipelines.MeanBaselineRegressionPipeline

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

Baseline Pipeline for regression problems.

name = 'Mean Baseline Regression Pipeline'

custom_name = None

summary = 'Baseline Regressor'

component_graph = ['Baseline Regressor']

problem_type = 'regression'

model_family = 'baseline'

hyperparameters = {'Baseline Regressor': {}}

custom_hyperparameters = {'strategy': ['mean']}

default_parameters = {'Baseline Regressor': {'strategy': 'mean'}}

Instance attributes

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

Methods:

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

evalml.pipelines.MeanBaselineRegressionPipeline.__init__

MeanBaselineRegressionPipeline.**__init__**(*parameters*, *random_state=0*)

Machine learning pipeline made out of transformers and a estimator.

Required Class Variables: component_graph (list): List of components in order. Accepts strings or

ComponentBase subclasses in the list

Parameters

- **parameters** (*dict*) – Dictionary with component names as keys and dictionary of that component's parameters as values. An empty dictionary { } implies using all default values for component parameters.
- **random_state** (*int, np.random.RandomState*) – The random seed/state. Defaults to 0.

evalml.pipelines.MeanBaselineRegressionPipeline.clone

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

Constructs a new pipeline with the same parameters and components.

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

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

evalml.pipelines.MeanBaselineRegressionPipeline.compute_estimator_features

MeanBaselineRegressionPipeline.**compute_estimator_features**(*X*)

Transforms the data by applying all pre-processing components.

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

Returns pd.DataFrame - New transformed features.

evalml.pipelines.MeanBaselineRegressionPipeline.describe

MeanBaselineRegressionPipeline.**describe**()

Outputs pipeline details including component parameters

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

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

Return type dict

evalml.pipelines.MeanBaselineRegressionPipeline.fit

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

Build a regression model.

Parameters

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

Returns self

evalml.pipelines.MeanBaselineRegressionPipeline.get_component

MeanBaselineRegressionPipeline.**get_component**(*name*)

Returns component by name

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

Returns Component to return

Return type Component

evalml.pipelines.MeanBaselineRegressionPipeline.graph

MeanBaselineRegressionPipeline.**graph**(*filepath=None*)

Generate an image representing the pipeline graph

Parameters **filepath** (*str, optional*) – Path to where the graph should be saved. If set to None (as by default), the graph will not be saved.

Returns Graph object that can be directly displayed in Jupyter notebooks.

Return type graphviz.Digraph

evalml.pipelines.MeanBaselineRegressionPipeline.graph_feature_importance

MeanBaselineRegressionPipeline.**graph_feature_importance**(*importance_threshold=0*)

Generate a bar graph of the pipeline's feature importance

Parameters **importance_threshold** (*float, optional*) – If provided, graph features with a permutation importance whose absolute value is larger than importance_threshold. Defaults to zero.

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

evalml.pipelines.MeanBaselineRegressionPipeline.load

static MeanBaselineRegressionPipeline.**load**(*file_path*)

Loads pipeline at file path

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

Returns PipelineBase object

evalml.pipelines.MeanBaselineRegressionPipeline.predict

MeanBaselineRegressionPipeline.**predict**(*X*, *objective=None*)

Make predictions using selected features.

Parameters

- **X** (*pd.DataFrame or np.array*) – Data of shape [n_samples, n_features]
- **objective** (*Object or string*) – The objective to use to make predictions

Returns Predicted values.

Return type pd.Series

evalml.pipelines.MeanBaselineRegressionPipeline.save

MeanBaselineRegressionPipeline.**save**(*file_path*, *pickle_protocol=4*)

Saves pipeline at file path

Parameters

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

Returns None

evalml.pipelines.MeanBaselineRegressionPipeline.score

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

Evaluate model performance on current and additional objectives

Parameters

- **X** (*pd.DataFrame or np.array*) – Data of shape [n_samples, n_features]
- **y** (*pd.Series*) – True values of length [n_samples]
- **objectives** (*list*) – Non-empty list of objectives to score on

Returns Ordered dictionary of objective scores

Return type dict

5.4.4 Pipeline Utils

make_pipeline	Given input data, target data, an estimator class and the problem type,
make_pipeline_from_components	Given a list of component instances and the problem type, an pipeline instance is generated with the component instances.

evalml.pipelines.utils.make_pipeline

evalml.pipelines.utils.**make_pipeline**(*X*, *y*, *estimator*, *problem_type*)

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

Parameters

- **X** (*pd.DataFrame*) – The input data of shape [n_samples, n_features]
- **y** (*pd.Series*) – The target data of length [n_samples]
- **estimator** (*Estimator*) – Estimator for pipeline
- **problem_type** – Problem type for pipeline to generate

evalml.pipelines.utils.make_pipeline_from_components

evalml.pipelines.utils.**make_pipeline_from_components**(*component_instances*, *problem_type*, *custom_name=None*)

Given a list of component instances and the problem type, an pipeline instance is generated with the component instances.

The pipeline will be a subclass of the appropriate pipeline base class for the specified problem_type. The pipeline will be untrained, even if the input components are already trained. A custom name for the pipeline can optionally be specified; otherwise the default pipeline name will be 'Templated Pipeline'.

Parameters

- **component_instances** (*list*) – a list of all of the components to include in the pipeline
- **problem_type** (*str or ProblemTypes*) – problem type for the pipeline to generate
- **custom_name** – a name for the new pipeline

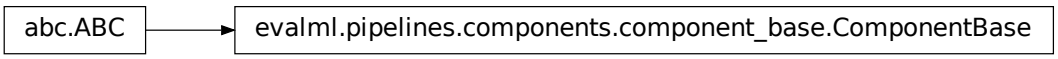
5.5 Components

5.5.1 Component Base Classes

Components represent a step in a pipeline.

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

evalml.pipelines.components.ComponentBase



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

Base class for all components.

Methods

__init__	Initialize self.

Continued on next page

Table 39 – continued from previous page

clone	Constructs a new component with the same parameters
describe	Describe a component and its parameters
fit	Fits component to data
load	Loads component at file path
save	Saves component at file path

evalml.pipelines.components.ComponentBase.__init__

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

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

evalml.pipelines.components.ComponentBase.clone

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

Constructs a new component with the same parameters

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

Returns A new instance of this component with identical parameters

evalml.pipelines.components.ComponentBase.describe

ComponentBase.**describe**(*print_name=False*, *return_dict=False*)

Describe a component and its parameters

Parameters

- **print_name** (*bool, optional*) – whether to print name of component
- **return_dict** (*bool, optional*) – whether to return description as dictionary in the format {"name": name, "parameters": parameters}

Returns prints and returns dictionary

Return type None or dict

evalml.pipelines.components.ComponentBase.fit

ComponentBase.**fit**(*X*, *y=None*)

Fits component to data

Parameters

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

Returns self

evalml.pipelines.components.ComponentBase.load

static ComponentBase.**load**(*file_path*)
Loads component at file path

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

Returns ComponentBase object

evalml.pipelines.components.ComponentBase.save

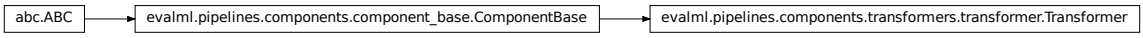
ComponentBase.**save**(*file_path*, *pickle_protocol=4*)
Saves component at file path

Parameters

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

Returns None

evalml.pipelines.components.Transformer



class evalml.pipelines.components.**Transformer**(*parameters=None*, *component_obj=None*, *random_state=0*, ***kwargs*)
A component that may or may not need fitting that transforms data. These components are used before an estimator.

To implement a new Transformer, define your own class which is a subclass of Transformer, including a name and a list of acceptable ranges for any parameters to be tuned during the automl search (hyperparameters). Define an *__init__* method which sets up any necessary state and objects. Make sure your *__init__* only uses standard keyword arguments and calls *super().__init__()* with a parameters dict. You may also override the *fit*, *transform*, *fit_transform* and other methods in this class if appropriate.

To see some examples, check out the definitions of any Transformer component.

Methods

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

Continued on next page

Table 40 – continued from previous page

save	Saves component at file path
transform	Transforms data X

evalml.pipelines.components.Transformer.__init__

Transformer.**__init__**(*parameters=None*, *component_obj=None*, *random_state=0*, ***kwargs*)
Initialize self. See help(type(self)) for accurate signature.

evalml.pipelines.components.Transformer.clone

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

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

Returns A new instance of this component with identical parameters

evalml.pipelines.components.Transformer.describe

Transformer.**describe**(*print_name=False*, *return_dict=False*)
Describe a component and its parameters

Parameters

- **print_name** (*bool, optional*) – whether to print name of component
- **return_dict** (*bool, optional*) – whether to return description as dictionary in the format {"name": name, "parameters": parameters}

Returns prints and returns dictionary

Return type None or dict

evalml.pipelines.components.Transformer.fit

Transformer.**fit**(*X*, *y=None*)
Fits component to data

Parameters

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

Returns self

evalml.pipelines.components.Transformer.fit_transform

Transformer.**fit_transform**(*X*, *y=None*)
Fits on X and transforms X

Parameters

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

Returns Transformed X

Return type pd.DataFrame

evalml.pipelines.components.Transformer.load

static Transformer.**load**(*file_path*)

Loads component at file path

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

Returns ComponentBase object

evalml.pipelines.components.Transformer.save

Transformer.**save**(*file_path*, *pickle_protocol=4*)

Saves component at file path

Parameters

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

Returns None

evalml.pipelines.components.Transformer.transform

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

Transforms data X

Parameters

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

Returns Transformed X

Return type pd.DataFrame

evalml.pipelines.components.Estimator

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

A component that fits and predicts given data.

To implement a new Transformer, define your own class which is a subclass of Transformer, including a name and a list of acceptable ranges for any parameters to be tuned during the automl search (hyperparameters). Define an *__init__* method which sets up any necessary state and objects. Make sure your *__init__* only uses standard keyword arguments and calls *super().__init__()* with a parameters dict. You may also override the *fit*, *transform*, *fit_transform* and other methods in this class if appropriate.

To see some examples, check out the definitions of any Estimator component.

Methods

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

evalml.pipelines.components.Estimator.__init__

Estimator.**__init__**(*parameters=None*, *component_obj=None*, *random_state=0*, ***kwargs*)
Initialize self. See help(type(self)) for accurate signature.

evalml.pipelines.components.Estimator.clone

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

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

Returns A new instance of this component with identical parameters

evalml.pipelines.components.Estimator.describe

Estimator.**describe**(*print_name=False*, *return_dict=False*)
Describe a component and its parameters

Parameters

- **print_name** (*bool, optional*) – whether to print name of component
- **return_dict** (*bool, optional*) – whether to return description as dictionary in the format {"name": name, "parameters": parameters}

Returns prints and returns dictionary

Return type None or dict

evalml.pipelines.components.Estimator.fit

Estimator.**fit**(*X*, *y=None*)
Fits component to data

Parameters

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

Returns self

evalml.pipelines.components.Estimator.load

static Estimator.**load**(*file_path*)
Loads component at file path

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

Returns ComponentBase object

evalml.pipelines.components.Estimator.predict

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

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

Returns Predicted values

Return type pd.Series

evalml.pipelines.components.Estimator.predict_proba

Estimator.**predict_proba**(*X*)
Make probability estimates for labels.

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

Returns Probability estimates

Return type pd.DataFrame

evalml.pipelines.components.Estimator.save

Estimator.**save**(*file_path*, *pickle_protocol=4*)
Saves component at file path

Parameters

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

Returns None

5.5.2 Component Utils

allowed_model_families	List the model types allowed for a particular problem type.
get_estimators	Returns the estimators allowed for a particular problem type.

evalml.pipelines.components.utils.allowed_model_families

evalml.pipelines.components.utils.**allowed_model_families**(*problem_type*)

List the model types allowed for a particular problem type.

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

Returns a list of model families

Return type list[*ModelFamily*]

evalml.pipelines.components.utils.get_estimators

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

Returns the estimators allowed for a particular problem type.

Can also optionally filter by a list of model types.

Parameters

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

Returns a list of estimator subclasses

Return type list[class]

5.5.3 Transformers

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

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

Continued on next page

Table 43 – continued from previous page

RFClassifierSelectFromModel	Selects top features based on importance weights using a Random Forest classifier.
DropNullColumns	Transformer to drop features whose percentage of NaN values exceeds a specified threshold
DateTimeFeaturizer	Transformer that can automatically featurize DateTime columns.
TextFeaturizer	Transformer that can automatically featurize text columns.

evalml.pipelines.components.DropColumns

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

Drops specified columns in input data.

name = 'Drop Columns Transformer'

model_family = 'none'

hyperparameter_ranges = {}

default_parameters = {'columns': None}

Instance attributes

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

Methods:

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

evalml.pipelines.components.DropColumns.__init__

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

Initalizes an transformer that drops specified columns in input data.

Parameters **columns** (*list(string)*) – List of column names, used to determine which columns to drop.

evalml.pipelines.components.DropColumns.clone

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

Constructs a new component with the same parameters

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

Returns A new instance of this component with identical parameters

evalml.pipelines.components.DropColumns.describe

DropColumns.**describe**(*print_name=False*, *return_dict=False*)

Describe a component and its parameters

Parameters

- **print_name** (*bool, optional*) – whether to print name of component
- **return_dict** (*bool, optional*) – whether to return description as dictionary in the format {"name": name, "parameters": parameters}

Returns prints and returns dictionary

Return type None or dict

evalml.pipelines.components.DropColumns.fit

DropColumns.**fit**(*X*, *y=None*)

'Fits' the transformer by checking if the column names are present in the dataset.

Parameters

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

Returns None.

evalml.pipelines.components.DropColumns.fit_transform

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

Fit transformer to data, then transform data.

Parameters

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

Returns Transformed X.

Return type pd.DataFrame

evalml.pipelines.components.DropColumns.load

static DropColumns.**load**(*file_path*)

Loads component at file path

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

Returns ComponentBase object

evalml.pipelines.components.DropColumns.save

DropColumns.**save**(*file_path*, *pickle_protocol=4*)

Saves component at file path

Parameters

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

Returns None

evalml.pipelines.components.DropColumns.transform

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

Transforms data X by dropping columns.

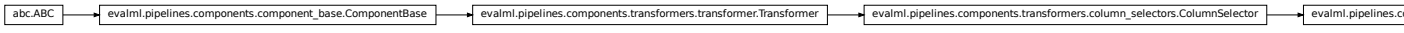
Parameters

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

Returns Transformed X.

Return type pd.DataFrame

evalml.pipelines.components.SelectColumns



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

Selects specified columns in input data.

name = 'Select Columns Transformer'

model_family = 'none'

hyperparameter_ranges = {}

default_parameters = {'columns': None}

Instance attributes

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

Methods:

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

evalml.pipelines.components.SelectColumns.__init__

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

Initalizes an transformer that drops specified columns in input data.

Parameters **columns** (*list(string)*) – List of column names, used to determine which columns to drop.

evalml.pipelines.components.SelectColumns.clone

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

Constructs a new component with the same parameters

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

Returns A new instance of this component with identical parameters

evalml.pipelines.components.SelectColumns.describe

SelectColumns.**describe**(*print_name=False*, *return_dict=False*)

Describe a component and its parameters

Parameters

- **print_name** (*bool, optional*) – whether to print name of component
- **return_dict** (*bool, optional*) – whether to return description as dictionary in the format {"name": name, "parameters": parameters}

Returns prints and returns dictionary

Return type None or dict

evalml.pipelines.components.SelectColumns.fit

SelectColumns.**fit**(*X*, *y=None*)

'Fits' the transformer by checking if the column names are present in the dataset.

Parameters

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

Returns None.

evalml.pipelines.components.SelectColumns.fit_transform

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

Fit transformer to data, then transform data.

Parameters

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

Returns Transformed X.

Return type pd.DataFrame

evalml.pipelines.components.SelectColumns.load

static SelectColumns.**load**(*file_path*)

Loads component at file path

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

Returns ComponentBase object

evalml.pipelines.components.SelectColumns.save

SelectColumns.**save**(*file_path*, *pickle_protocol=4*)

Saves component at file path

Parameters

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

Returns None

evalml.pipelines.components.SelectColumns.transform

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

Transforms data X by selecting columns.

Parameters

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

Returns Transformed X.

Return type pd.DataFrame

evalml.pipelines.components.OneHotEncoder

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

One-hot encoder to encode non-numeric data.

name = 'One Hot Encoder'

model_family = 'none'

hyperparameter_ranges = {}

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

Instance attributes

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

Methods:

__init__	Initalizes an transformer that encodes categorical features in a one-hot numeric array."
categories	Returns a list of the unique categories to be encoded for the particular feature, in order.
clone	Constructs a new component with the same parameters
describe	Describe a component and its parameters
fit	Fits component to data
fit_transform	Fits on X and transforms X

Continued on next page

Table 49 – continued from previous page

get_feature_names	Return feature names for the input features after fitting.
load	Loads component at file path
save	Saves component at file path
transform	One-hot encode the input DataFrame.

evalml.pipelines.components.OneHotEncoder.__init__

OneHotEncoder.**__init__**(*top_n=10*, *categories=None*, *drop=None*, *handle_unknown='ignore'*, *handle_missing='error'*, *random_state=0*, ***kwargs*)

Initalizes an transformer that encodes categorical features in a one-hot numeric array."

Parameters

- **top_n** (*int*) – Number of categories per column to encode. If None, all categories will be encoded. Otherwise, the *n* most frequent will be encoded and all others will be dropped. Defaults to 10.
- **categories** (*list*) – A two dimensional list of categories, where *categories[i]* is a list of the categories for the column at index *i*. This can also be *None*, or *"auto"* if *top_n* is not None. Defaults to None.
- **drop** (*string*) – Method ("first" or "if_binary") to use to drop one category per feature. Can also be a list specifying which method to use for each feature. Defaults to None.
- **handle_unknown** (*string*) – Whether to ignore or error for unknown categories for a feature encountered during *fit* or *transform*. If either *top_n* or *categories* is used to limit the number of categories per column, this must be "ignore". Defaults to "ignore".
- **handle_missing** (*string*) – Options for how to handle missing (NaN) values encountered during *fit* or *transform*. If this is set to "as_category" and NaN values are within the *n* most frequent, "nan" values will be encoded as their own column. If this is set to "error", any missing values encountered will raise an error. Defaults to "error".

evalml.pipelines.components.OneHotEncoder.categories

OneHotEncoder.**categories**(*feature_name*)

Returns a list of the unique categories to be encoded for the particular feature, in order.

Parameters **feature_name** (*str*) – the name of any feature provided to one-hot encoder during fit

Returns the unique categories, in the same dtype as they were provided during fit

Return type np.array

evalml.pipelines.components.OneHotEncoder.clone

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

Constructs a new component with the same parameters

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

Returns A new instance of this component with identical parameters

evalml.pipelines.components.OneHotEncoder.describe

OneHotEncoder.**describe**(*print_name=False*, *return_dict=False*)

Describe a component and its parameters

Parameters

- **print_name** (*bool, optional*) – whether to print name of component
- **return_dict** (*bool, optional*) – whether to return description as dictionary in the format {"name": name, "parameters": parameters}

Returns prints and returns dictionary

Return type None or dict

evalml.pipelines.components.OneHotEncoder.fit

OneHotEncoder.**fit**(*X*, *y=None*)

Fits component to data

Parameters

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

Returns self

evalml.pipelines.components.OneHotEncoder.fit_transform

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

Fits on X and transforms X

Parameters

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

Returns Transformed X

Return type pd.DataFrame

evalml.pipelines.components.OneHotEncoder.get_feature_names

OneHotEncoder.**get_feature_names**()

Return feature names for the input features after fitting.

Returns The feature names after encoding, provided in the same order as input_features.

Return type np.array

evalml.pipelines.components.OneHotEncoder.load

static OneHotEncoder.**load**(*file_path*)

Loads component at file path

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

Returns ComponentBase object

evalml.pipelines.components.OneHotEncoder.save

OneHotEncoder.**save**(*file_path*, *pickle_protocol=4*)

Saves component at file path

Parameters

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

Returns None

evalml.pipelines.components.OneHotEncoder.transform

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

One-hot encode the input DataFrame.

Parameters

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

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

evalml.pipelines.components.PerColumnImputer

class evalml.pipelines.components.**PerColumnImputer**(*impute_strategies=None*, *default_impute_strategy='most_frequent'*, *random_state=0*, ***kwargs*)

Imputes missing data according to a specified imputation strategy per column

name = 'Per Column Imputer'

model_family = 'none'

hyperparameter_ranges = {}

default_parameters = {'default_impute_strategy': 'most_frequent', 'impute_strategies'

Instance attributes

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

Methods:

__init__	Initializes a transformer that imputes missing data according to the specified imputation strategy per column."
clone	Constructs a new component with the same parameters
describe	Describe a component and its parameters
fit	Fits imputers on input data
fit_transform	Fits imputer and imputes missing values in input data.
load	Loads component at file path
save	Saves component at file path
transform	Transforms input data by imputing missing values

evalml.pipelines.components.PerColumnImputer.__init__

PerColumnImputer.**__init__**(*impute_strategies=None*, *default_impute_strategy='most_frequent'*, *random_state=0*, ***kwargs*)

Initializes a transformer that imputes missing data according to the specified imputation strategy per column."

Parameters

- **impute_strategies** (*dict*) – Column and {"impute_strategy": strategy, "fill_value":value} pairings. Valid values for impute strategy include "mean", "median", "most_frequent", "constant" for numerical data, and "most_frequent", "constant" for object data types. Defaults to "most_frequent" for all columns.

  When impute_strategy == "constant", fill_value is used to replace missing data. Defaults to 0 when imputing numerical data and "missing_value" for strings or object data types.

- **default_impute_strategy** (*str*) – Impute strategy to fall back on when none is provided for a certain column. Valid values include "mean", "median", "most_frequent", "constant" for numerical data, and "most_frequent", "constant" for object data types. Defaults to "most_frequent"

evalml.pipelines.components.PerColumnImputer.clone

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

Constructs a new component with the same parameters

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

Returns A new instance of this component with identical parameters

evalml.pipelines.components.PerColumnImputer.describe

PerColumnImputer.**describe**(*print_name=False*, *return_dict=False*)
Describe a component and its parameters

Parameters

- **print_name** (*bool, optional*) – whether to print name of component
- **return_dict** (*bool, optional*) – whether to return description as dictionary in the format {"name": name, "parameters": parameters}

Returns prints and returns dictionary

Return type None or dict

evalml.pipelines.components.PerColumnImputer.fit

PerColumnImputer.**fit**(*X*, *y=None*)
Fits imputers on input data

Parameters

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

Returns self

evalml.pipelines.components.PerColumnImputer.fit_transform

PerColumnImputer.**fit_transform**(*X*, *y=None*)
Fits imputer and imputes missing values in input data.

Parameters

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

Returns Transformed X

Return type pd.DataFrame

evalml.pipelines.components.PerColumnImputer.load

static PerColumnImputer.**load**(*file_path*)
Loads component at file path

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

Returns ComponentBase object

evalml.pipelines.components.PerColumnImputer.save

PerColumnImputer.**save**(*file_path*, *pickle_protocol=4*)
Saves component at file path

Parameters

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

Returns None

evalml.pipelines.components.PerColumnImputer.transform

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

Transforms input data by imputing missing values

Parameters

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

Returns Transformed X

Return type pd.DataFrame

evalml.pipelines.components.Imputer

class evalml.pipelines.components.**Imputer**(*categorical_impute_strategy='most_frequent'*, *categorical_fill_value=None*, *numeric_impute_strategy='mean'*, *numeric_fill_value=None*, *random_state=0*, ***kwargs*)

Imputes missing data according to a specified imputation strategy.

name = 'Imputer'

model_family = 'none'

hyperparameter_ranges = {'categorical_impute_strategy': ['most_frequent'], 'numeric_i

default_parameters = {'categorical_fill_value': None, 'categorical_impute_strategy':

Instance attributes

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

Methods:

__init__	Initalizes an transformer that imputes missing data according to the specified imputation strategy."
clone	Constructs a new component with the same parameters
describe	Describe a component and its parameters
fit	Fits imputer to data. 'None' values are converted to np.nan before imputation and are
fit_transform	Fits on X and transforms X
load	Loads component at file path
save	Saves component at file path
transform	Transforms data X by imputing missing values. 'None' values are converted to np.nan before imputation and are

evalml.pipelines.components.Imputer.__init__

Imputer.**__init__**(*categorical_impute_strategy='most_frequent'*, *categorical_fill_value=None*, *numeric_impute_strategy='mean'*, *numeric_fill_value=None*, *random_state=0*, ***kwargs*)

Initalizes an transformer that imputes missing data according to the specified imputation strategy."

Parameters

- **categorical_impute_strategy** (*string*) – Impute strategy to use for string, object, boolean, categorical dtypes. Valid values include "most_frequent" and "constant".
- **numeric_impute_strategy** (*string*) – Impute strategy to use for numeric columns. Valid values include "mean", "median", "most_frequent", and "constant".
- **categorical_fill_value** (*string*) – When categorical_impute_strategy == "constant", fill_value is used to replace missing data. The default value of None will fill with the string "missing_value".
- **numeric_fill_value** (*int, float*) – When numeric_impute_strategy == "constant", fill_value is used to replace missing data. The default value of None will fill with 0.

evalml.pipelines.components.Imputer.clone

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

Constructs a new component with the same parameters

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

Returns A new instance of this component with identical parameters

evalml.pipelines.components.Imputer.describe

Imputer.**describe**(*print_name=False*, *return_dict=False*)

Describe a component and its parameters

Parameters

- **print_name** (*bool, optional*) – whether to print name of component

- **return_dict** (*bool, optional*) – whether to return description as dictionary in the format {"name": name, "parameters": parameters}

Returns prints and returns dictionary

Return type None or dict

evalml.pipelines.components.Imputer.fit

Imputer.**fit**(*X*, *y=None*)

Fits imputer to data. 'None' values are converted to np.nan before imputation and are treated as the same.

Parameters

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

Returns self

evalml.pipelines.components.Imputer.fit_transform

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

Fits on X and transforms X

Parameters

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

Returns Transformed X

Return type pd.DataFrame

evalml.pipelines.components.Imputer.load

static Imputer.**load**(*file_path*)

Loads component at file path

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

Returns ComponentBase object

evalml.pipelines.components.Imputer.save

Imputer.**save**(*file_path*, *pickle_protocol=4*)

Saves component at file path

Parameters

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

Returns None

evalml.pipelines.components.Imputer.transform

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

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

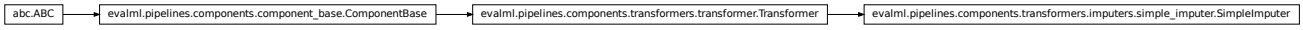
Parameters

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

Returns Transformed X

Return type pd.DataFrame

evalml.pipelines.components.SimpleImputer



class evalml.pipelines.components.**SimpleImputer**(*impute_strategy='most_frequent'*, *fill_value=None*, *random_state=0*, ***kwargs*)

Imputes missing data according to a specified imputation strategy.

name = 'Simple Imputer'

model_family = 'none'

hyperparameter_ranges = {'impute_strategy': ['mean', 'median', 'most_frequent']}

default_parameters = {'fill_value': None, 'impute_strategy': 'most_frequent'}

Instance attributes

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

Methods:

__init__	Initalizes an transformer that imputes missing data according to the specified imputation strategy."
clone	Constructs a new component with the same parameters
describe	Describe a component and its parameters
fit	Fits imputer to data. 'None' values are converted to np.nan before imputation and are
fit_transform	Fits on X and transforms X
load	Loads component at file path

Continued on next page

Table 55 – continued from previous page

save	Saves component at file path
transform	Transforms data X by imputing missing values. 'None' values are converted to np.nan before imputation and are

evalml.pipelines.components.SimpleImputer.__init__

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

Initalizes an transformer that imputes missing data according to the specified imputation strategy."

Parameters

- **impute_strategy** (*string*) – Impute strategy to use. Valid values include "mean", "median", "most_frequent", "constant" for numerical data, and "most_frequent", "constant" for object data types.
- **fill_value** (*string*) – When impute_strategy == "constant", fill_value is used to replace missing data. Defaults to 0 when imputing numerical data and "missing_value" for strings or object data types.

evalml.pipelines.components.SimpleImputer.clone

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

Constructs a new component with the same parameters

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

Returns A new instance of this component with identical parameters

evalml.pipelines.components.SimpleImputer.describe

SimpleImputer.**describe**(*print_name=False*, *return_dict=False*)

Describe a component and its parameters

Parameters

- **print_name** (*bool, optional*) – whether to print name of component
- **return_dict** (*bool, optional*) – whether to return description as dictionary in the format {"name": name, "parameters": parameters}

Returns prints and returns dictionary

Return type None or dict

evalml.pipelines.components.SimpleImputer.fit

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

Fits imputer to data. 'None' values are converted to np.nan before imputation and are treated as the same.

Parameters

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

Returns self

evalml.pipelines.components.SimpleImputer.fit_transform

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

Fits on X and transforms X

Parameters

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

Returns Transformed X

Return type pd.DataFrame

evalml.pipelines.components.SimpleImputer.load

static SimpleImputer.**load**(*file_path*)

Loads component at file path

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

Returns ComponentBase object

evalml.pipelines.components.SimpleImputer.save

SimpleImputer.**save**(*file_path*, *pickle_protocol=4*)

Saves component at file path

Parameters

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

Returns None

evalml.pipelines.components.SimpleImputer.transform

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

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

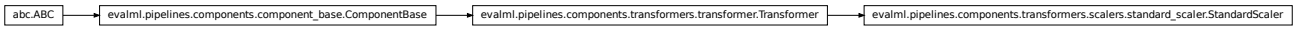
Parameters

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

Returns Transformed X

Return type pd.DataFrame

evalml.pipelines.components.StandardScaler



class evalml.pipelines.components.**StandardScaler**(*random_state=0*, ***kwargs*)

Standardize features: removes mean and scales to unit variance.

name = 'Standard Scaler'

model_family = 'none'

hyperparameter_ranges = {}

default_parameters = {}

Instance attributes

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

Methods:

__init__	Initialize self.
clone	Constructs a new component with the same parameters
describe	Describe a component and its parameters
fit	Fits component to data
fit_transform	Fits on X and transforms X
load	Loads component at file path
save	Saves component at file path
transform	Transforms data X

evalml.pipelines.components.StandardScaler.__init__

StandardScaler.**__init__**(*random_state=0*, ***kwargs*)

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

evalml.pipelines.components.StandardScaler.clone

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

Constructs a new component with the same parameters

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

Returns A new instance of this component with identical parameters

evalml.pipelines.components.StandardScaler.describe

StandardScaler.**describe**(*print_name=False*, *return_dict=False*)
Describe a component and its parameters

Parameters

- **print_name** (*bool, optional*) – whether to print name of component
- **return_dict** (*bool, optional*) – whether to return description as dictionary in the format {"name": name, "parameters": parameters}

Returns prints and returns dictionary

Return type None or dict

evalml.pipelines.components.StandardScaler.fit

StandardScaler.**fit**(*X*, *y=None*)
Fits component to data

Parameters

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

Returns self

evalml.pipelines.components.StandardScaler.fit_transform

StandardScaler.**fit_transform**(*X*, *y=None*)
Fits on X and transforms X

Parameters

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

Returns Transformed X

Return type pd.DataFrame

evalml.pipelines.components.StandardScaler.load

static StandardScaler.**load**(*file_path*)
Loads component at file path

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

Returns ComponentBase object

evalml.pipelines.components.StandardScaler.save

StandardScaler.**save**(*file_path*, *pickle_protocol=4*)

Saves component at file path

Parameters

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

Returns None

evalml.pipelines.components.StandardScaler.transform

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

Transforms data X

Parameters

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

Returns Transformed X

Return type pd.DataFrame

evalml.pipelines.components.RFRegressorSelectFromModel

class evalml.pipelines.components.**RFRegressorSelectFromModel**(*number_features=None*, *n_estimators=10*, *max_depth=None*, *percent_features=0.5*, *threshold=-inf*, *n_jobs=-1*, *random_state=0*, ***kwargs*)

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

```
name = 'RF Regressor Select From Model'

model_family = 'none'

hyperparameter_ranges = {'percent_features':  Real(low=0.01, high=1, prior='uniform',

default_parameters = {'max_depth':  None, 'n_estimators':  10, 'n_jobs':  -1, 'number_
```

Instance attributes

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

Methods:

__init__	Initialize self.
clone	Constructs a new component with the same parameters
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_names	Get names of selected features.
load	Loads component at file path
save	Saves component at file path
transform	Transforms data X by selecting features

evalml.pipelines.components.RFRegressorSelectFromModel.__init__

RFRegressorSelectFromModel.**__init__**(*number_features=None*, *n_estimators=10*, *max_depth=None*, *percent_features=0.5*, *threshold=-inf*, *n_jobs=-1*, *random_state=0*, ***kwargs*)

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

evalml.pipelines.components.RFRegressorSelectFromModel.clone

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

Constructs a new component with the same parameters

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

Returns A new instance of this component with identical parameters

evalml.pipelines.components.RFRegressorSelectFromModel.describe

RFRegressorSelectFromModel.**describe**(*print_name=False*, *return_dict=False*)

Describe a component and its parameters

Parameters

- **print_name** (*bool, optional*) – whether to print name of component
- **return_dict** (*bool, optional*) – whether to return description as dictionary in the format {"name": name, "parameters": parameters}

Returns prints and returns dictionary

Return type None or dict

evalml.pipelines.components.RFRegressorSelectFromModel.fit

RFRegressorSelectFromModel.**fit**(*X*, *y=None*)
Fits component to data

Parameters

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

Returns self

evalml.pipelines.components.RFRegressorSelectFromModel.fit_transform

RFRegressorSelectFromModel.**fit_transform**(*X*, *y=None*)
Fits feature selector on data X then transforms X by selecting features

Parameters

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

Returns Transformed X

Return type pd.DataFrame

evalml.pipelines.components.RFRegressorSelectFromModel.get_names

RFRegressorSelectFromModel.**get_names**()
Get names of selected features.

Returns list of the names of features selected

evalml.pipelines.components.RFRegressorSelectFromModel.load

static RFRegressorSelectFromModel.**load**(*file_path*)
Loads component at file path

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

Returns ComponentBase object

evalml.pipelines.components.RFRegressorSelectFromModel.save

RFRegressorSelectFromModel.**save**(*file_path*, *pickle_protocol=4*)
Saves component at file path

Parameters

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

Returns None

evalml.pipelines.components.RFRegressorSelectFromModel.transform

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

Transforms data X by selecting features

Parameters

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

Returns Transformed X

Return type pd.DataFrame

evalml.pipelines.components.RFClassifierSelectFromModel

class evalml.pipelines.components.**RFClassifierSelectFromModel**(*number_features=None*, *n_estimators=10*, *max_depth=None*, *percent_features=0.5*, *threshold=-inf*, *n_jobs=-1*, *random_state=0*, ***kwargs*)

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

```
name = 'RF Classifier Select From Model'

model_family = 'none'

hyperparameter_ranges = {'percent_features':  Real(low=0.01, high=1, prior='uniform',

default_parameters = {'max_depth':  None, 'n_estimators':  10, 'n_jobs':  -1, 'number_
```

Instance attributes

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

Methods:

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

Continued on next page

Table 61 – continued from previous page

fit_transform	Fits feature selector on data X then transforms X by selecting features
get_names	Get names of selected features.
load	Loads component at file path
save	Saves component at file path
transform	Transforms data X by selecting features

evalml.pipelines.components.RFClassifierSelectFromModel.__init__

RFClassifierSelectFromModel.**__init__**(*number_features=None*, *n_estimators=10*, *max_depth=None*, *percent_features=0.5*, *threshold=-inf*, *n_jobs=-1*, *random_state=0*, ***kwargs*)

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

evalml.pipelines.components.RFClassifierSelectFromModel.clone

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

Constructs a new component with the same parameters

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

Returns A new instance of this component with identical parameters

evalml.pipelines.components.RFClassifierSelectFromModel.describe

RFClassifierSelectFromModel.**describe**(*print_name=False*, *return_dict=False*)

Describe a component and its parameters

Parameters

- **print_name** (*bool, optional*) – whether to print name of component
- **return_dict** (*bool, optional*) – whether to return description as dictionary in the format {"name": name, "parameters": parameters}

Returns prints and returns dictionary

Return type None or dict

evalml.pipelines.components.RFClassifierSelectFromModel.fit

RFClassifierSelectFromModel.**fit**(*X*, *y=None*)

Fits component to data

Parameters

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

Returns self

evalml.pipelines.components.RFClassifierSelectFromModel.fit_transform

RFClassifierSelectFromModel.**fit_transform**(*X*, *y=None*)
Fits feature selector on data X then transforms X by selecting features

Parameters

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

Returns Transformed X

Return type pd.DataFrame

evalml.pipelines.components.RFClassifierSelectFromModel.get_names

RFClassifierSelectFromModel.**get_names**()
Get names of selected features.

Returns list of the names of features selected

evalml.pipelines.components.RFClassifierSelectFromModel.load

static RFClassifierSelectFromModel.**load**(*file_path*)
Loads component at file path

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

Returns ComponentBase object

evalml.pipelines.components.RFClassifierSelectFromModel.save

RFClassifierSelectFromModel.**save**(*file_path*, *pickle_protocol=4*)
Saves component at file path

Parameters

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

Returns None

evalml.pipelines.components.RFClassifierSelectFromModel.transform

RFClassifierSelectFromModel.**transform**(*X*, *y=None*)
Transforms data X by selecting features

Parameters

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

Returns Transformed X

Return type pd.DataFrame

evalml.pipelines.components.DropNullColumns

class evalml.pipelines.components.**DropNullColumns**(*pct_null_threshold=1.0*, *random_state=0*, ***kwargs*)

Transformer to drop features whose percentage of NaN values exceeds a specified threshold

name = 'Drop Null Columns Transformer'

model_family = 'none'

hyperparameter_ranges = {}

default_parameters = {'pct_null_threshold': 1.0}

Instance attributes

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

Methods:

__init__	Initalizes an transformer to drop features whose percentage of NaN values exceeds a specified threshold.
clone	Constructs a new component with the same parameters
describe	Describe a component and its parameters
fit	Fits component to data
fit_transform	Fits on X and transforms X
load	Loads component at file path
save	Saves component at file path
transform	Transforms data X by dropping columns that exceed the threshold of null values.

evalml.pipelines.components.DropNullColumns.__init__

DropNullColumns.**__init__**(*pct_null_threshold=1.0*, *random_state=0*, ***kwargs*)

Initalizes an transformer to drop features whose percentage of NaN values exceeds a specified threshold.

Parameters **pct_null_threshold** (*float*) – The percentage of NaN values in an input feature to drop. Must be a value between [0, 1] inclusive. If equal to 0.0, will drop columns with any null values. If equal to 1.0, will drop columns with all null values. Defaults to 0.95.

evalml.pipelines.components.DropNullColumns.clone

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

Constructs a new component with the same parameters

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

Returns A new instance of this component with identical parameters

evalml.pipelines.components.DropNullColumns.describe

DropNullColumns.**describe**(*print_name=False*, *return_dict=False*)

Describe a component and its parameters

Parameters

- **print_name** (*bool, optional*) – whether to print name of component
- **return_dict** (*bool, optional*) – whether to return description as dictionary in the format {"name": name, "parameters": parameters}

Returns prints and returns dictionary

Return type None or dict

evalml.pipelines.components.DropNullColumns.fit

DropNullColumns.**fit**(*X*, *y=None*)

Fits component to data

Parameters

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

Returns self

evalml.pipelines.components.DropNullColumns.fit_transform

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

Fits on X and transforms X

Parameters

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

Returns Transformed X

Return type pd.DataFrame

evalml.pipelines.components.DropNullColumns.load

static DropNullColumns.**load**(*file_path*)

Loads component at file path

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

Returns ComponentBase object

evalml.pipelines.components.DropNullColumns.save

DropNullColumns.**save**(*file_path*, *pickle_protocol=4*)

Saves component at file path

Parameters

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

Returns None

evalml.pipelines.components.DropNullColumns.transform

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

Transforms data X by dropping columns that exceed the threshold of null values.

Parameters

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

Returns Transformed X

Return type pd.DataFrame

evalml.pipelines.components.DateTimeFeaturizer

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

Transformer that can automatically featurize DateTime columns.

name = 'DateTime Featurization Component'

model_family = 'none'

hyperparameter_ranges = {}

default_parameters = {'features_to_extract': ['year', 'month', 'day_of_week', 'hour']

Instance attributes

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

Methods:

__init__	Extracts features from DateTime columns
clone	Constructs a new component with the same parameters
describe	Describe a component and its parameters
fit	Fits component to data
fit_transform	Fits on X and transforms X
load	Loads component at file path
save	Saves component at file path
transform	Transforms data X by creating new features using existing DateTime columns, and then dropping those DateTime columns

evalml.pipelines.components.DateTimeFeaturizer.__init__

DateTimeFeaturizer.**__init__**(*features_to_extract=None*, *random_state=0*, ***kwargs*)

Extracts features from DateTime columns

Parameters

- **features_to_extract** (*list*) – List of features to extract. Valid options include "year", "month", "day_of_week", "hour".
- **random_state** (*int, np.random.RandomState*) – Seed for the random number generator.

evalml.pipelines.components.DateTimeFeaturizer.clone

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

Constructs a new component with the same parameters

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

Returns A new instance of this component with identical parameters

evalml.pipelines.components.DateTimeFeaturizer.describe

DateTimeFeaturizer.**describe**(*print_name=False*, *return_dict=False*)

Describe a component and its parameters

Parameters

- **print_name** (*bool, optional*) – whether to print name of component
- **return_dict** (*bool, optional*) – whether to return description as dictionary in the format {"name": name, "parameters": parameters}

Returns prints and returns dictionary

Return type None or dict

evalml.pipelines.components.DateTimeFeaturizer.fit

DateTimeFeaturizer.**fit**(*X*, *y=None*)

Fits component to data

Parameters

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

Returns self

evalml.pipelines.components.DateTimeFeaturizer.fit_transform

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

Fits on X and transforms X

Parameters

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

Returns Transformed X

Return type pd.DataFrame

evalml.pipelines.components.DateTimeFeaturizer.load

static DateTimeFeaturizer.**load**(*file_path*)

Loads component at file path

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

Returns ComponentBase object

evalml.pipelines.components.DateTimeFeaturizer.save

DateTimeFeaturizer.**save**(*file_path*, *pickle_protocol=4*)

Saves component at file path

Parameters

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

Returns None

evalml.pipelines.components.DateTimeFeaturizer.transform

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

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

Parameters

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

Returns Transformed X

Return type pd.DataFrame

evalml.pipelines.components.TextFeaturizer

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

Transformer that can automatically featurize text columns.

name = 'Text Featurization Component'

model_family = 'none'

hyperparameter_ranges = {}

default_parameters = {'text_columns': None}

Instance attributes

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

Methods:

__init__	Extracts features from text columns using featuretools' nlp_primitives
clone	Constructs a new component with the same parameters
describe	Describe a component and its parameters
fit	Fits component to data
fit_transform	Fits on X and transforms X
load	Loads component at file path
save	Saves component at file path
transform	Transforms data X by creating new features using existing text columns

evalml.pipelines.components.TextFeaturizer.__init__

TextFeaturizer.**__init__**(*text_columns=None*, *random_state=0*, ***kwargs*)

Extracts features from text columns using featuretools' nlp_primitives

Parameters

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

evalml.pipelines.components.TextFeaturizer.clone

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

Constructs a new component with the same parameters

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

Returns A new instance of this component with identical parameters

evalml.pipelines.components.TextFeaturizer.describe

TextFeaturizer.**describe**(*print_name=False*, *return_dict=False*)

Describe a component and its parameters

Parameters

- **print_name** (*bool, optional*) – whether to print name of component
- **return_dict** (*bool, optional*) – whether to return description as dictionary in the format {"name": name, "parameters": parameters}

Returns prints and returns dictionary

Return type None or dict

evalml.pipelines.components.TextFeaturizer.fit

TextFeaturizer.**fit**(*X*, *y=None*)

Fits component to data

Parameters

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

Returns self

evalml.pipelines.components.TextFeaturizer.fit_transform

TextFeaturizer.**fit_transform**(*X*, *y=None*)

Fits on X and transforms X

Parameters

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

Returns Transformed X

Return type pd.DataFrame

evalml.pipelines.components.TextFeaturizer.load

static TextFeaturizer.**load**(*file_path*)
Loads component at file path

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

Returns ComponentBase object

evalml.pipelines.components.TextFeaturizer.save

TextFeaturizer.**save**(*file_path*, *pickle_protocol=4*)
Saves component at file path

Parameters

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

Returns None

evalml.pipelines.components.TextFeaturizer.transform

TextFeaturizer.**transform**(*X*, *y=None*)
Transforms data X by creating new features using existing text columns

Parameters

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

Returns Transformed X

Return type pd.DataFrame

5.5.4 Estimators

Classifiers

Classifiers are components that output a predicted class label.

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

evalml.pipelines.components.CatBoostClassifier

class evalml.pipelines.components.**CatBoostClassifier**(*n_estimators=10*, *eta=0.03*, *max_depth=6*, *bootstrap_type=None*, *silent=True*, *allow_writing_files=False*, *random_state=0*, ***kwargs*)

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

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

name = 'CatBoost Classifier'

model_family = 'catboost'

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

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

default_parameters = {'allow_writing_files': False, 'bootstrap_type': None, 'eta':

Instance attributes

SEED_MAX	
SEED_MIN	
feature_importance	Return an attribute of instance, which is of type owner.
needs_fitting	
parameters	Returns the parameters which were used to initialize the component

Methods:

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

evalml.pipelines.components.CatBoostClassifier.__init__

CatBoostClassifier.**__init__**(*n_estimators=10*, *eta=0.03*, *max_depth=6*, *bootstrap_type=None*, *silent=True*, *allow_writing_files=False*, *random_state=0*, ***kwargs*)

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

evalml.pipelines.components.CatBoostClassifier.clone

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

Constructs a new component with the same parameters

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

Returns A new instance of this component with identical parameters

evalml.pipelines.components.CatBoostClassifier.describe

CatBoostClassifier.**describe**(*print_name=False*, *return_dict=False*)

Describe a component and its parameters

Parameters

- **print_name** (*bool, optional*) – whether to print name of component
- **return_dict** (*bool, optional*) – whether to return description as dictionary in the format {"name": name, "parameters": parameters}

Returns prints and returns dictionary

Return type None or dict

evalml.pipelines.components.CatBoostClassifier.fit

CatBoostClassifier.**fit**(*X*, *y=None*)

Fits component to data

Parameters

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

Returns self

evalml.pipelines.components.CatBoostClassifier.load

static CatBoostClassifier.**load**(*file_path*)

Loads component at file path

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

Returns ComponentBase object

evalml.pipelines.components.CatBoostClassifier.predict

CatBoostClassifier.**predict**(*X*)

Make predictions using selected features.

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

Returns Predicted values

Return type pd.Series

evalml.pipelines.components.CatBoostClassifier.predict_proba

CatBoostClassifier.**predict_proba**(*X*)

Make probability estimates for labels.

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

Returns Probability estimates

Return type pd.DataFrame

evalml.pipelines.components.CatBoostClassifier.save

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

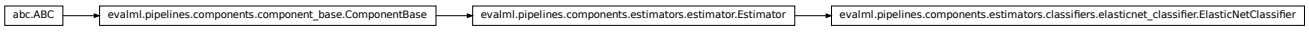
Saves component at file path

Parameters

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

Returns None

evalml.pipelines.components.ElasticNetClassifier



class evalml.pipelines.components.**ElasticNetClassifier**(*alpha=0.5*, *l1_ratio=0.5*, *n_jobs=-1*, *max_iter=1000*, *random_state=0*, *penalty='elasticnet'*, ***kwargs*)

Elastic Net Classifier.

name = 'Elastic Net Classifier'

model_family = 'linear_model'

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

hyperparameter_ranges = {'alpha': Real(low=0, high=1, prior='uniform', transform='ide

default_parameters = {'alpha': 0.5, 'l1_ratio': 0.5, 'loss': 'log', 'max_iter': 10

Instance attributes

feature_importance	Return an attribute of instance, which is of type owner.
needs_fitting	
parameters	Returns the parameters which were used to initialize the component

Methods:

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

evalml.pipelines.components.ElasticNetClassifier.__init__

ElasticNetClassifier.**__init__**(*alpha=0.5*, *l1_ratio=0.5*, *n_jobs=-1*, *max_iter=1000*, *random_state=0*, *penalty='elasticnet'*, ***kwargs*)

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

evalml.pipelines.components.ElasticNetClassifier.clone

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

Constructs a new component with the same parameters

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

Returns A new instance of this component with identical parameters

evalml.pipelines.components.ElasticNetClassifier.describe

ElasticNetClassifier.**describe**(*print_name=False*, *return_dict=False*)

Describe a component and its parameters

Parameters

- **print_name** (*bool, optional*) – whether to print name of component
- **return_dict** (*bool, optional*) – whether to return description as dictionary in the format {"name": name, "parameters": parameters}

Returns prints and returns dictionary

Return type None or dict

evalml.pipelines.components.ElasticNetClassifier.fit

ElasticNetClassifier.**fit**(*X*, *y=None*)

Fits component to data

Parameters

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

Returns self

evalml.pipelines.components.ElasticNetClassifier.load

static ElasticNetClassifier.**load**(*file_path*)

Loads component at file path

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

Returns ComponentBase object

evalml.pipelines.components.ElasticNetClassifier.predict

ElasticNetClassifier.**predict**(*X*)

Make predictions using selected features.

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

Returns Predicted values

Return type pd.Series

evalml.pipelines.components.ElasticNetClassifier.predict_proba

ElasticNetClassifier.**predict_proba**(*X*)

Make probability estimates for labels.

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

Returns Probability estimates

Return type pd.DataFrame

evalml.pipelines.components.ElasticNetClassifier.save

ElasticNetClassifier.**save**(*file_path*, *pickle_protocol=4*)

Saves component at file path

Parameters

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

Returns None

evalml.pipelines.components.ExtraTreesClassifier

class evalml.pipelines.components.**ExtraTreesClassifier**(*n_estimators=100*, *max_features='auto'*, *max_depth=6*, *min_samples_split=2*, *min_weight_fraction_leaf=0.0*, *n_jobs=-1*, *random_state=0*, ***kwargs*)

Extra Trees Classifier.

name = 'Extra Trees Classifier'

model_family = 'extra_trees'

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

hyperparameter_ranges = {'max_depth': Integer(low=4, high=10, prior='uniform', transf

default_parameters = {'max_depth': 6, 'max_features': 'auto', 'min_samples_split':

Instance attributes

feature_importance	Return an attribute of instance, which is of type owner.
needs_fitting	
parameters	Returns the parameters which were used to initialize the component

Methods:

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

evalml.pipelines.components.ExtraTreesClassifier.__init__

ExtraTreesClassifier.**__init__**(*n_estimators=100*, *max_features='auto'*, *max_depth=6*, *min_samples_split=2*, *min_weight_fraction_leaf=0.0*, *n_jobs=-1*, *random_state=0*, ***kwargs*)

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

evalml.pipelines.components.ExtraTreesClassifier.clone

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

Constructs a new component with the same parameters

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

Returns A new instance of this component with identical parameters

evalml.pipelines.components.ExtraTreesClassifier.describe

ExtraTreesClassifier.**describe**(*print_name=False*, *return_dict=False*)

Describe a component and its parameters

Parameters

- **print_name** (*bool, optional*) – whether to print name of component
- **return_dict** (*bool, optional*) – whether to return description as dictionary in the format {"name": name, "parameters": parameters}

Returns prints and returns dictionary

Return type None or dict

evalml.pipelines.components.ExtraTreesClassifier.fit

ExtraTreesClassifier.**fit**(*X*, *y=None*)

Fits component to data

Parameters

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

Returns self

evalml.pipelines.components.ExtraTreesClassifier.load

static ExtraTreesClassifier.**load**(*file_path*)

Loads component at file path

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

Returns ComponentBase object

evalml.pipelines.components.ExtraTreesClassifier.predict

ExtraTreesClassifier.**predict**(*X*)

Make predictions using selected features.

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

Returns Predicted values

Return type pd.Series

evalml.pipelines.components.ExtraTreesClassifier.predict_proba

ExtraTreesClassifier.**predict_proba**(*X*)

Make probability estimates for labels.

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

Returns Probability estimates

Return type pd.DataFrame

evalml.pipelines.components.ExtraTreesClassifier.save

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

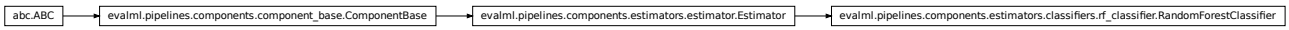
Saves component at file path

Parameters

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

Returns None

evalml.pipelines.components.RandomForestClassifier



class evalml.pipelines.components.**RandomForestClassifier**(*n_estimators=100*, *max_depth=6*, *n_jobs=-1*, *random_state=0*, ***kwargs*)

Random Forest Classifier.

name = 'Random Forest Classifier'

model_family = 'random_forest'

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

hyperparameter_ranges = {'max_depth': Integer(low=1, high=10, prior='uniform', transf

default_parameters = {'max_depth': 6, 'n_estimators': 100, 'n_jobs': -1}

Instance attributes

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

Continued on next page

Table 75 – continued from previous page

parameters	Returns the parameters which were used to initialize the component

Methods:

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

evalml.pipelines.components.RandomForestClassifier.__init__

RandomForestClassifier.**__init__**(*n_estimators=100*, *max_depth=6*, *n_jobs=-1*, *random_state=0*, ***kwargs*)

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

evalml.pipelines.components.RandomForestClassifier.clone

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

Constructs a new component with the same parameters

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

Returns A new instance of this component with identical parameters

evalml.pipelines.components.RandomForestClassifier.describe

RandomForestClassifier.**describe**(*print_name=False*, *return_dict=False*)

Describe a component and its parameters

Parameters

- **print_name** (*bool, optional*) – whether to print name of component
- **return_dict** (*bool, optional*) – whether to return description as dictionary in the format {"name": name, "parameters": parameters}

Returns prints and returns dictionary

Return type None or dict

evalml.pipelines.components.RandomForestClassifier.fit

RandomForestClassifier.**fit**(*X*, *y=None*)

Fits component to data

Parameters

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

Returns self

evalml.pipelines.components.RandomForestClassifier.load

static RandomForestClassifier.**load**(*file_path*)

Loads component at file path

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

Returns ComponentBase object

evalml.pipelines.components.RandomForestClassifier.predict

RandomForestClassifier.**predict**(*X*)

Make predictions using selected features.

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

Returns Predicted values

Return type pd.Series

evalml.pipelines.components.RandomForestClassifier.predict_proba

RandomForestClassifier.**predict_proba**(*X*)

Make probability estimates for labels.

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

Returns Probability estimates

Return type pd.DataFrame

evalml.pipelines.components.RandomForestClassifier.save

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

Saves component at file path

Parameters

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

Returns None

evalml.pipelines.components.LightGBMClassifier

class evalml.pipelines.components.**LightGBMClassifier**(*boosting_type='gbdt'*, *learning_rate=0.1*, *n_estimators=100*, *max_depth=0*, *num_leaves=31*, *min_child_samples=20*, *n_jobs=-1*, *random_state=0*, ***kwargs*)

LightGBM Classifier

name = 'LightGBM Classifier'

model_family = 'lightgbm'

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

hyperparameter_ranges = {'boosting_type': ['gbdt', 'dart', 'goss', 'rf'], 'learning_r

default_parameters = {'boosting_type': 'gbdt', 'learning_rate': 0.1, 'max_depth': 0

Instance attributes

SEED_MAX	
SEED_MIN	
feature_importance	Return an attribute of instance, which is of type owner.
needs_fitting	
parameters	Returns the parameters which were used to initialize the component

Methods:

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

evalml.pipelines.components.LightGBMClassifier.__init__

LightGBMClassifier.**__init__**(*boosting_type='gbdt'*, *learning_rate=0.1*, *n_estimators=100*, *max_depth=0*, *num_leaves=31*, *min_child_samples=20*, *n_jobs=-1*, *random_state=0*, ***kwargs*)

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

evalml.pipelines.components.LightGBMClassifier.clone

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

Constructs a new component with the same parameters

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

Returns A new instance of this component with identical parameters

evalml.pipelines.components.LightGBMClassifier.describe

LightGBMClassifier.**describe**(*print_name=False*, *return_dict=False*)

Describe a component and its parameters

Parameters

- **print_name** (*bool, optional*) – whether to print name of component
- **return_dict** (*bool, optional*) – whether to return description as dictionary in the format {"name": name, "parameters": parameters}

Returns prints and returns dictionary

Return type None or dict

evalml.pipelines.components.LightGBMClassifier.fit

LightGBMClassifier.**fit**(*X*, *y=None*)

Fits component to data

Parameters

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

Returns self

evalml.pipelines.components.LightGBMClassifier.load

static LightGBMClassifier.**load**(*file_path*)

Loads component at file path

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

Returns ComponentBase object

evalml.pipelines.components.LightGBMClassifier.predict

LightGBMClassifier.**predict**(*X*)

Make predictions using selected features.

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

Returns Predicted values

Return type pd.Series

evalml.pipelines.components.LightGBMClassifier.predict_proba

LightGBMClassifier.**predict_proba**(*X*)

Make probability estimates for labels.

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

Returns Probability estimates

Return type pd.DataFrame

evalml.pipelines.components.LightGBMClassifier.save

LightGBMClassifier.**save**(*file_path*, *pickle_protocol=4*)

Saves component at file path

Parameters

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

Returns None

evalml.pipelines.components.LogisticRegressionClassifier

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

Logistic Regression Classifier.

name = 'Logistic Regression Classifier'

model_family = 'linear_model'

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

```
hyperparameter_ranges = {'C': Real(low=0.01, high=10, prior='uniform', transform='iden
default_parameters = {'C': 1.0, 'multi_class': 'auto', 'n_jobs': -1, 'penalty': 'l2
```

Instance attributes

feature_importance	Return an attribute of instance, which is of type owner.
needs_fitting	
parameters	Returns the parameters which were used to initialize the component

Methods:

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

evalml.pipelines.components.LogisticRegressionClassifier.__init__

LogisticRegressionClassifier.**__init__**(*penalty='l2'*, *C=1.0*, *n_jobs=-1*, *multi_class='auto'*, *solver='lbfgs'*, *random_state=0*, ***kwargs*)

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

evalml.pipelines.components.LogisticRegressionClassifier.clone

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

Constructs a new component with the same parameters

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

Returns A new instance of this component with identical parameters

evalml.pipelines.components.LogisticRegressionClassifier.describe

LogisticRegressionClassifier.**describe**(*print_name=False*, *return_dict=False*)

Describe a component and its parameters

Parameters

- **print_name** (*bool, optional*) – whether to print name of component
- **return_dict** (*bool, optional*) – whether to return description as dictionary in the format {"name": name, "parameters": parameters}

Returns prints and returns dictionary

Return type None or dict

evalml.pipelines.components.LogisticRegressionClassifier.fit

LogisticRegressionClassifier.**fit**(*X*, *y=None*)

Fits component to data

Parameters

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

Returns self

evalml.pipelines.components.LogisticRegressionClassifier.load

static LogisticRegressionClassifier.**load**(*file_path*)

Loads component at file path

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

Returns ComponentBase object

evalml.pipelines.components.LogisticRegressionClassifier.predict

LogisticRegressionClassifier.**predict**(*X*)

Make predictions using selected features.

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

Returns Predicted values

Return type pd.Series

evalml.pipelines.components.LogisticRegressionClassifier.predict_proba

LogisticRegressionClassifier.**predict_proba**(*X*)

Make probability estimates for labels.

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

Returns Probability estimates

Return type pd.DataFrame

evalml.pipelines.components.LogisticRegressionClassifier.save

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

Saves component at file path

Parameters

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

Returns None

evalml.pipelines.components.XGBoostClassifier

class evalml.pipelines.components.**XGBoostClassifier**(*eta=0.1*, *max_depth=6*, *min_child_weight=1*, *n_estimators=100*, *random_state=0*, ***kwargs*)

XGBoost Classifier.

name = 'XGBoost Classifier'

model_family = 'xgboost'

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

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

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

Instance attributes

SEED_MAX	
SEED_MIN	
feature_importance	Return an attribute of instance, which is of type owner.
needs_fitting	
parameters	Returns the parameters which were used to initialize the component

Methods:

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

evalml.pipelines.components.XGBoostClassifier.__init__

XGBoostClassifier.**__init__**(*eta=0.1*, *max_depth=6*, *min_child_weight=1*, *n_estimators=100*, *random_state=0*, ***kwargs*)

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

evalml.pipelines.components.XGBoostClassifier.clone

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

Constructs a new component with the same parameters

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

Returns A new instance of this component with identical parameters

evalml.pipelines.components.XGBoostClassifier.describe

XGBoostClassifier.**describe**(*print_name=False*, *return_dict=False*)

Describe a component and its parameters

Parameters

- **print_name** (*bool, optional*) – whether to print name of component
- **return_dict** (*bool, optional*) – whether to return description as dictionary in the format {"name": name, "parameters": parameters}

Returns prints and returns dictionary

Return type None or dict

evalml.pipelines.components.XGBoostClassifier.fit

XGBoostClassifier.**fit**(*X*, *y=None*)

Fits component to data

Parameters

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

Returns self

evalml.pipelines.components.XGBoostClassifier.load

static XGBoostClassifier.**load**(*file_path*)

Loads component at file path

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

Returns ComponentBase object

evalml.pipelines.components.XGBoostClassifier.predict

XGBoostClassifier.**predict**(*X*)

Make predictions using selected features.

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

Returns Predicted values

Return type pd.Series

evalml.pipelines.components.XGBoostClassifier.predict_proba

XGBoostClassifier.**predict_proba**(*X*)

Make probability estimates for labels.

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

Returns Probability estimates

Return type pd.DataFrame

evalml.pipelines.components.XGBoostClassifier.save

XGBoostClassifier.**save**(*file_path*, *pickle_protocol=4*)

Saves component at file path

Parameters

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

Returns None

evalml.pipelines.components.BaselineClassifier

class evalml.pipelines.components.**BaselineClassifier**(*strategy='mode'*, *random_state=0*, ***kwargs*)

Classifier that predicts using the specified strategy.

This is useful as a simple baseline classifier to compare with other classifiers.

name = 'Baseline Classifier'

model_family = 'baseline'

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

hyperparameter_ranges = {}

default_parameters = {'strategy': 'mode'}

Instance attributes

classes_	Returns class labels.
feature_importance	Return an attribute of instance, which is of type owner.
needs_fitting	
parameters	Returns the parameters which were used to initialize the component

Methods:

__init__	Baseline classifier that uses a simple strategy to make predictions.
clone	Constructs a new component with the same parameters
describe	Describe a component and its parameters
fit	Fits component to data
load	Loads component at file path
predict	Make predictions using selected features.
predict_proba	Make probability estimates for labels.
save	Saves component at file path

evalml.pipelines.components.BaselineClassifier.__init__

BaselineClassifier.**__init__**(*strategy='mode'*, *random_state=0*, ***kwargs*)

Baseline classifier that uses a simple strategy to make predictions.

Parameters

- **strategy** (*str*) – Method used to predict. Valid options are "mode", "random" and "random_weighted". Defaults to "mode".
- **random_state** (*int, np.random.RandomState*) – Seed for the random number generator

evalml.pipelines.components.BaselineClassifier.clone

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

Constructs a new component with the same parameters

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

Returns A new instance of this component with identical parameters

evalml.pipelines.components.BaselineClassifier.describe

BaselineClassifier.**describe**(*print_name=False*, *return_dict=False*)

Describe a component and its parameters

Parameters

- **print_name** (*bool, optional*) – whether to print name of component
- **return_dict** (*bool, optional*) – whether to return description as dictionary in the format {"name": name, "parameters": parameters}

Returns prints and returns dictionary

Return type None or dict

evalml.pipelines.components.BaselineClassifier.fit

BaselineClassifier.**fit**(*X*, *y=None*)

Fits component to data

Parameters

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

Returns self

evalml.pipelines.components.BaselineClassifier.load

static BaselineClassifier.**load**(*file_path*)

Loads component at file path

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

Returns ComponentBase object

evalml.pipelines.components.BaselineClassifier.predict

BaselineClassifier.**predict**(*X*)

Make predictions using selected features.

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

Returns Predicted values

Return type pd.Series

evalml.pipelines.components.BaselineClassifier.predict_proba

BaselineClassifier.**predict_proba**(*X*)

Make probability estimates for labels.

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

Returns Probability estimates

Return type pd.DataFrame

evalml.pipelines.components.BaselineClassifier.save

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

Saves component at file path

Parameters

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

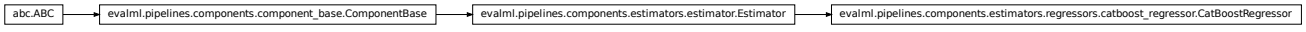
Returns None

Regressors

Regressors are components that output a predicted target value.

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

evalml.pipelines.components.CatBoostRegressor



class evalml.pipelines.components.**CatBoostRegressor**(*n_estimators=10*, *eta=0.03*, *max_depth=6*, *bootstrap_type=None*, *silent=False*, *allow_writing_files=False*, *random_state=0*, ***kwargs*)

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

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

name = 'CatBoost Regressor'

model_family = 'catboost'

supported_problem_types = [<ProblemTypes.REGRESSION: 'regression'>]

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

default_parameters = {'allow_writing_files': False, 'bootstrap_type': None, 'eta':

Instance attributes

SEED_MAX	
SEED_MIN	
feature_importance	Return an attribute of instance, which is of type owner.
needs_fitting	
parameters	Returns the parameters which were used to initialize the component

Methods:

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

evalml.pipelines.components.CatBoostRegressor.__init__

CatBoostRegressor.**__init__**(*n_estimators=10*, *eta=0.03*, *max_depth=6*, *bootstrap_type=None*, *silent=False*, *allow_writing_files=False*, *random_state=0*, ***kwargs*)

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

evalml.pipelines.components.CatBoostRegressor.clone

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

Constructs a new component with the same parameters

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

Returns A new instance of this component with identical parameters

evalml.pipelines.components.CatBoostRegressor.describe

CatBoostRegressor.**describe**(*print_name=False*, *return_dict=False*)

Describe a component and its parameters

Parameters

- **print_name** (*bool, optional*) – whether to print name of component
- **return_dict** (*bool, optional*) – whether to return description as dictionary in the format {"name": name, "parameters": parameters}

Returns prints and returns dictionary

Return type None or dict

evalml.pipelines.components.CatBoostRegressor.fit

CatBoostRegressor.**fit**(*X*, *y=None*)

Fits component to data

Parameters

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

Returns self

evalml.pipelines.components.CatBoostRegressor.load

static CatBoostRegressor.**load**(*file_path*)

Loads component at file path

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

Returns ComponentBase object

evalml.pipelines.components.CatBoostRegressor.predict

CatBoostRegressor.**predict**(*X*)

Make predictions using selected features.

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

Returns Predicted values

Return type pd.Series

evalml.pipelines.components.CatBoostRegressor.predict_proba

CatBoostRegressor.**predict_proba**(*X*)

Make probability estimates for labels.

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

Returns Probability estimates

Return type pd.DataFrame

evalml.pipelines.components.CatBoostRegressor.save

CatBoostRegressor.**save**(*file_path*, *pickle_protocol=4*)

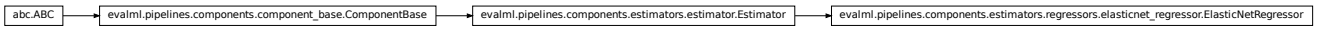
Saves component at file path

Parameters

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

Returns None

evalml.pipelines.components.ElasticNetRegressor



class evalml.pipelines.components.**ElasticNetRegressor**(*alpha=0.5*, *l1_ratio=0.5*, *max_iter=1000*, *normalize=False*, *random_state=0*, ***kwargs*)

Elastic Net Regressor.

name = 'Elastic Net Regressor'

model_family = 'linear_model'

supported_problem_types = [<ProblemTypes.REGRESSION: 'regression'>]

hyperparameter_ranges = {'alpha': Real(low=0, high=1, prior='uniform', transform='ide

default_parameters = {'alpha': 0.5, 'l1_ratio': 0.5, 'max_iter': 1000, 'normalize':

Instance attributes

feature_importance	Return an attribute of instance, which is of type owner.
needs_fitting	
parameters	Returns the parameters which were used to initialize the component

Methods:

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

evalml.pipelines.components.ElasticNetRegressor.__init__

ElasticNetRegressor.**__init__**(*alpha=0.5*, *l1_ratio=0.5*, *max_iter=1000*, *normalize=False*, *random_state=0*, ***kwargs*)

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

evalml.pipelines.components.ElasticNetRegressor.clone

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

Constructs a new component with the same parameters

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

Returns A new instance of this component with identical parameters

evalml.pipelines.components.ElasticNetRegressor.describe

ElasticNetRegressor.**describe**(*print_name=False*, *return_dict=False*)

Describe a component and its parameters

Parameters

- **print_name** (*bool, optional*) – whether to print name of component
- **return_dict** (*bool, optional*) – whether to return description as dictionary in the format {"name": name, "parameters": parameters}

Returns prints and returns dictionary

Return type None or dict

evalml.pipelines.components.ElasticNetRegressor.fit

ElasticNetRegressor.**fit**(*X*, *y=None*)

Fits component to data

Parameters

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

Returns self

evalml.pipelines.components.ElasticNetRegressor.load

static ElasticNetRegressor.**load**(*file_path*)

Loads component at file path

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

Returns ComponentBase object

evalml.pipelines.components.ElasticNetRegressor.predict

ElasticNetRegressor.**predict**(*X*)

Make predictions using selected features.

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

Returns Predicted values

Return type pd.Series

evalml.pipelines.components.ElasticNetRegressor.predict_proba

ElasticNetRegressor.**predict_proba**(*X*)
Make probability estimates for labels.

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

Returns Probability estimates

Return type pd.DataFrame

evalml.pipelines.components.ElasticNetRegressor.save

ElasticNetRegressor.**save**(*file_path*, *pickle_protocol=4*)
Saves component at file path

Parameters

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

Returns None

evalml.pipelines.components.LinearRegressor

class evalml.pipelines.components.**LinearRegressor**(*fit_intercept=True*, *normalize=False*, *n_jobs=-1*, *random_state=0*, ***kwargs*)
Linear Regressor.

name = 'Linear Regressor'

model_family = 'linear_model'

supported_problem_types = [<ProblemTypes.REGRESSION: 'regression'>]

hyperparameter_ranges = {'fit_intercept': [True, False], 'normalize': [True, False]}

default_parameters = {'fit_intercept': True, 'n_jobs': -1, 'normalize': False}

Instance attributes

feature_importance	Return an attribute of instance, which is of type owner.
needs_fitting	
parameters	Returns the parameters which were used to initialize the component

Methods:

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

evalml.pipelines.components.LinearRegressor.__init__

LinearRegressor.**__init__**(*fit_intercept=True*, *normalize=False*, *n_jobs=-1*, *random_state=0*, ***kwargs*)

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

evalml.pipelines.components.LinearRegressor.clone

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

Constructs a new component with the same parameters

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

Returns A new instance of this component with identical parameters

evalml.pipelines.components.LinearRegressor.describe

LinearRegressor.**describe**(*print_name=False*, *return_dict=False*)

Describe a component and its parameters

Parameters

- **print_name** (*bool, optional*) – whether to print name of component
- **return_dict** (*bool, optional*) – whether to return description as dictionary in the format {"name": name, "parameters": parameters}

Returns prints and returns dictionary

Return type None or dict

evalml.pipelines.components.LinearRegressor.fit

LinearRegressor.**fit**(*X*, *y=None*)

Fits component to data

Parameters

- **X** (*pd.DataFrame or np.array*) – the input training data of shape [n_samples, n_features]

- **y** (*pd.Series, optional*) – the target training data of length [n_samples]

Returns self

evalml.pipelines.components.LinearRegressor.load

static LinearRegressor.**load**(*file_path*)

Loads component at file path

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

Returns ComponentBase object

evalml.pipelines.components.LinearRegressor.predict

LinearRegressor.**predict**(*X*)

Make predictions using selected features.

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

Returns Predicted values

Return type pd.Series

evalml.pipelines.components.LinearRegressor.predict_proba

LinearRegressor.**predict_proba**(*X*)

Make probability estimates for labels.

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

Returns Probability estimates

Return type pd.DataFrame

evalml.pipelines.components.LinearRegressor.save

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

Saves component at file path

Parameters

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

Returns None

evalml.pipelines.components.ExtraTreesRegressor

class evalml.pipelines.components.**ExtraTreesRegressor**(*n_estimators=100*, *max_features='auto'*, *max_depth=6*, *min_samples_split=2*, *min_weight_fraction_leaf=0.0*, *n_jobs=-1*, *random_state=0*, ***kwargs*)

Extra Trees Regressor.

name = 'Extra Trees Regressor'

model_family = 'extra_trees'

supported_problem_types = [<ProblemTypes.REGRESSION: 'regression'>]

hyperparameter_ranges = {'max_depth': Integer(low=4, high=10, prior='uniform', transf

default_parameters = {'max_depth': 6, 'max_features': 'auto', 'min_samples_split':

Instance attributes

feature_importance	Return an attribute of instance, which is of type owner.
needs_fitting	
parameters	Returns the parameters which were used to initialize the component

Methods:

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

evalml.pipelines.components.ExtraTreesRegressor.__init__

ExtraTreesRegressor.**__init__**(*n_estimators=100*, *max_features='auto'*, *max_depth=6*, *min_samples_split=2*, *min_weight_fraction_leaf=0.0*, *n_jobs=-1*, *random_state=0*, ***kwargs*)

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

evalml.pipelines.components.ExtraTreesRegressor.clone

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

Constructs a new component with the same parameters

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

Returns A new instance of this component with identical parameters

evalml.pipelines.components.ExtraTreesRegressor.describe

ExtraTreesRegressor.**describe**(*print_name=False*, *return_dict=False*)

Describe a component and its parameters

Parameters

- **print_name** (*bool, optional*) – whether to print name of component
- **return_dict** (*bool, optional*) – whether to return description as dictionary in the format {"name": name, "parameters": parameters}

Returns prints and returns dictionary

Return type None or dict

evalml.pipelines.components.ExtraTreesRegressor.fit

ExtraTreesRegressor.**fit**(*X*, *y=None*)

Fits component to data

Parameters

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

Returns self

evalml.pipelines.components.ExtraTreesRegressor.load

static ExtraTreesRegressor.**load**(*file_path*)

Loads component at file path

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

Returns ComponentBase object

evalml.pipelines.components.ExtraTreesRegressor.predict

ExtraTreesRegressor.**predict**(*X*)

Make predictions using selected features.

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

Returns Predicted values

Return type pd.Series

evalml.pipelines.components.ExtraTreesRegressor.predict_proba

ExtraTreesRegressor.**predict_proba**(*X*)

Make probability estimates for labels.

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

Returns Probability estimates

Return type pd.DataFrame

evalml.pipelines.components.ExtraTreesRegressor.save

ExtraTreesRegressor.**save**(*file_path*, *pickle_protocol=4*)

Saves component at file path

Parameters

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

Returns None

evalml.pipelines.components.RandomForestRegressor

class evalml.pipelines.components.**RandomForestRegressor**(*n_estimators=100*, *max_depth=6*, *n_jobs=-1*, *random_state=0*, ***kwargs*)

Random Forest Regressor.

name = 'Random Forest Regressor'

model_family = 'random_forest'

supported_problem_types = [<ProblemTypes.REGRESSION: 'regression'>]

hyperparameter_ranges = {'max_depth': Integer(low=1, high=32, prior='uniform', transf

default_parameters = {'max_depth': 6, 'n_estimators': 100, 'n_jobs': -1}

Instance attributes

feature_importance	Return an attribute of instance, which is of type owner.
needs_fitting	
parameters	Returns the parameters which were used to initialize the component

Methods:

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

evalml.pipelines.components.RandomForestRegressor.__init__

RandomForestRegressor.**__init__**(*n_estimators=100*, *max_depth=6*, *n_jobs=-1*, *random_state=0*, ***kwargs*)

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

evalml.pipelines.components.RandomForestRegressor.clone

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

Constructs a new component with the same parameters

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

Returns A new instance of this component with identical parameters

evalml.pipelines.components.RandomForestRegressor.describe

RandomForestRegressor.**describe**(*print_name=False*, *return_dict=False*)

Describe a component and its parameters

Parameters

- **print_name** (*bool, optional*) – whether to print name of component
- **return_dict** (*bool, optional*) – whether to return description as dictionary in the format {"name": name, "parameters": parameters}

Returns prints and returns dictionary

Return type None or dict

evalml.pipelines.components.RandomForestRegressor.fit

RandomForestRegressor.**fit**(*X*, *y=None*)

Fits component to data

Parameters

- **X** (*pd.DataFrame or np.array*) – the input training data of shape [n_samples, n_features]

- **y** (*pd.Series, optional*) – the target training data of length [n_samples]

Returns self

evalml.pipelines.components.RandomForestRegressor.load

static RandomForestRegressor.**load**(*file_path*)

Loads component at file path

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

Returns ComponentBase object

evalml.pipelines.components.RandomForestRegressor.predict

RandomForestRegressor.**predict**(*X*)

Make predictions using selected features.

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

Returns Predicted values

Return type pd.Series

evalml.pipelines.components.RandomForestRegressor.predict_proba

RandomForestRegressor.**predict_proba**(*X*)

Make probability estimates for labels.

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

Returns Probability estimates

Return type pd.DataFrame

evalml.pipelines.components.RandomForestRegressor.save

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

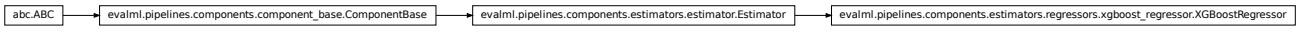
Saves component at file path

Parameters

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

Returns None

evalml.pipelines.components.XGBoostRegressor



class evalml.pipelines.components.**XGBoostRegressor**(*eta=0.1*, *max_depth=6*, *min_child_weight=1*, *n_estimators=100*, *random_state=0*, ***kwargs*)

XGBoost Regressor.

name = 'XGBoost Regressor'

model_family = 'xgboost'

supported_problem_types = [<ProblemTypes.REGRESSION: 'regression'>]

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

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

Instance attributes

SEED_MAX	
SEED_MIN	
feature_importance	Return an attribute of instance, which is of type owner.
needs_fitting	
parameters	Returns the parameters which were used to initialize the component

Methods:

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

evalml.pipelines.components.XGBoostRegressor.__init__

XGBoostRegressor.**__init__**(*eta=0.1*, *max_depth=6*, *min_child_weight=1*, *n_estimators=100*, *random_state=0*, ***kwargs*)

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

evalml.pipelines.components.XGBoostRegressor.clone

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

Constructs a new component with the same parameters

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

Returns A new instance of this component with identical parameters

evalml.pipelines.components.XGBoostRegressor.describe

XGBoostRegressor.**describe**(*print_name=False*, *return_dict=False*)

Describe a component and its parameters

Parameters

- **print_name** (*bool, optional*) – whether to print name of component
- **return_dict** (*bool, optional*) – whether to return description as dictionary in the format {"name": name, "parameters": parameters}

Returns prints and returns dictionary

Return type None or dict

evalml.pipelines.components.XGBoostRegressor.fit

XGBoostRegressor.**fit**(*X*, *y=None*)

Fits component to data

Parameters

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

Returns self

evalml.pipelines.components.XGBoostRegressor.load

static XGBoostRegressor.**load**(*file_path*)

Loads component at file path

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

Returns ComponentBase object

evalml.pipelines.components.XGBoostRegressor.predict

XGBoostRegressor.**predict**(*X*)

Make predictions using selected features.

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

Returns Predicted values

Return type pd.Series

evalml.pipelines.components.XGBoostRegressor.predict_proba

XGBoostRegressor.**predict_proba**(*X*)

Make probability estimates for labels.

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

Returns Probability estimates

Return type pd.DataFrame

evalml.pipelines.components.XGBoostRegressor.save

XGBoostRegressor.**save**(*file_path*, *pickle_protocol=4*)

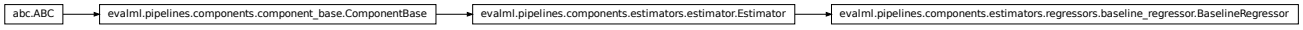
Saves component at file path

Parameters

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

Returns None

evalml.pipelines.components.BaselineRegressor



class evalml.pipelines.components.**BaselineRegressor**(*strategy='mean'*, *random_state=0*, ***kwargs*)

Regressor that predicts using the specified strategy.

This is useful as a simple baseline regressor to compare with other regressors.

name = 'Baseline Regressor'

model_family = 'baseline'

supported_problem_types = [<ProblemTypes.REGRESSION: 'regression'>]

hyperparameter_ranges = {}

default_parameters = {'strategy': 'mean'}

Instance attributes

feature_importance	Return an attribute of instance, which is of type owner.
needs_fitting	
parameters	Returns the parameters which were used to initialize the component

Methods:

__init__	Baseline regressor that uses a simple strategy to make predictions.
clone	Constructs a new component with the same parameters
describe	Describe a component and its parameters

Continued on next page

Table 99 – continued from previous page

fit	Fits component to data
load	Loads component at file path
predict	Make predictions using selected features.
predict_proba	Make probability estimates for labels.
save	Saves component at file path

evalml.pipelines.components.BaselineRegressor.__init__

BaselineRegressor.**__init__**(*strategy='mean'*, *random_state=0*, ***kwargs*)

Baseline regressor that uses a simple strategy to make predictions.

Parameters

- **strategy** (*str*) – method used to predict. Valid options are "mean", "median". Defaults to "mean".
- **random_state** (*int, np.random.RandomState*) – seed for the random number generator

evalml.pipelines.components.BaselineRegressor.clone

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

Constructs a new component with the same parameters

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

Returns A new instance of this component with identical parameters

evalml.pipelines.components.BaselineRegressor.describe

BaselineRegressor.**describe**(*print_name=False*, *return_dict=False*)

Describe a component and its parameters

Parameters

- **print_name** (*bool, optional*) – whether to print name of component
- **return_dict** (*bool, optional*) – whether to return description as dictionary in the format {"name": name, "parameters": parameters}

Returns prints and returns dictionary

Return type None or dict

evalml.pipelines.components.BaselineRegressor.fit

BaselineRegressor.**fit**(*X*, *y=None*)

Fits component to data

Parameters

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

Returns self

evalml.pipelines.components.BaselineRegressor.load

static BaselineRegressor.**load**(*file_path*)
Loads component at file path

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

Returns ComponentBase object

evalml.pipelines.components.BaselineRegressor.predict

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

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

Returns Predicted values

Return type pd.Series

evalml.pipelines.components.BaselineRegressor.predict_proba

BaselineRegressor.**predict_proba**(*X*)
Make probability estimates for labels.

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

Returns Probability estimates

Return type pd.DataFrame

evalml.pipelines.components.BaselineRegressor.save

BaselineRegressor.**save**(*file_path*, *pickle_protocol=4*)
Saves component at file path

Parameters

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

Returns None

5.6 Model Understanding

5.6.1 Graph Utils

confusion_matrix	Confusion matrix for binary and multiclass classification.

Continued on next page

Table 100 – continued from previous page

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

evalml.model_understanding.confusion_matrix

evalml.model_understanding.**confusion_matrix**(*y_true*, *y_predicted*, *normalize_method='true'*)

Confusion matrix for binary and multiclass classification.

Parameters

- **y_true** (*pd.Series or np.array*) – True binary labels.
- **y_pred** (*pd.Series or np.array*) – Predictions from a binary classifier.
- **normalize_method** (*{'true', 'pred', 'all'}*) – Normalization method. Supported options are: 'true' to normalize by row, 'pred' to normalize by column, or 'all' to normalize by all values. Defaults to 'true'.

Returns Confusion matrix. The column header represents the predicted labels while row header represents the actual labels.

Return type pd.DataFrame

evalml.model_understanding.normalize_confusion_matrix

evalml.model_understanding.**normalize_confusion_matrix**(*conf_mat*, *normalize_method='true'*)

Normalizes a confusion matrix.

Parameters

- **conf_mat** (*pd.DataFrame or np.array*) – Confusion matrix to normalize.
- **normalize_method** (*{'true', 'pred', 'all'}*) – Normalization method. Supported options are: 'true' to normalize by row, 'pred' to normalize by column, or 'all' to normalize by all values. Defaults to 'true'.

Returns normalized version of the input confusion matrix. The column header represents the predicted labels while row header represents the actual labels.

Return type pd.DataFrame

evalml.model_understanding.precision_recall_curve

evalml.model_understanding.**precision_recall_curve**(*y_true*, *y_pred_proba*)

Given labels and binary classifier predicted probabilities, compute and return the data representing a precision-recall curve.

Parameters

- **y_true** (*pd.Series or np.array*) – True binary labels.
- **y_pred_proba** (*pd.Series or np.array*) – Predictions from a binary classifier, before thresholding has been applied. Note this should be the predicted probability for the "true" label.

Returns

Dictionary containing metrics used to generate a precision-recall plot, with the following keys:

- *precision*: Precision values.
- *recall*: Recall values.
- *thresholds*: Threshold values used to produce the precision and recall.
- *auc_score*: The area under the ROC curve.

Return type list

evalml.model_understanding.graph_precision_recall_curve

evalml.model_understanding.**graph_precision_recall_curve**(*y_true*, *y_pred_proba*, *title_addition=None*)

Generate and display a precision-recall plot.

Parameters

- **y_true** (*pd.Series or np.array*) – True binary labels.
- **y_pred_proba** (*pd.Series or np.array*) – Predictions from a binary classifier, before thresholding has been applied. Note this should be the predicted probability for the "true" label.
- **title_addition** (*str or None*) – If not None, append to plot title. Default None.

Returns plotly.Figure representing the precision-recall plot generated

evalml.model_understanding.roc_curve

evalml.model_understanding.**roc_curve**(*y_true*, *y_pred_proba*)

Given labels and classifier predicted probabilities, compute and return the data representing a Receiver Operating Characteristic (ROC) curve. Works with binary or multiclass problems.

Parameters

- **y_true** (*pd.Series or np.array*) – True labels.
- **y_pred_proba** (*pd.Series or np.array*) – Predictions from a classifier, before thresholding has been applied.

Returns

A list of dictionaries (with one for each class) is returned. Binary classification problems return a list with one di

Each dictionary contains metrics used to generate an ROC plot with the following keys:

- *fpr_rate*: False positive rate.
- *tpr_rate*: True positive rate.
- *threshold*: Threshold values used to produce each pair of true/false positive rates.
- *auc_score*: The area under the ROC curve.

Return type list(dict)

evalml.model_understanding.graph_roc_curve

evalml.model_understanding.**graph_roc_curve**(*y_true*, *y_pred_proba*, *custom_class_names=None*, *title_addition=None*)

Generate and display a Receiver Operating Characteristic (ROC) plot for binary and multiclass classification problems.

Parameters

- **y_true** (*pd.Series or np.array*) – True labels.
- **y_pred_proba** (*pd.Series or np.array*) – Predictions from a classifier, before thresholding has been applied. Note this should a one dimensional array with the predicted probability for the "true" label in the binary case.
- **custom_class_labels** (*list or None*) – If not None, custom labels for classes. Default None.
- **title_addition** (*str or None*) – if not None, append to plot title. Default None.

Returns plotly.Figure representing the ROC plot generated

evalml.model_understanding.graph_confusion_matrix

evalml.model_understanding.**graph_confusion_matrix**(*y_true*, *y_pred*, *normalize_method='true'*, *title_addition=None*)

Generate and display a confusion matrix plot.

If *normalize_method* is set, hover text will show raw count, otherwise hover text will show count normalized with method 'true'.

Parameters

- **y_true** (*pd.Series or np.array*) – True binary labels.
- **y_pred** (*pd.Series or np.array*) – Predictions from a binary classifier.
- **normalize_method** (*{'true', 'pred', 'all'}*) – Normalization method. Supported options are: 'true' to normalize by row, 'pred' to normalize by column, or 'all' to normalize by all values. Defaults to 'true'.
- **title_addition** (*str or None*) – if not None, append to plot title. Default None.

Returns plotly.Figure representing the confusion matrix plot generated

evalml.model_understanding.calculate_permutation_importance

evalml.model_understanding.**calculate_permutation_importance**(*pipeline*, *X*, *y*, *objective*, *n_repeats=5*, *n_jobs=None*, *random_state=0*)

Calculates permutation importance for features.

Parameters

- **pipeline** (*PipelineBase or subclass*) – Fitted pipeline
- **X** (*pd.DataFrame*) – The input data used to score and compute permutation importance
- **y** (*pd.Series*) – The target data
- **objective** (*str, ObjectiveBase*) – Objective to score on
- **n_repeats** (*int*) – Number of times to permute a feature. Defaults to 5.
- **n_jobs** (*int or None*) – Non-negative integer describing level of parallelism used for pipelines. None and 1 are equivalent. If set to -1, all CPUs are used. For n_jobs below -1, (n_cpus + 1 + n_jobs) are used.
- **random_state** (*int, np.random.RandomState*) – The random seed/state. Defaults to 0.

Returns Mean feature importance scores over 5 shuffles.

evalml.model_understanding.graph_permutation_importance

evalml.model_understanding.**graph_permutation_importance**(*pipeline*, *X*, *y*, *objective*, *importance_threshold=0*)

Generate a bar graph of the pipeline's permutation importance.

Parameters

- **pipeline** (*PipelineBase or subclass*) – Fitted pipeline
- **X** (*pd.DataFrame*) – The input data used to score and compute permutation importance
- **y** (*pd.Series*) – The target data
- **objective** (*str, ObjectiveBase*) – Objective to score on
- **importance_threshold** (*float, optional*) – If provided, graph features with a permutation importance whose absolute value is larger than importance_threshold. Defaults to zero.

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

evalml.model_understanding.binary_objective_vs_threshold

evalml.model_understanding.**binary_objective_vs_threshold**(*pipeline*, *X*, *y*, *objective*, *steps=100*)

Computes objective score as a function of potential binary classification decision thresholds for a fitted binary classification pipeline.

Parameters

- **pipeline** (*BinaryClassificationPipeline obj*) – Fitted binary classification pipeline
- **X** (*pd.DataFrame*) – The input data used to compute objective score
- **y** (*pd.Series*) – The target labels
- **objective** (*ObjectiveBase obj, str*) – Objective used to score
- **steps** (*int*) – Number of intervals to divide and calculate objective score at

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

Return type pd.DataFrame

evalml.model_understanding.graph_binary_objective_vs_threshold

evalml.model_understanding.**graph_binary_objective_vs_threshold**(*pipeline*, *X*, *y*, *objective*, *steps=100*)

Generates a plot graphing objective score vs. decision thresholds for a fitted binary classification pipeline.

Parameters

- **pipeline** (*PipelineBase or subclass*) – Fitted pipeline
- **X** (*pd.DataFrame*) – The input data used to score and compute scores
- **y** (*pd.Series*) – The target labels
- **objective** (*ObjectiveBase obj, str*) – Objective used to score, shown on the y-axis of the graph
- **steps** (*int*) – Number of intervals to divide and calculate objective score at

Returns plotly.Figure representing the objective score vs. threshold graph generated

5.6.2 Prediction Explanations

explain_prediction	Creates table summarizing the top_k positive and top_k negative contributing features to the prediction of a single datapoint.
explain_predictions	Creates a report summarizing the top contributing features for each data point in the input features.
explain_predictions_best_worst	Creates a report summarizing the top contributing features for the best and worst points in the dataset as measured by error to true labels.

evalml.model_understanding.prediction_explanations.explain_prediction

evalml.model_understanding.prediction_explanations.**explain_prediction**(*pipeline*, *input_features*, *top_k=3*, *training_data=None*, *include_shap_values=False*, *output_format='text'*)

Creates table summarizing the top_k positive and top_k negative contributing features to the prediction of a single datapoint.

XGBoost models and CatBoost multiclass classifiers are not currently supported.

Parameters

- **pipeline** (PipelineBase) – Fitted pipeline whose predictions we want to explain with SHAP.
- **input_features** (*pd.DataFrame*) – Dataframe of features - needs to correspond to data the pipeline was fit on.
- **top_k** (*int*) – How many of the highest/lowest features to include in the table.
- **training_data** (*pd.DataFrame*) – Training data the pipeline was fit on. This is required for non-tree estimators because we need a sample of training data for the KernelSHAP algorithm.
- **include_shap_values** (*bool*) – Whether the SHAP values should be included in an extra column in the output. Default is False.
- **output_format** (*str*) – Either "text" or "dict". Default is "text".

Returns str or dict - A report explaining the most positive/negative contributing features to the predictions.

evalml.model_understanding.prediction_explanations.explain_predictions

evalml.model_understanding.prediction_explanations.**explain_predictions**(*pipeline*, *input_features*, *training_data=None*, *top_k_features=3*, *include_shap_values=False*, *output_format='text'*)

Creates a report summarizing the top contributing features for each data point in the input features.

XGBoost models and CatBoost multiclass classifiers are not currently supported.

Parameters

- **pipeline** (PipelineBase) – Fitted pipeline whose predictions we want to explain with SHAP.

- **input_features** (*pd.DataFrame*) – Dataframe of input data to evaluate the pipeline on.
- **training_data** (*pd.DataFrame*) – Dataframe of data the pipeline was fit on. This can be omitted for pipelines with tree-based estimators.
- **top_k_features** (*int*) – How many of the highest/lowest contributing feature to include in the table for each data point.
- **include_shap_values** (*bool*) – Whether SHAP values should be included in the table. Default is False.
- **output_format** (*str*) – Either "text" or "dict". Default is "text".

Returns

str or dict - A report explaining the top contributing features to each prediction for each row of input_features. The report will include the feature names, prediction contribution, and SHAP Value (optional).

evalml.model_understanding.prediction_explanations.explain_predictions_best_worst

evalml.model_understanding.prediction_explanations.**explain_predictions_best_worst**(*pipeline*, *input_features*, *y_true*, *num_to_expl*, *top_k_featur*, *include_shap_*, *metric=None*, *output_format=*

Creates a report summarizing the top contributing features for the best and worst points in the dataset as measured by error to true labels.

XGBoost models and CatBoost multiclass classifiers are not currently supported.

Parameters

- **pipeline** (PipelineBase) – Fitted pipeline whose predictions we want to explain with SHAP.
- **input_features** (*pd.DataFrame*) – Dataframe of input data to evaluate the pipeline on.
- **y_true** (*pd.Series*) – True labels for the input data.
- **num_to_explain** (*int*) – How many of the best, worst, random data points to explain.
- **top_k_features** (*int*) – How many of the highest/lowest contributing feature to include in the table for each data point.
- **include_shap_values** (*bool*) – Whether SHAP values should be included in the table. Default is False.
- **metric** (*callable*) – The metric used to identify the best and worst points in the dataset. Function must accept the true labels and predicted value or probabilities as the only arguments and lower values must be better. By default, this will be the absolute error for regression problems and cross entropy loss for classification problems.

- **output_format** (*str*) – Either "text" or "dict". Default is "text".

Returns

str or dict - A report explaining the top contributing features for the best/worst predictions in the input_features
For each of the best/worst rows of input_features, the predicted values, true labels, metric value, feature names, prediction contribution, and SHAP Value (optional) will be listed.

5.7 Objective Functions

5.7.1 Objective Base Classes

ObjectiveBase	Base class for all objectives.
BinaryClassificationObjective	Base class for all binary classification objectives.
MulticlassClassificationObjective	Base class for all multiclass classification objectives.
RegressionObjective	Base class for all regression objectives.

evalml.objectives.ObjectiveBase

class evalml.objectives.**ObjectiveBase**
Base class for all objectives.

Methods

calculate_percent_difference	Calculate the percent difference between scores.
objective_function	Computes the relative value of the provided predictions compared to the actual labels, according a specified metric
score	Returns a numerical score indicating performance based on the differences between the predicted and actual values.
validate_inputs	Validates the input based on a few simple checks.

evalml.objectives.ObjectiveBase.calculate_percent_difference

classmethod ObjectiveBase.**calculate_percent_difference**(*score*, *baseline_score*)
Calculate the percent difference between scores.

Parameters

- **score** (*float*) – A score. Output of the score method of this objective.
- **baseline_score** (*float*) – A score. Output of the score method of this objective. In practice, this is the score achieved on this objective with a baseline estimator.

Returns

The percent difference between the scores. This will be the difference normalized by the baseline score.

Return type float

evalml.objectives.ObjectiveBase.objective_function

classmethod ObjectiveBase.**objective_function**(*y_true*, *y_predicted*, *X=None*)

Computes the relative value of the provided predictions compared to the actual labels, according a specified metric

Arguments: y_predicted (pd.Series): Predicted values of length [n_samples] y_true (pd.Series): Actual class labels of length [n_samples] X (pd.DataFrame or np.array): Extra data of shape [n_samples, n_features] necessary to calculate score

Returns Numerical value used to calculate score

evalml.objectives.ObjectiveBase.score

ObjectiveBase.**score**(*y_true*, *y_predicted*, *X=None*)

Returns a numerical score indicating performance based on the differences between the predicted and actual values.

Parameters

- **y_predicted** (*pd.Series*) – Predicted values of length [n_samples]
- **y_true** (*pd.Series*) – Actual class labels of length [n_samples]
- **X** (*pd.DataFrame or np.array*) – Extra data of shape [n_samples, n_features] necessary to calculate score

Returns score

evalml.objectives.ObjectiveBase.validate_inputs

ObjectiveBase.**validate_inputs**(*y_true*, *y_predicted*)

Validates the input based on a few simple checks.

Parameters

- **y_predicted** (*pd.Series*) – Predicted values of length [n_samples]
- **y_true** (*pd.Series*) – Actual class labels of length [n_samples]

Returns None

evalml.objectives.BinaryClassificationObjective

class evalml.objectives.**BinaryClassificationObjective**

Base class for all binary classification objectives.

problem_type (ProblemTypes): Type of problem this objective is. Set to ProblemTypes.BINARY. can_optimize_threshold (bool): Determines if threshold used by objective can be optimized or not.

Methods

calculate_percent_difference	Calculate the percent difference between scores.
decision_function	Apply a learned threshold to predicted probabilities to get predicted classes.
objective_function	Computes the relative value of the provided predictions compared to the actual labels, according a specified metric
optimize_threshold	Learn a binary classification threshold which optimizes the current objective.
score	Returns a numerical score indicating performance based on the differences between the predicted and actual values.
validate_inputs	Validates the input based on a few simple checks.

evalml.objectives.BinaryClassificationObjective.calculate_percent_difference

classmethod BinaryClassificationObjective.**calculate_percent_difference**(*score*, *baseline_score*)

Calculate the percent difference between scores.

Parameters

- **score** (*float*) – A score. Output of the score method of this objective.
- **baseline_score** (*float*) – A score. Output of the score method of this objective. In practice, this is the score achieved on this objective with a baseline estimator.

Returns

The percent difference between the scores. This will be the difference normalized by the baseline score.

Return type float

evalml.objectives.BinaryClassificationObjective.decision_function

BinaryClassificationObjective.**decision_function**(*ypred_proba*, *threshold=0.5*, *X=None*)

Apply a learned threshold to predicted probabilities to get predicted classes.

Parameters

- **ypred_proba** (*list*) – The classifier's predicted probabilities
- **threshold** (*float, optional*) – Threshold used to make a prediction. Defaults to 0.5.
- **X** (*pd.DataFrame, optional*) – Any extra columns that are needed from training data.

Returns predictions

evalml.objectives.BinaryClassificationObjective.objective_function

classmethod BinaryClassificationObjective.**objective_function**(*y_true*, *y_predicted*, *X=None*)

Computes the relative value of the provided predictions compared to the actual labels, according a specified metric

Arguments: y_predicted (pd.Series): Predicted values of length [n_samples] y_true (pd.Series): Actual class labels of length [n_samples] X (pd.DataFrame or np.array): Extra data of shape [n_samples, n_features] necessary to calculate score

Returns Numerical value used to calculate score

evalml.objectives.BinaryClassificationObjective.optimize_threshold

BinaryClassificationObjective.**optimize_threshold**(*ypred_proba*, *y_true*, *X=None*)

Learn a binary classification threshold which optimizes the current objective.

Parameters

- **ypred_proba** (*list*) – The classifier's predicted probabilities
- **y_true** (*list*) – The ground truth for the predictions.
- **X** (*pd.DataFrame, optional*) – Any extra columns that are needed from training data.

Returns Optimal threshold for this objective

evalml.objectives.BinaryClassificationObjective.score

BinaryClassificationObjective.**score**(*y_true*, *y_predicted*, *X=None*)

Returns a numerical score indicating performance based on the differences between the predicted and actual values.

Parameters

- **y_predicted** (*pd.Series*) – Predicted values of length [n_samples]

- **y_true** (*pd.Series*) – Actual class labels of length [n_samples]
- **X** (*pd.DataFrame or np.array*) – Extra data of shape [n_samples, n_features] necessary to calculate score

Returns score

evalml.objectives.BinaryClassificationObjective.validate_inputs

BinaryClassificationObjective.**validate_inputs**(*y_true*, *y_predicted*)

Validates the input based on a few simple checks.

Parameters

- **y_predicted** (*pd.Series*) – Predicted values of length [n_samples]
- **y_true** (*pd.Series*) – Actual class labels of length [n_samples]

Returns None

evalml.objectives.MulticlassClassificationObjective

class evalml.objectives.**MulticlassClassificationObjective**

Base class for all multiclass classification objectives.

problem_type (ProblemTypes): Type of problem this objective is. Set to ProblemTypes.MULTICLASS.

Methods

calculate_percent_difference	Calculate the percent difference between scores.
objective_function	Computes the relative value of the provided predictions compared to the actual labels, according a specified metric
score	Returns a numerical score indicating performance based on the differences between the predicted and actual values.
validate_inputs	Validates the input based on a few simple checks.

evalml.objectives.MulticlassClassificationObjective.calculate_percent_difference

classmethod MulticlassClassificationObjective.**calculate_percent_difference**(*score*, *baseline_score*)

Calculate the percent difference between scores.

Parameters

- **score** (*float*) – A score. Output of the score method of this objective.

- **baseline_score** (*float*) – A score. Output of the score method of this objective. In practice, this is the score achieved on this objective with a baseline estimator.

Returns

The percent difference between the scores. This will be the difference normalized by the baseline score.

Return type float

evalml.objectives.MulticlassClassificationObjective.objective_function

classmethod MulticlassClassificationObjective.**objective_function**(*y_true*, *y_predicted*, *X=None*)

Computes the relative value of the provided predictions compared to the actual labels, according a specified metric

Arguments: y_predicted (pd.Series): Predicted values of length [n_samples] y_true (pd.Series): Actual class labels of length [n_samples] X (pd.DataFrame or np.array): Extra data of shape [n_samples, n_features] necessary to calculate score

Returns Numerical value used to calculate score

evalml.objectives.MulticlassClassificationObjective.score

MulticlassClassificationObjective.**score**(*y_true*, *y_predicted*, *X=None*)

Returns a numerical score indicating performance based on the differences between the predicted and actual values.

Parameters

- **y_predicted** (*pd.Series*) – Predicted values of length [n_samples]
- **y_true** (*pd.Series*) – Actual class labels of length [n_samples]
- **X** (*pd.DataFrame or np.array*) – Extra data of shape [n_samples, n_features] necessary to calculate score

Returns score

evalml.objectives.MulticlassClassificationObjective.validate_inputs

MulticlassClassificationObjective.**validate_inputs**(*y_true*, *y_predicted*)

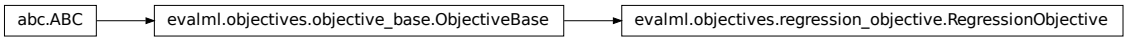
Validates the input based on a few simple checks.

Parameters

- **y_predicted** (*pd.Series*) – Predicted values of length [n_samples]
- **y_true** (*pd.Series*) – Actual class labels of length [n_samples]

Returns None

evalml.objectives.RegressionObjective



class evalml.objectives.**RegressionObjective**

Base class for all regression objectives.

problem_type (ProblemTypes): Type of problem this objective is. Set to ProblemTypes.REGRESSION.

Methods

calculate_percent_difference	Calculate the percent difference between scores.
objective_function	Computes the relative value of the provided predictions compared to the actual labels, according a specified metric
score	Returns a numerical score indicating performance based on the differences between the predicted and actual values.
validate_inputs	Validates the input based on a few simple checks.

evalml.objectives.RegressionObjective.calculate_percent_difference

classmethod RegressionObjective.**calculate_percent_difference**(*score*, *baseline_score*)

Calculate the percent difference between scores.

Parameters

- **score** (*float*) – A score. Output of the score method of this objective.
- **baseline_score** (*float*) – A score. Output of the score method of this objective. In practice, this is the score achieved on this objective with a baseline estimator.

Returns

The percent difference between the scores. This will be the difference normalized by the baseline score.

Return type float

evalml.objectives.RegressionObjective.objective_function

classmethod RegressionObjective.**objective_function**(*y_true*, *y_predicted*, *X=None*)

Computes the relative value of the provided predictions compared to the actual labels, according a specified metric

Arguments: y_predicted (pd.Series): Predicted values of length [n_samples] y_true (pd.Series): Actual class labels of length [n_samples] X (pd.DataFrame or np.array): Extra data of shape [n_samples, n_features] necessary to calculate score

Returns Numerical value used to calculate score

evalml.objectives.RegressionObjective.score

RegressionObjective.**score**(*y_true*, *y_predicted*, *X=None*)

Returns a numerical score indicating performance based on the differences between the predicted and actual values.

Parameters

- **y_predicted** (*pd.Series*) – Predicted values of length [n_samples]
- **y_true** (*pd.Series*) – Actual class labels of length [n_samples]
- **X** (*pd.DataFrame or np.array*) – Extra data of shape [n_samples, n_features] necessary to calculate score

Returns score

evalml.objectives.RegressionObjective.validate_inputs

RegressionObjective.**validate_inputs**(*y_true*, *y_predicted*)

Validates the input based on a few simple checks.

Parameters

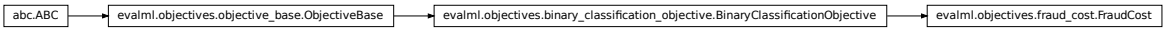
- **y_predicted** (*pd.Series*) – Predicted values of length [n_samples]
- **y_true** (*pd.Series*) – Actual class labels of length [n_samples]

Returns None

5.7.2 Domain-Specific Objectives

FraudCost	Score the percentage of money lost of the total transaction amount process due to fraud.
LeadScoring	Lead scoring.
CostBenefitMatrix	Score using a cost-benefit matrix.

evalml.objectives.FraudCost



class evalml.objectives.**FraudCost**(*retry_percentage=0.5*, *interchange_fee=0.02*, *fraud_payout_percentage=1.0*, *amount_col='amount'*)

Score the percentage of money lost of the total transaction amount process due to fraud.

Methods

__init__	Create instance of FraudCost
calculate_percent_difference	Calculate the percent difference between scores.
decision_function	Determine if a transaction is fraud given predicted probabilities, threshold, and dataframe with transaction amount.
objective_function	Calculate amount lost to fraud per transaction given predictions, true values, and dataframe with transaction amount.
optimize_threshold	Learn a binary classification threshold which optimizes the current objective.
score	Returns a numerical score indicating performance based on the differences between the predicted and actual values.
validate_inputs	Validates the input based on a few simple checks.

evalml.objectives.FraudCost.__init__

FraudCost.**__init__**(*retry_percentage=0.5*, *interchange_fee=0.02*, *fraud_payout_percentage=1.0*, *amount_col='amount'*)

Create instance of FraudCost

Parameters

- **retry_percentage** (*float*) – What percentage of customers that will retry a transaction if it is declined. Between 0 and 1. Defaults to .5
- **interchange_fee** (*float*) – How much of each successful transaction you can collect. Between 0 and 1. Defaults to .02
- **fraud_payout_percentage** (*float*) – Percentage of fraud you will not be able to collect. Between 0 and 1. Defaults to 1.0
- **amount_col** (*str*) – Name of column in data that contains the amount. Defaults to "amount"

evalml.objectives.FraudCost.calculate_percent_difference

classmethod FraudCost.**calculate_percent_difference**(*score*, *baseline_score*)

Calculate the percent difference between scores.

Parameters

- **score** (*float*) – A score. Output of the score method of this objective.
- **baseline_score** (*float*) – A score. Output of the score method of this objective. In practice, this is the score achieved on this objective with a baseline estimator.

Returns

The percent difference between the scores. This will be the difference normalized by the baseline score.

Return type float

evalml.objectives.FraudCost.decision_function

FraudCost.**decision_function**(*ypred_proba*, *threshold=0.0*, *X=None*)

Determine if a transaction is fraud given predicted probabilities, threshold, and dataframe with transaction amount.

Parameters

- **ypred_proba** (*pd.Series*) – Predicted probablities
- **X** (*pd.DataFrame*) – Dataframe containing transaction amount
- **threshold** (*float*) – Dollar threshold to determine if transaction is fraud

Returns pd.Series of predicted fraud labels using X and threshold

Return type pd.Series

evalml.objectives.FraudCost.objective_function

FraudCost.**objective_function**(*y_true*, *y_predicted*, *X*)

Calculate amount lost to fraud per transaction given predictions, true values, and dataframe with transaction amount.

Parameters

- **y_predicted** (*pd.Series*) – Predicted fraud labels y_true (pd.Series): True fraud labels
- **X** (*pd.DataFrame*) – pd.Dataframe with transaction amounts

Returns Amount lost to fraud per transaction

Return type float

evalml.objectives.FraudCost.optimize_threshold

FraudCost.**optimize_threshold**(*ypred_proba*, *y_true*, *X=None*)

Learn a binary classification threshold which optimizes the current objective.

Parameters

- **ypred_proba** (*list*) – The classifier's predicted probabilities
- **y_true** (*list*) – The ground truth for the predictions.
- **X** (*pd.DataFrame, optional*) – Any extra columns that are needed from training data.

Returns Optimal threshold for this objective

evalml.objectives.FraudCost.score

FraudCost.**score**(*y_true*, *y_predicted*, *X=None*)

Returns a numerical score indicating performance based on the differences between the predicted and actual values.

Parameters

- **y_predicted** (*pd.Series*) – Predicted values of length [n_samples]

- **y_true** (*pd.Series*) – Actual class labels of length [n_samples]
- **X** (*pd.DataFrame or np.array*) – Extra data of shape [n_samples, n_features] necessary to calculate score

Returns score

evalml.objectives.FraudCost.validate_inputs

FraudCost.**validate_inputs**(*y_true*, *y_predicted*)

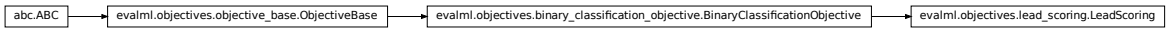
Validates the input based on a few simple checks.

Parameters

- **y_predicted** (*pd.Series*) – Predicted values of length [n_samples]
- **y_true** (*pd.Series*) – Actual class labels of length [n_samples]

Returns None

evalml.objectives.LeadScoring



class evalml.objectives.**LeadScoring**(*true_positives=1*, *false_positives=-1*)

Lead scoring.

Methods

__init__	Create instance.
calculate_percent_difference	Calculate the percent difference between scores.
decision_function	Apply a learned threshold to predicted probabilities to get predicted classes.
objective_function	Calculate the profit per lead.
optimize_threshold	Learn a binary classification threshold which optimizes the current objective.
score	Returns a numerical score indicating performance based on the differences between the predicted and actual values.
validate_inputs	Validates the input based on a few simple checks.

evalml.objectives.LeadScoring.__init__

LeadScoring.**__init__**(*true_positives=1*, *false_positives=-1*)

Create instance.

Parameters

- **true_positives** (*int*) – Reward for a true positive
- **false_positives** (*int*) – Cost for a false positive. Should be negative.

evalml.objectives.LeadScoring.calculate_percent_difference

classmethod LeadScoring.**calculate_percent_difference**(*score*, *baseline_score*)

Calculate the percent difference between scores.

Parameters

- **score** (*float*) – A score. Output of the score method of this objective.
- **baseline_score** (*float*) – A score. Output of the score method of this objective. In practice, this is the score achieved on this objective with a baseline estimator.

Returns

The percent difference between the scores. This will be the difference normalized by the baseline score.

Return type float

evalml.objectives.LeadScoring.decision_function

LeadScoring.**decision_function**(*ypred_proba*, *threshold=0.5*, *X=None*)

Apply a learned threshold to predicted probabilities to get predicted classes.

Parameters

- **ypred_proba** (*list*) – The classifier's predicted probabilities
- **threshold** (*float, optional*) – Threshold used to make a prediction. Defaults to 0.5.
- **X** (*pd.DataFrame, optional*) – Any extra columns that are needed from training data.

Returns predictions

evalml.objectives.LeadScoring.objective_function

LeadScoring.**objective_function**(*y_true*, *y_predicted*, *X=None*)

Calculate the profit per lead.

Parameters

- **y_predicted** (*pd.Series*) – Predicted labels
- **y_true** (*pd.Series*) – True labels
- **X** (*pd.DataFrame*) – None, not used.

Returns profit per lead

Return type float

evalml.objectives.LeadScoring.optimize_threshold

LeadScoring.**optimize_threshold**(*ypred_proba*, *y_true*, *X=None*)

Learn a binary classification threshold which optimizes the current objective.

Parameters

- **ypred_proba** (*list*) – The classifier's predicted probabilities
- **y_true** (*list*) – The ground truth for the predictions.
- **X** (*pd.DataFrame, optional*) – Any extra columns that are needed from training data.

Returns Optimal threshold for this objective

evalml.objectives.LeadScoring.score

LeadScoring.**score**(*y_true*, *y_predicted*, *X=None*)

Returns a numerical score indicating performance based on the differences between the predicted and actual values.

Parameters

- **y_predicted** (*pd.Series*) – Predicted values of length [n_samples]
- **y_true** (*pd.Series*) – Actual class labels of length [n_samples]
- **X** (*pd.DataFrame or np.array*) – Extra data of shape [n_samples, n_features] necessary to calculate score

Returns score

evalml.objectives.LeadScoring.validate_inputs

LeadScoring.**validate_inputs**(*y_true*, *y_predicted*)

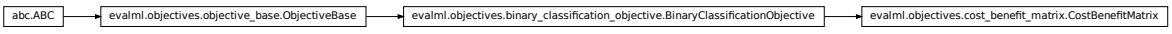
Validates the input based on a few simple checks.

Parameters

- **y_predicted** (*pd.Series*) – Predicted values of length [n_samples]
- **y_true** (*pd.Series*) – Actual class labels of length [n_samples]

Returns None

evalml.objectives.CostBenefitMatrix



class evalml.objectives.**CostBenefitMatrix**(*true_positive*, *true_negative*, *false_positive*, *false_negative*)

Score using a cost-benefit matrix. Scores quantify the benefits of a given value, so greater numeric scores represents a better score. Costs and scores can be negative, indicating that a value is not beneficial. For example, in the case of monetary profit, a negative cost and/or score represents loss of cash flow.

Methods

__init__	Create instance of CostBenefitMatrix.
calculate_percent_difference	Calculate the percent difference between scores.
decision_function	Apply a learned threshold to predicted probabilities to get predicted classes.
objective_function	Calculates cost-benefit of the using the predicted and true values.
optimize_threshold	Learn a binary classification threshold which optimizes the current objective.
score	Returns a numerical score indicating performance based on the differences between the predicted and actual values.
validate_inputs	Validates the input based on a few simple checks.

evalml.objectives.CostBenefitMatrix.__init__

CostBenefitMatrix.**__init__**(*true_positive*, *true_negative*, *false_positive*, *false_negative*)

Create instance of CostBenefitMatrix.

Parameters

- **true_positive** (*float*) – Cost associated with true positive predictions
- **true_negative** (*float*) – Cost associated with true negative predictions
- **false_positive** (*float*) – Cost associated with false positive predictions
- **false_negative** (*float*) – Cost associated with false negative predictions

evalml.objectives.CostBenefitMatrix.calculate_percent_difference

classmethod CostBenefitMatrix.**calculate_percent_difference**(*score*, *baseline_score*)

Calculate the percent difference between scores.

Parameters

- **score** (*float*) – A score. Output of the score method of this objective.
- **baseline_score** (*float*) – A score. Output of the score method of this objective. In practice, this is the score achieved on this objective with a baseline estimator.

Returns

The percent difference between the scores. This will be the difference normalized by the baseline score.

Return type float

evalml.objectives.CostBenefitMatrix.decision_function

CostBenefitMatrix.**decision_function**(*ypred_proba*, *threshold=0.5*, *X=None*)

Apply a learned threshold to predicted probabilities to get predicted classes.

Parameters

- **ypred_proba** (*list*) – The classifier's predicted probabilities

- **threshold** (*float, optional*) – Threshold used to make a prediction. Defaults to 0.5.
- **X** (*pd.DataFrame, optional*) – Any extra columns that are needed from training data.

Returns predictions

evalml.objectives.CostBenefitMatrix.objective_function

CostBenefitMatrix.**objective_function**(*y_true*, *y_predicted*, *X=None*)

Calculates cost-benefit of the using the predicted and true values.

Parameters

- **y_predicted** (*pd.Series*) – Predicted labels
- **y_true** (*pd.Series*) – True labels
- **X** (*pd.DataFrame*) – Ignored.

Returns Cost-benefit matrix score

Return type float

evalml.objectives.CostBenefitMatrix.optimize_threshold

CostBenefitMatrix.**optimize_threshold**(*ypred_proba*, *y_true*, *X=None*)

Learn a binary classification threshold which optimizes the current objective.

Parameters

- **ypred_proba** (*list*) – The classifier's predicted probabilities
- **y_true** (*list*) – The ground truth for the predictions.
- **X** (*pd.DataFrame, optional*) – Any extra columns that are needed from training data.

Returns Optimal threshold for this objective

evalml.objectives.CostBenefitMatrix.score

CostBenefitMatrix.**score**(*y_true*, *y_predicted*, *X=None*)

Returns a numerical score indicating performance based on the differences between the predicted and actual values.

Parameters

- **y_predicted** (*pd.Series*) – Predicted values of length [n_samples]
- **y_true** (*pd.Series*) – Actual class labels of length [n_samples]
- **X** (*pd.DataFrame or np.array*) – Extra data of shape [n_samples, n_features] necessary to calculate score

Returns score

evalml.objectives.CostBenefitMatrix.validate_inputs

CostBenefitMatrix.**validate_inputs**(*y_true*, *y_predicted*)
Validates the input based on a few simple checks.

Parameters

- **y_predicted** (*pd.Series*) – Predicted values of length [n_samples]
- **y_true** (*pd.Series*) – Actual class labels of length [n_samples]

Returns None

5.7.3 Classification Objectives

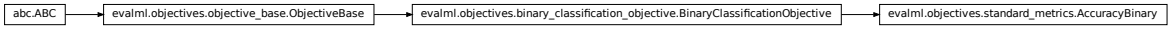
AccuracyBinary	Accuracy score for binary classification.
AccuracyMulticlass	Accuracy score for multiclass classification.
AUC	AUC score for binary classification.
AUCMacro	AUC score for multiclass classification using macro averaging.
AUCMicro	AUC score for multiclass classification using micro averaging.
AUCWeighted	AUC Score for multiclass classification using weighted averaging.
BalancedAccuracyBinary	Balanced accuracy score for binary classification.
BalancedAccuracyMulticlass	Balanced accuracy score for multiclass classification.
F1	F1 score for binary classification.
F1Micro	F1 score for multiclass classification using micro averaging.
F1Macro	F1 score for multiclass classification using macro averaging.
F1Weighted	F1 score for multiclass classification using weighted averaging.
LogLossBinary	Log Loss for binary classification.
LogLossMulticlass	Log Loss for multiclass classification.
MCCBinary	Matthews correlation coefficient for binary classification.
MCCMulticlass	Matthews correlation coefficient for multiclass classification.
Precision	Precision score for binary classification.
PrecisionMicro	Precision score for multiclass classification using micro averaging.
PrecisionMacro	Precision score for multiclass classification using macro averaging.
PrecisionWeighted	Precision score for multiclass classification using weighted averaging.
Recall	Recall score for binary classification.
RecallMicro	Recall score for multiclass classification using micro averaging.
RecallMacro	Recall score for multiclass classification using macro averaging.

Continued on next page

Table 111 – continued from previous page

RecallWeighted	Recall score for multiclass classification using weighted averaging.

evalml.objectives.AccuracyBinary



class evalml.objectives.**AccuracyBinary**

Accuracy score for binary classification.

Methods

calculate_percent_difference	Calculate the percent difference between scores.
decision_function	Apply a learned threshold to predicted probabilities to get predicted classes.
objective_function	Computes the relative value of the provided predictions compared to the actual labels, according a specified metric
optimize_threshold	Learn a binary classification threshold which optimizes the current objective.
score	Returns a numerical score indicating performance based on the differences between the predicted and actual values.
validate_inputs	Validates the input based on a few simple checks.

evalml.objectives.AccuracyBinary.calculate_percent_difference

classmethod AccuracyBinary.**calculate_percent_difference**(*score*, *baseline_score*)

Calculate the percent difference between scores.

Parameters

- **score** (*float*) – A score. Output of the score method of this objective.
- **baseline_score** (*float*) – A score. Output of the score method of this objective. In practice, this is the score achieved on this objective with a baseline estimator.

Returns

The percent difference between the scores. This will be the difference normalized by the baseline score.

Return type float

evalml.objectives.AccuracyBinary.decision_function

AccuracyBinary.**decision_function**(*ypred_proba*, *threshold=0.5*, *X=None*)

Apply a learned threshold to predicted probabilities to get predicted classes.

Parameters

- **ypred_proba** (*list*) – The classifier's predicted probabilities
- **threshold** (*float, optional*) – Threshold used to make a prediction. Defaults to 0.5.
- **X** (*pd.DataFrame, optional*) – Any extra columns that are needed from training data.

Returns predictions

evalml.objectives.AccuracyBinary.objective_function

AccuracyBinary.**objective_function**(*y_true*, *y_predicted*, *X=None*)

Computes the relative value of the provided predictions compared to the actual labels, according a specified metric

Arguments: y_predicted (pd.Series): Predicted values of length [n_samples] y_true (pd.Series): Actual class labels of length [n_samples] X (pd.DataFrame or np.array): Extra data of shape [n_samples, n_features] necessary to calculate score

Returns Numerical value used to calculate score

evalml.objectives.AccuracyBinary.optimize_threshold

AccuracyBinary.**optimize_threshold**(*ypred_proba*, *y_true*, *X=None*)

Learn a binary classification threshold which optimizes the current objective.

Parameters

- **ypred_proba** (*list*) – The classifier's predicted probabilities
- **y_true** (*list*) – The ground truth for the predictions.
- **X** (*pd.DataFrame, optional*) – Any extra columns that are needed from training data.

Returns Optimal threshold for this objective

evalml.objectives.AccuracyBinary.score

AccuracyBinary.**score**(*y_true*, *y_predicted*, *X=None*)

Returns a numerical score indicating performance based on the differences between the predicted and actual values.

Parameters

- **y_predicted** (*pd.Series*) – Predicted values of length [n_samples]
- **y_true** (*pd.Series*) – Actual class labels of length [n_samples]
- **X** (*pd.DataFrame or np.array*) – Extra data of shape [n_samples, n_features] necessary to calculate score

Returns score

evalml.objectives.AccuracyBinary.validate_inputs

AccuracyBinary.**validate_inputs**(*y_true*, *y_predicted*)

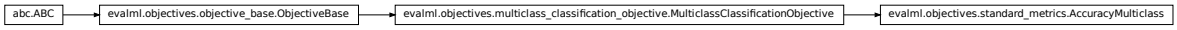
Validates the input based on a few simple checks.

Parameters

- **y_predicted** (*pd.Series*) – Predicted values of length [n_samples]
- **y_true** (*pd.Series*) – Actual class labels of length [n_samples]

Returns None

evalml.objectives.AccuracyMulticlass



class evalml.objectives.**AccuracyMulticlass**

Accuracy score for multiclass classification.

Methods

calculate_percent_difference	Calculate the percent difference between scores.
objective_function	Computes the relative value of the provided predictions compared to the actual labels, according a specified metric
score	Returns a numerical score indicating performance based on the differences between the predicted and actual values.
validate_inputs	Validates the input based on a few simple checks.

evalml.objectives.AccuracyMulticlass.calculate_percent_difference

classmethod AccuracyMulticlass.**calculate_percent_difference**(*score*, *baseline_score*)

Calculate the percent difference between scores.

Parameters

- **score** (*float*) – A score. Output of the score method of this objective.
- **baseline_score** (*float*) – A score. Output of the score method of this objective. In practice, this is the score achieved on this objective with a baseline estimator.

Returns

The percent difference between the scores. This will be the difference normalized by the baseline score.

Return type float

evalml.objectives.AccuracyMulticlass.objective_function

AccuracyMulticlass.**objective_function**(*y_true*, *y_predicted*, *X=None*)

Computes the relative value of the provided predictions compared to the actual labels, according a specified metric

Arguments: y_predicted (pd.Series): Predicted values of length [n_samples] y_true (pd.Series): Actual class labels of length [n_samples] X (pd.DataFrame or np.array): Extra data of shape [n_samples, n_features] necessary to calculate score

Returns Numerical value used to calculate score

evalml.objectives.AccuracyMulticlass.score

AccuracyMulticlass.**score**(*y_true*, *y_predicted*, *X=None*)

Returns a numerical score indicating performance based on the differences between the predicted and actual values.

Parameters

- **y_predicted** (*pd.Series*) – Predicted values of length [n_samples]
- **y_true** (*pd.Series*) – Actual class labels of length [n_samples]
- **X** (*pd.DataFrame or np.array*) – Extra data of shape [n_samples, n_features] necessary to calculate score

Returns score

evalml.objectives.AccuracyMulticlass.validate_inputs

AccuracyMulticlass.**validate_inputs**(*y_true*, *y_predicted*)

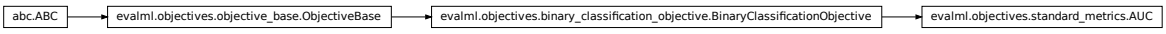
Validates the input based on a few simple checks.

Parameters

- **y_predicted** (*pd.Series*) – Predicted values of length [n_samples]
- **y_true** (*pd.Series*) – Actual class labels of length [n_samples]

Returns None

evalml.objectives.AUC



class evalml.objectives.**AUC**

AUC score for binary classification.

Methods

calculate_percent_difference	Calculate the percent difference between scores.
decision_function	Apply a learned threshold to predicted probabilities to get predicted classes.
objective_function	Computes the relative value of the provided predictions compared to the actual labels, according a specified metric
optimize_threshold	Learn a binary classification threshold which optimizes the current objective.
score	Returns a numerical score indicating performance based on the differences between the predicted and actual values.
validate_inputs	Validates the input based on a few simple checks.

evalml.objectives.AUC.calculate_percent_difference

classmethod AUC.**calculate_percent_difference**(*score*, *baseline_score*)

Calculate the percent difference between scores.

Parameters

- **score** (*float*) – A score. Output of the score method of this objective.
- **baseline_score** (*float*) – A score. Output of the score method of this objective. In practice, this is the score achieved on this objective with a baseline estimator.

Returns

The percent difference between the scores. This will be the difference normalized by the baseline score.

Return type float

evalml.objectives.AUC.decision_function

AUC.**decision_function**(*ypred_proba*, *threshold=0.5*, *X=None*)

Apply a learned threshold to predicted probabilities to get predicted classes.

Parameters

- **ypred_proba** (*list*) – The classifier's predicted probabilities
- **threshold** (*float, optional*) – Threshold used to make a prediction. Defaults to 0.5.
- **X** (*pd.DataFrame, optional*) – Any extra columns that are needed from training data.

Returns predictions

evalml.objectives.AUC.objective_function

AUC.**objective_function**(*y_true*, *y_predicted*, *X=None*)

Computes the relative value of the provided predictions compared to the actual labels, according a specified metric

Arguments: y_predicted (pd.Series): Predicted values of length [n_samples] y_true (pd.Series): Actual class labels of length [n_samples] X (pd.DataFrame or np.array): Extra data of shape [n_samples, n_features] necessary to calculate score

Returns Numerical value used to calculate score

evalml.objectives.AUC.optimize_threshold

AUC.**optimize_threshold**(*ypred_proba*, *y_true*, *X=None*)

Learn a binary classification threshold which optimizes the current objective.

Parameters

- **ypred_proba** (*list*) – The classifier's predicted probabilities
- **y_true** (*list*) – The ground truth for the predictions.
- **X** (*pd.DataFrame, optional*) – Any extra columns that are needed from training data.

Returns Optimal threshold for this objective

evalml.objectives.AUC.score

AUC.**score**(*y_true*, *y_predicted*, *X=None*)

Returns a numerical score indicating performance based on the differences between the predicted and actual values.

Parameters

- **y_predicted** (*pd.Series*) – Predicted values of length [n_samples]
- **y_true** (*pd.Series*) – Actual class labels of length [n_samples]
- **X** (*pd.DataFrame or np.array*) – Extra data of shape [n_samples, n_features] necessary to calculate score

Returns score

evalml.objectives.AUC.validate_inputs

AUC.**validate_inputs**(*y_true*, *y_predicted*)

Validates the input based on a few simple checks.

Parameters

- **y_predicted** (*pd.Series*) – Predicted values of length [n_samples]
- **y_true** (*pd.Series*) – Actual class labels of length [n_samples]

Returns None

evalml.objectives.AUCMacro

class evalml.objectives.**AUCMacro**

AUC score for multiclass classification using macro averaging.

Methods

calculate_percent_difference	Calculate the percent difference between scores.
objective_function	Computes the relative value of the provided predictions compared to the actual labels, according a specified metric
score	Returns a numerical score indicating performance based on the differences between the predicted and actual values.
validate_inputs	Validates the input based on a few simple checks.

evalml.objectives.AUCMacro.calculate_percent_difference

classmethod AUCMacro.**calculate_percent_difference**(*score*, *baseline_score*)

Calculate the percent difference between scores.

Parameters

- **score** (*float*) – A score. Output of the score method of this objective.
- **baseline_score** (*float*) – A score. Output of the score method of this objective. In practice, this is the score achieved on this objective with a baseline estimator.

Returns

The percent difference between the scores. This will be the difference normalized by the baseline score.

Return type float

evalml.objectives.AUCMacro.objective_function

AUCMacro.**objective_function**(*y_true*, *y_predicted*, *X=None*)

Computes the relative value of the provided predictions compared to the actual labels, according a specified metric

Arguments: y_predicted (pd.Series): Predicted values of length [n_samples] y_true (pd.Series): Actual class labels of length [n_samples] X (pd.DataFrame or np.array): Extra data of shape [n_samples, n_features] necessary to calculate score

Returns Numerical value used to calculate score

evalml.objectives.AUCMacro.score

AUCMacro.**score**(*y_true*, *y_predicted*, *X=None*)

Returns a numerical score indicating performance based on the differences between the predicted and actual values.

Parameters

- **y_predicted** (*pd.Series*) – Predicted values of length [n_samples]
- **y_true** (*pd.Series*) – Actual class labels of length [n_samples]
- **X** (*pd.DataFrame or np.array*) – Extra data of shape [n_samples, n_features] necessary to calculate score

Returns score

evalml.objectives.AUCMacro.validate_inputs

AUCMacro.**validate_inputs**(*y_true*, *y_predicted*)

Validates the input based on a few simple checks.

Parameters

- **y_predicted** (*pd.Series*) – Predicted values of length [n_samples]
- **y_true** (*pd.Series*) – Actual class labels of length [n_samples]

Returns None

evalml.objectives.AUCMicro

class evalml.objectives.**AUCMicro**

AUC score for multiclass classification using micro averaging.

Methods

calculate_percent_difference	Calculate the percent difference between scores.
objective_function	Computes the relative value of the provided predictions compared to the actual labels, according a specified metric
score	Returns a numerical score indicating performance based on the differences between the predicted and actual values.
validate_inputs	Validates the input based on a few simple checks.

evalml.objectives.AUCMicro.calculate_percent_difference

classmethod AUCMicro.**calculate_percent_difference**(*score*, *baseline_score*)

Calculate the percent difference between scores.

Parameters

- **score** (*float*) – A score. Output of the score method of this objective.
- **baseline_score** (*float*) – A score. Output of the score method of this objective. In practice, this is the score achieved on this objective with a baseline estimator.

Returns

The percent difference between the scores. This will be the difference normalized by the baseline score.

Return type float

evalml.objectives.AUCMicro.objective_function

AUCMicro.**objective_function**(*y_true*, *y_predicted*, *X=None*)

Computes the relative value of the provided predictions compared to the actual labels, according a specified metric

Arguments: y_predicted (pd.Series): Predicted values of length [n_samples] y_true (pd.Series): Actual class labels of length [n_samples] X (pd.DataFrame or np.array): Extra data of shape [n_samples, n_features] necessary to calculate score

Returns Numerical value used to calculate score

evalml.objectives.AUCMicro.score

AUCMicro.**score**(*y_true*, *y_predicted*, *X=None*)

Returns a numerical score indicating performance based on the differences between the predicted and actual values.

Parameters

- **y_predicted** (*pd.Series*) – Predicted values of length [n_samples]
- **y_true** (*pd.Series*) – Actual class labels of length [n_samples]
- **X** (*pd.DataFrame or np.array*) – Extra data of shape [n_samples, n_features] necessary to calculate score

Returns score

evalml.objectives.AUCMicro.validate_inputs

AUCMicro.**validate_inputs**(*y_true*, *y_predicted*)

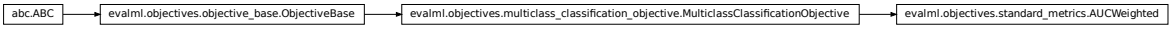
Validates the input based on a few simple checks.

Parameters

- **y_predicted** (*pd.Series*) – Predicted values of length [n_samples]
- **y_true** (*pd.Series*) – Actual class labels of length [n_samples]

Returns None

evalml.objectives.AUCWeighted



class evalml.objectives.**AUCWeighted**

AUC Score for multiclass classification using weighted averaging.

Methods

calculate_percent_difference	Calculate the percent difference between scores.
objective_function	Computes the relative value of the provided predictions compared to the actual labels, according a specified metric
score	Returns a numerical score indicating performance based on the differences between the predicted and actual values.
validate_inputs	Validates the input based on a few simple checks.

evalml.objectives.AUCWeighted.calculate_percent_difference

classmethod AUCWeighted.**calculate_percent_difference**(*score*, *baseline_score*)

Calculate the percent difference between scores.

Parameters

- **score** (*float*) – A score. Output of the score method of this objective.
- **baseline_score** (*float*) – A score. Output of the score method of this objective. In practice, this is the score achieved on this objective with a baseline estimator.

Returns

The percent difference between the scores. This will be the difference normalized by the baseline score.

Return type float

evalml.objectives.AUCWeighted.objective_function

AUCWeighted.**objective_function**(*y_true*, *y_predicted*, *X=None*)

Computes the relative value of the provided predictions compared to the actual labels, according a specified metric

Arguments: y_predicted (pd.Series): Predicted values of length [n_samples] y_true (pd.Series): Actual class labels of length [n_samples] X (pd.DataFrame or np.array): Extra data of shape [n_samples, n_features] necessary to calculate score

Returns Numerical value used to calculate score

evalml.objectives.AUCWeighted.score

AUCWeighted.**score**(*y_true*, *y_predicted*, *X=None*)

Returns a numerical score indicating performance based on the differences between the predicted and actual values.

Parameters

- **y_predicted** (*pd.Series*) – Predicted values of length [n_samples]
- **y_true** (*pd.Series*) – Actual class labels of length [n_samples]
- **X** (*pd.DataFrame or np.array*) – Extra data of shape [n_samples, n_features] necessary to calculate score

Returns score

evalml.objectives.AUCWeighted.validate_inputs

AUCWeighted.**validate_inputs**(*y_true*, *y_predicted*)

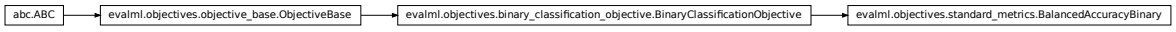
Validates the input based on a few simple checks.

Parameters

- **y_predicted** (*pd.Series*) – Predicted values of length [n_samples]
- **y_true** (*pd.Series*) – Actual class labels of length [n_samples]

Returns None

evalml.objectives.BalancedAccuracyBinary



class evalml.objectives.**BalancedAccuracyBinary**

Balanced accuracy score for binary classification.

Methods

calculate_percent_difference	Calculate the percent difference between scores.
decision_function	Apply a learned threshold to predicted probabilities to get predicted classes.
objective_function	Computes the relative value of the provided predictions compared to the actual labels, according a specified metric
optimize_threshold	Learn a binary classification threshold which optimizes the current objective.
score	Returns a numerical score indicating performance based on the differences between the predicted and actual values.
validate_inputs	Validates the input based on a few simple checks.

evalml.objectives.BalancedAccuracyBinary.calculate_percent_difference

classmethod BalancedAccuracyBinary.**calculate_percent_difference**(*score*, *baseline_score*)

Calculate the percent difference between scores.

Parameters

- **score** (*float*) – A score. Output of the score method of this objective.
- **baseline_score** (*float*) – A score. Output of the score method of this objective. In practice, this is the score achieved on this objective with a baseline estimator.

Returns

The percent difference between the scores. This will be the difference normalized by the baseline score.

Return type float

evalml.objectives.BalancedAccuracyBinary.decision_function

BalancedAccuracyBinary.**decision_function**(*ypred_proba*, *threshold=0.5*, *X=None*)

Apply a learned threshold to predicted probabilities to get predicted classes.

Parameters

- **ypred_proba** (*list*) – The classifier's predicted probabilities
- **threshold** (*float, optional*) – Threshold used to make a prediction. Defaults to 0.5.
- **X** (*pd.DataFrame, optional*) – Any extra columns that are needed from training data.

Returns predictions

evalml.objectives.BalancedAccuracyBinary.objective_function

BalancedAccuracyBinary.**objective_function**(*y_true*, *y_predicted*, *X=None*)

Computes the relative value of the provided predictions compared to the actual labels, according a specified metric

Arguments: y_predicted (pd.Series): Predicted values of length [n_samples] y_true (pd.Series): Actual class labels of length [n_samples] X (pd.DataFrame or np.array): Extra data of shape [n_samples, n_features] necessary to calculate score

Returns Numerical value used to calculate score

evalml.objectives.BalancedAccuracyBinary.optimize_threshold

BalancedAccuracyBinary.**optimize_threshold**(*ypred_proba*, *y_true*, *X=None*)

Learn a binary classification threshold which optimizes the current objective.

Parameters

- **ypred_proba** (*list*) – The classifier's predicted probabilities
- **y_true** (*list*) – The ground truth for the predictions.
- **X** (*pd.DataFrame, optional*) – Any extra columns that are needed from training data.

Returns Optimal threshold for this objective

evalml.objectives.BalancedAccuracyBinary.score

BalancedAccuracyBinary.**score**(*y_true*, *y_predicted*, *X=None*)

Returns a numerical score indicating performance based on the differences between the predicted and actual values.

Parameters

- **y_predicted** (*pd.Series*) – Predicted values of length [n_samples]
- **y_true** (*pd.Series*) – Actual class labels of length [n_samples]
- **X** (*pd.DataFrame or np.array*) – Extra data of shape [n_samples, n_features] necessary to calculate score

Returns score

evalml.objectives.BalancedAccuracyBinary.validate_inputs

BalancedAccuracyBinary.**validate_inputs**(*y_true*, *y_predicted*)

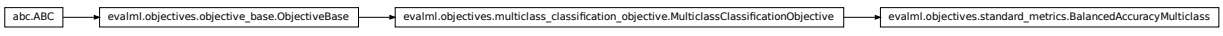
Validates the input based on a few simple checks.

Parameters

- **y_predicted** (*pd.Series*) – Predicted values of length [n_samples]
- **y_true** (*pd.Series*) – Actual class labels of length [n_samples]

Returns None

evalml.objectives.BalancedAccuracyMulticlass



class evalml.objectives.**BalancedAccuracyMulticlass**

Balanced accuracy score for multiclass classification.

Methods

calculate_percent_difference	Calculate the percent difference between scores.
objective_function	Computes the relative value of the provided predictions compared to the actual labels, according a specified metric

Continued on next page

Table 119 – continued from previous page

score	Returns a numerical score indicating performance based on the differences between the predicted and actual values.
validate_inputs	Validates the input based on a few simple checks.

evalml.objectives.BalancedAccuracyMulticlass.calculate_percent_difference

classmethod BalancedAccuracyMulticlass.**calculate_percent_difference**(*score*, *baseline_score*)

Calculate the percent difference between scores.

Parameters

- **score** (*float*) – A score. Output of the score method of this objective.
- **baseline_score** (*float*) – A score. Output of the score method of this objective. In practice, this is the score achieved on this objective with a baseline estimator.

Returns

The percent difference between the scores. This will be the difference normalized by the baseline score.

Return type float

evalml.objectives.BalancedAccuracyMulticlass.objective_function

BalancedAccuracyMulticlass.**objective_function**(*y_true*, *y_predicted*, *X=None*)

Computes the relative value of the provided predictions compared to the actual labels, according a specified metric

Arguments: y_predicted (pd.Series): Predicted values of length [n_samples] y_true (pd.Series): Actual class labels of length [n_samples] X (pd.DataFrame or np.array): Extra data of shape [n_samples, n_features] necessary to calculate score

Returns Numerical value used to calculate score

evalml.objectives.BalancedAccuracyMulticlass.score

BalancedAccuracyMulticlass.**score**(*y_true*, *y_predicted*, *X=None*)

Returns a numerical score indicating performance based on the differences between the predicted and actual values.

Parameters

- **y_predicted** (*pd.Series*) – Predicted values of length [n_samples]
- **y_true** (*pd.Series*) – Actual class labels of length [n_samples]
- **X** (*pd.DataFrame or np.array*) – Extra data of shape [n_samples, n_features] necessary to calculate score

Returns score

evalml.objectives.BalancedAccuracyMulticlass.validate_inputs

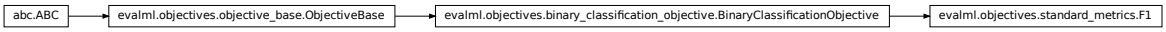
BalancedAccuracyMulticlass.**validate_inputs**(*y_true*, *y_predicted*)
Validates the input based on a few simple checks.

Parameters

- **y_predicted** (*pd.Series*) – Predicted values of length [n_samples]
- **y_true** (*pd.Series*) – Actual class labels of length [n_samples]

Returns None

evalml.objectives.F1



class evalml.objectives.**F1**
F1 score for binary classification.

Methods

calculate_percent_difference	Calculate the percent difference between scores.
decision_function	Apply a learned threshold to predicted probabilities to get predicted classes.
objective_function	Computes the relative value of the provided predictions compared to the actual labels, according a specified metric
optimize_threshold	Learn a binary classification threshold which optimizes the current objective.
score	Returns a numerical score indicating performance based on the differences between the predicted and actual values.
validate_inputs	Validates the input based on a few simple checks.

evalml.objectives.F1.calculate_percent_difference

classmethod F1.**calculate_percent_difference**(*score*, *baseline_score*)
Calculate the percent difference between scores.

Parameters

- **score** (*float*) – A score. Output of the score method of this objective.
- **baseline_score** (*float*) – A score. Output of the score method of this objective. In practice, this is the score achieved on this objective with a baseline estimator.

Returns

The percent difference between the scores. This will be the difference normalized by the baseline score.

Return type float

evalml.objectives.F1.decision_function

F1.**decision_function**(*ypred_proba*, *threshold=0.5*, *X=None*)

Apply a learned threshold to predicted probabilities to get predicted classes.

Parameters

- **ypred_proba** (*list*) – The classifier's predicted probabilities
- **threshold** (*float, optional*) – Threshold used to make a prediction. Defaults to 0.5.
- **X** (*pd.DataFrame, optional*) – Any extra columns that are needed from training data.

Returns predictions

evalml.objectives.F1.objective_function

F1.**objective_function**(*y_true*, *y_predicted*, *X=None*)

Computes the relative value of the provided predictions compared to the actual labels, according a specified metric

Arguments: y_predicted (pd.Series): Predicted values of length [n_samples] y_true (pd.Series): Actual class labels of length [n_samples] X (pd.DataFrame or np.array): Extra data of shape [n_samples, n_features] necessary to calculate score

Returns Numerical value used to calculate score

evalml.objectives.F1.optimize_threshold

F1.**optimize_threshold**(*ypred_proba*, *y_true*, *X=None*)

Learn a binary classification threshold which optimizes the current objective.

Parameters

- **ypred_proba** (*list*) – The classifier's predicted probabilities
- **y_true** (*list*) – The ground truth for the predictions.
- **X** (*pd.DataFrame, optional*) – Any extra columns that are needed from training data.

Returns Optimal threshold for this objective

evalml.objectives.F1.score

F1.**score**(*y_true*, *y_predicted*, *X=None*)

Returns a numerical score indicating performance based on the differences between the predicted and actual values.

Parameters

- **y_predicted** (*pd.Series*) – Predicted values of length [n_samples]

- **y_true** (*pd.Series*) – Actual class labels of length [n_samples]
- **X** (*pd.DataFrame or np.array*) – Extra data of shape [n_samples, n_features] necessary to calculate score

Returns score

evalml.objectives.F1.validate_inputs

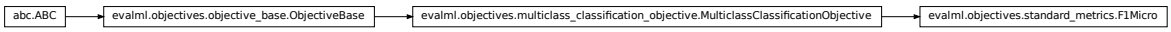
F1.**validate_inputs**(*y_true*, *y_predicted*)
Validates the input based on a few simple checks.

Parameters

- **y_predicted** (*pd.Series*) – Predicted values of length [n_samples]
- **y_true** (*pd.Series*) – Actual class labels of length [n_samples]

Returns None

evalml.objectives.F1Micro



class evalml.objectives.**F1Micro**
F1 score for multiclass classification using micro averaging.

Methods

calculate_percent_difference	Calculate the percent difference between scores.
objective_function	Computes the relative value of the provided predictions compared to the actual labels, according a specified metric
score	Returns a numerical score indicating performance based on the differences between the predicted and actual values.
validate_inputs	Validates the input based on a few simple checks.

evalml.objectives.F1Micro.calculate_percent_difference

classmethod F1Micro.**calculate_percent_difference**(*score*, *baseline_score*)
Calculate the percent difference between scores.

Parameters

- **score** (*float*) – A score. Output of the score method of this objective.
- **baseline_score** (*float*) – A score. Output of the score method of this objective. In practice, this is the score achieved on this objective with a baseline estimator.

Returns

The percent difference between the scores. This will be the difference normalized by the baseline score.

Return type float

evalml.objectives.F1Micro.objective_function

F1Micro.**objective_function**(*y_true*, *y_predicted*, *X=None*)

Computes the relative value of the provided predictions compared to the actual labels, according a specified metric

Arguments: y_predicted (pd.Series): Predicted values of length [n_samples] y_true (pd.Series): Actual class labels of length [n_samples] X (pd.DataFrame or np.array): Extra data of shape [n_samples, n_features] necessary to calculate score

Returns Numerical value used to calculate score

evalml.objectives.F1Micro.score

F1Micro.**score**(*y_true*, *y_predicted*, *X=None*)

Returns a numerical score indicating performance based on the differences between the predicted and actual values.

Parameters

- **y_predicted** (*pd.Series*) – Predicted values of length [n_samples]
- **y_true** (*pd.Series*) – Actual class labels of length [n_samples]
- **X** (*pd.DataFrame or np.array*) – Extra data of shape [n_samples, n_features] necessary to calculate score

Returns score

evalml.objectives.F1Micro.validate_inputs

F1Micro.**validate_inputs**(*y_true*, *y_predicted*)

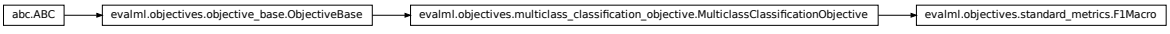
Validates the input based on a few simple checks.

Parameters

- **y_predicted** (*pd.Series*) – Predicted values of length [n_samples]
- **y_true** (*pd.Series*) – Actual class labels of length [n_samples]

Returns None

evalml.objectives.F1Macro



class evalml.objectives.**F1Macro**

F1 score for multiclass classification using macro averaging.

Methods

calculate_percent_difference	Calculate the percent difference between scores.
objective_function	Computes the relative value of the provided predictions compared to the actual labels, according a specified metric
score	Returns a numerical score indicating performance based on the differences between the predicted and actual values.
validate_inputs	Validates the input based on a few simple checks.

evalml.objectives.F1Macro.calculate_percent_difference

classmethod F1Macro.**calculate_percent_difference**(*score*, *baseline_score*)

Calculate the percent difference between scores.

Parameters

- **score** (*float*) – A score. Output of the score method of this objective.
- **baseline_score** (*float*) – A score. Output of the score method of this objective. In practice, this is the score achieved on this objective with a baseline estimator.

Returns

The percent difference between the scores. This will be the difference normalized by the baseline score.

Return type float

evalml.objectives.F1Macro.objective_function

F1Macro.**objective_function**(*y_true*, *y_predicted*, *X=None*)

Computes the relative value of the provided predictions compared to the actual labels, according a specified metric

Arguments: y_predicted (pd.Series): Predicted values of length [n_samples] y_true (pd.Series): Actual class labels of length [n_samples] X (pd.DataFrame or np.array): Extra data of shape [n_samples, n_features] necessary to calculate score

Returns Numerical value used to calculate score

evalml.objectives.F1Macro.score

F1Macro.**score**(*y_true*, *y_predicted*, *X=None*)

Returns a numerical score indicating performance based on the differences between the predicted and actual values.

Parameters

- **y_predicted** (*pd.Series*) – Predicted values of length [n_samples]

- **y_true** (*pd.Series*) – Actual class labels of length [n_samples]
- **X** (*pd.DataFrame or np.array*) – Extra data of shape [n_samples, n_features] necessary to calculate score

Returns score

evalml.objectives.F1Macro.validate_inputs

F1Macro.**validate_inputs**(*y_true*, *y_predicted*)
Validates the input based on a few simple checks.

Parameters

- **y_predicted** (*pd.Series*) – Predicted values of length [n_samples]
- **y_true** (*pd.Series*) – Actual class labels of length [n_samples]

Returns None

evalml.objectives.F1Weighted

class evalml.objectives.**F1Weighted**
F1 score for multiclass classification using weighted averaging.

Methods

calculate_percent_difference	Calculate the percent difference between scores.
objective_function	Computes the relative value of the provided predictions compared to the actual labels, according a specified metric
score	Returns a numerical score indicating performance based on the differences between the predicted and actual values.
validate_inputs	Validates the input based on a few simple checks.

evalml.objectives.F1Weighted.calculate_percent_difference

classmethod F1Weighted.**calculate_percent_difference**(*score*, *baseline_score*)
Calculate the percent difference between scores.

Parameters

- **score** (*float*) – A score. Output of the score method of this objective.
- **baseline_score** (*float*) – A score. Output of the score method of this objective. In practice, this is the score achieved on this objective with a baseline estimator.

Returns

The percent difference between the scores. This will be the difference normalized by the baseline score.

Return type float

evalml.objectives.F1Weighted.objective_function

F1Weighted.**objective_function**(*y_true*, *y_predicted*, *X=None*)

Computes the relative value of the provided predictions compared to the actual labels, according a specified metric

Arguments: y_predicted (pd.Series): Predicted values of length [n_samples] y_true (pd.Series): Actual class labels of length [n_samples] X (pd.DataFrame or np.array): Extra data of shape [n_samples, n_features] necessary to calculate score

Returns Numerical value used to calculate score

evalml.objectives.F1Weighted.score

F1Weighted.**score**(*y_true*, *y_predicted*, *X=None*)

Returns a numerical score indicating performance based on the differences between the predicted and actual values.

Parameters

- **y_predicted** (*pd.Series*) – Predicted values of length [n_samples]
- **y_true** (*pd.Series*) – Actual class labels of length [n_samples]
- **X** (*pd.DataFrame or np.array*) – Extra data of shape [n_samples, n_features] necessary to calculate score

Returns score

evalml.objectives.F1Weighted.validate_inputs

F1Weighted.**validate_inputs**(*y_true*, *y_predicted*)

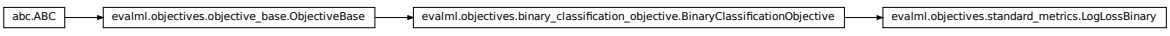
Validates the input based on a few simple checks.

Parameters

- **y_predicted** (*pd.Series*) – Predicted values of length [n_samples]
- **y_true** (*pd.Series*) – Actual class labels of length [n_samples]

Returns None

evalml.objectives.LogLossBinary



class evalml.objectives.**LogLossBinary**

Log Loss for binary classification.

Methods

calculate_percent_difference	Calculate the percent difference between scores.
decision_function	Apply a learned threshold to predicted probabilities to get predicted classes.
objective_function	Computes the relative value of the provided predictions compared to the actual labels, according a specified metric
optimize_threshold	Learn a binary classification threshold which optimizes the current objective.
score	Returns a numerical score indicating performance based on the differences between the predicted and actual values.
validate_inputs	Validates the input based on a few simple checks.

evalml.objectives.LogLossBinary.calculate_percent_difference

classmethod LogLossBinary.**calculate_percent_difference**(*score*, *baseline_score*)

Calculate the percent difference between scores.

Parameters

- **score** (*float*) – A score. Output of the score method of this objective.
- **baseline_score** (*float*) – A score. Output of the score method of this objective. In practice, this is the score achieved on this objective with a baseline estimator.

Returns

The percent difference between the scores. This will be the difference normalized by the baseline score.

Return type float

evalml.objectives.LogLossBinary.decision_function

LogLossBinary.**decision_function**(*ypred_proba*, *threshold=0.5*, *X=None*)

Apply a learned threshold to predicted probabilities to get predicted classes.

Parameters

- **ypred_proba** (*list*) – The classifier's predicted probabilities
- **threshold** (*float, optional*) – Threshold used to make a prediction. Defaults to 0.5.
- **X** (*pd.DataFrame, optional*) – Any extra columns that are needed from training data.

Returns predictions

evalml.objectives.LogLossBinary.objective_function

LogLossBinary.**objective_function**(*y_true*, *y_predicted*, *X=None*)

Computes the relative value of the provided predictions compared to the actual labels, according a specified metric

Arguments: y_predicted (pd.Series): Predicted values of length [n_samples] y_true (pd.Series): Actual class labels of length [n_samples] X (pd.DataFrame or np.array): Extra data of shape [n_samples, n_features] necessary to calculate score

Returns Numerical value used to calculate score

evalml.objectives.LogLossBinary.optimize_threshold

LogLossBinary.**optimize_threshold**(*ypred_proba*, *y_true*, *X=None*)

Learn a binary classification threshold which optimizes the current objective.

Parameters

- **ypred_proba** (*list*) – The classifier's predicted probabilities
- **y_true** (*list*) – The ground truth for the predictions.
- **X** (*pd.DataFrame, optional*) – Any extra columns that are needed from training data.

Returns Optimal threshold for this objective

evalml.objectives.LogLossBinary.score

LogLossBinary.**score**(*y_true*, *y_predicted*, *X=None*)

Returns a numerical score indicating performance based on the differences between the predicted and actual values.

Parameters

- **y_predicted** (*pd.Series*) – Predicted values of length [n_samples]
- **y_true** (*pd.Series*) – Actual class labels of length [n_samples]
- **X** (*pd.DataFrame or np.array*) – Extra data of shape [n_samples, n_features] necessary to calculate score

Returns score

evalml.objectives.LogLossBinary.validate_inputs

LogLossBinary.**validate_inputs**(*y_true*, *y_predicted*)

Validates the input based on a few simple checks.

Parameters

- **y_predicted** (*pd.Series*) – Predicted values of length [n_samples]
- **y_true** (*pd.Series*) – Actual class labels of length [n_samples]

Returns None

evalml.objectives.LogLossMulticlass

class evalml.objectives.**LogLossMulticlass**

Log Loss for multiclass classification.

Methods

calculate_percent_difference	Calculate the percent difference between scores.
objective_function	Computes the relative value of the provided predictions compared to the actual labels, according a specified metric
score	Returns a numerical score indicating performance based on the differences between the predicted and actual values.
validate_inputs	Validates the input based on a few simple checks.

evalml.objectives.LogLossMulticlass.calculate_percent_difference

classmethod LogLossMulticlass.**calculate_percent_difference**(*score*, *baseline_score*)

Calculate the percent difference between scores.

Parameters

- **score** (*float*) – A score. Output of the score method of this objective.
- **baseline_score** (*float*) – A score. Output of the score method of this objective. In practice, this is the score achieved on this objective with a baseline estimator.

Returns

The percent difference between the scores. This will be the difference normalized by the baseline score.

Return type float

evalml.objectives.LogLossMulticlass.objective_function

LogLossMulticlass.**objective_function**(*y_true*, *y_predicted*, *X=None*)

Computes the relative value of the provided predictions compared to the actual labels, according a specified metric

Arguments: y_predicted (pd.Series): Predicted values of length [n_samples] y_true (pd.Series): Actual class labels of length [n_samples] X (pd.DataFrame or np.array): Extra data of shape [n_samples, n_features] necessary to calculate score

Returns Numerical value used to calculate score

evalml.objectives.LogLossMulticlass.score

LogLossMulticlass.**score**(*y_true*, *y_predicted*, *X=None*)

Returns a numerical score indicating performance based on the differences between the predicted and actual values.

Parameters

- **y_predicted** (*pd.Series*) – Predicted values of length [n_samples]
- **y_true** (*pd.Series*) – Actual class labels of length [n_samples]
- **X** (*pd.DataFrame or np.array*) – Extra data of shape [n_samples, n_features] necessary to calculate score

Returns score

evalml.objectives.LogLossMulticlass.validate_inputs

LogLossMulticlass.**validate_inputs**(*y_true*, *y_predicted*)

Validates the input based on a few simple checks.

Parameters

- **y_predicted** (*pd.Series*) – Predicted values of length [n_samples]
- **y_true** (*pd.Series*) – Actual class labels of length [n_samples]

Returns None

evalml.objectives.MCCBinary

class evalml.objectives.**MCCBinary**

Matthews correlation coefficient for binary classification.

Methods

calculate_percent_difference	Calculate the percent difference between scores.
decision_function	Apply a learned threshold to predicted probabilities to get predicted classes.
objective_function	Computes the relative value of the provided predictions compared to the actual labels, according a specified metric
optimize_threshold	Learn a binary classification threshold which optimizes the current objective.
score	Returns a numerical score indicating performance based on the differences between the predicted and actual values.
validate_inputs	Validates the input based on a few simple checks.

evalml.objectives.MCCBinary.calculate_percent_difference

classmethod MCCBinary.**calculate_percent_difference**(*score*, *baseline_score*)

Calculate the percent difference between scores.

Parameters

- **score** (*float*) – A score. Output of the score method of this objective.
- **baseline_score** (*float*) – A score. Output of the score method of this objective. In practice, this is the score achieved on this objective with a baseline estimator.

Returns

The percent difference between the scores. This will be the difference normalized by the baseline score.

Return type float

evalml.objectives.MCCBinary.decision_function

MCCBinary.**decision_function**(*ypred_proba*, *threshold=0.5*, *X=None*)

Apply a learned threshold to predicted probabilities to get predicted classes.

Parameters

- **ypred_proba** (*list*) – The classifier's predicted probabilities
- **threshold** (*float, optional*) – Threshold used to make a prediction. Defaults to 0.5.
- **X** (*pd.DataFrame, optional*) – Any extra columns that are needed from training data.

Returns predictions

evalml.objectives.MCCBinary.objective_function

MCCBinary.**objective_function**(*y_true*, *y_predicted*, *X=None*)

Computes the relative value of the provided predictions compared to the actual labels, according a specified metric

Arguments: y_predicted (pd.Series): Predicted values of length [n_samples] y_true (pd.Series): Actual class labels of length [n_samples] X (pd.DataFrame or np.array): Extra data of shape [n_samples, n_features] necessary to calculate score

Returns Numerical value used to calculate score

evalml.objectives.MCCBinary.optimize_threshold

MCCBinary.**optimize_threshold**(*ypred_proba*, *y_true*, *X=None*)

Learn a binary classification threshold which optimizes the current objective.

Parameters

- **ypred_proba** (*list*) – The classifier's predicted probabilities
- **y_true** (*list*) – The ground truth for the predictions.

- **X** (*pd.DataFrame, optional*) – Any extra columns that are needed from training data.

Returns Optimal threshold for this objective

evalml.objectives.MCCBinary.score

MCCBinary.**score**(*y_true*, *y_predicted*, *X=None*)

Returns a numerical score indicating performance based on the differences between the predicted and actual values.

Parameters

- **y_predicted** (*pd.Series*) – Predicted values of length [n_samples]
- **y_true** (*pd.Series*) – Actual class labels of length [n_samples]
- **X** (*pd.DataFrame or np.array*) – Extra data of shape [n_samples, n_features] necessary to calculate score

Returns score

evalml.objectives.MCCBinary.validate_inputs

MCCBinary.**validate_inputs**(*y_true*, *y_predicted*)

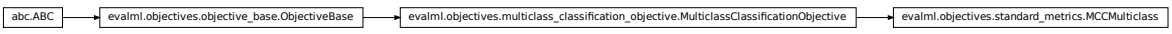
Validates the input based on a few simple checks.

Parameters

- **y_predicted** (*pd.Series*) – Predicted values of length [n_samples]
- **y_true** (*pd.Series*) – Actual class labels of length [n_samples]

Returns None

evalml.objectives.MCCMulticlass



class evalml.objectives.**MCCMulticlass**

Matthews correlation coefficient for multiclass classification.

Methods

calculate_percent_difference	Calculate the percent difference between scores.
objective_function	Computes the relative value of the provided predictions compared to the actual labels, according a specified metric
score	Returns a numerical score indicating performance based on the differences between the predicted and actual values.

Continued on next page

Table 127 – continued from previous page

validate_inputs	Validates the input based on a few simple checks.

evalml.objectives.MCCMulticlass.calculate_percent_difference

classmethod MCCMulticlass.**calculate_percent_difference**(*score*, *baseline_score*)

Calculate the percent difference between scores.

Parameters

- **score** (*float*) – A score. Output of the score method of this objective.
- **baseline_score** (*float*) – A score. Output of the score method of this objective. In practice, this is the score achieved on this objective with a baseline estimator.

Returns

The percent difference between the scores. This will be the difference normalized by the baseline score.

Return type float

evalml.objectives.MCCMulticlass.objective_function

MCCMulticlass.**objective_function**(*y_true*, *y_predicted*, *X=None*)

Computes the relative value of the provided predictions compared to the actual labels, according a specified metric

Arguments: y_predicted (pd.Series): Predicted values of length [n_samples] y_true (pd.Series): Actual class labels of length [n_samples] X (pd.DataFrame or np.array): Extra data of shape [n_samples, n_features] necessary to calculate score

Returns Numerical value used to calculate score

evalml.objectives.MCCMulticlass.score

MCCMulticlass.**score**(*y_true*, *y_predicted*, *X=None*)

Returns a numerical score indicating performance based on the differences between the predicted and actual values.

Parameters

- **y_predicted** (*pd.Series*) – Predicted values of length [n_samples]
- **y_true** (*pd.Series*) – Actual class labels of length [n_samples]
- **X** (*pd.DataFrame or np.array*) – Extra data of shape [n_samples, n_features] necessary to calculate score

Returns score

evalml.objectives.MCCMulticlass.validate_inputs

MCCMulticlass.**validate_inputs**(*y_true*, *y_predicted*)

Validates the input based on a few simple checks.

Parameters

- **y_predicted** (*pd.Series*) – Predicted values of length [n_samples]
- **y_true** (*pd.Series*) – Actual class labels of length [n_samples]

Returns None

evalml.objectives.Precision

class evalml.objectives.**Precision**

Precision score for binary classification.

Methods

calculate_percent_difference	Calculate the percent difference between scores.
decision_function	Apply a learned threshold to predicted probabilities to get predicted classes.
objective_function	Computes the relative value of the provided predictions compared to the actual labels, according a specified metric
optimize_threshold	Learn a binary classification threshold which optimizes the current objective.
score	Returns a numerical score indicating performance based on the differences between the predicted and actual values.
validate_inputs	Validates the input based on a few simple checks.

evalml.objectives.Precision.calculate_percent_difference

classmethod Precision.**calculate_percent_difference**(*score*, *baseline_score*)

Calculate the percent difference between scores.

Parameters

- **score** (*float*) – A score. Output of the score method of this objective.
- **baseline_score** (*float*) – A score. Output of the score method of this objective. In practice, this is the score achieved on this objective with a baseline estimator.

Returns

The percent difference between the scores. This will be the difference normalized by the baseline score.

Return type float

evalml.objectives.Precision.decision_function

Precision.**decision_function**(*ypred_proba*, *threshold=0.5*, *X=None*)

Apply a learned threshold to predicted probabilities to get predicted classes.

Parameters

- **ypred_proba** (*list*) – The classifier's predicted probabilities
- **threshold** (*float, optional*) – Threshold used to make a prediction. Defaults to 0.5.
- **X** (*pd.DataFrame, optional*) – Any extra columns that are needed from training data.

Returns predictions

evalml.objectives.Precision.objective_function

Precision.**objective_function**(*y_true*, *y_predicted*, *X=None*)

Computes the relative value of the provided predictions compared to the actual labels, according a specified metric

Arguments: y_predicted (pd.Series): Predicted values of length [n_samples] y_true (pd.Series): Actual class labels of length [n_samples] X (pd.DataFrame or np.array): Extra data of shape [n_samples, n_features] necessary to calculate score

Returns Numerical value used to calculate score

evalml.objectives.Precision.optimize_threshold

Precision.**optimize_threshold**(*ypred_proba*, *y_true*, *X=None*)

Learn a binary classification threshold which optimizes the current objective.

Parameters

- **ypred_proba** (*list*) – The classifier's predicted probabilities
- **y_true** (*list*) – The ground truth for the predictions.
- **X** (*pd.DataFrame, optional*) – Any extra columns that are needed from training data.

Returns Optimal threshold for this objective

evalml.objectives.Precision.score

Precision.**score**(*y_true*, *y_predicted*, *X=None*)

Returns a numerical score indicating performance based on the differences between the predicted and actual values.

Parameters

- **y_predicted** (*pd.Series*) – Predicted values of length [n_samples]
- **y_true** (*pd.Series*) – Actual class labels of length [n_samples]

- **X** (*pd.DataFrame or np.array*) – Extra data of shape [n_samples, n_features] necessary to calculate score

Returns score

evalml.objectives.Precision.validate_inputs

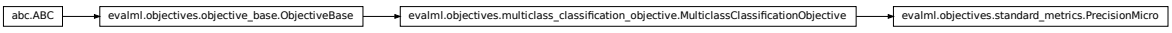
Precision.**validate_inputs**(*y_true*, *y_predicted*)
Validates the input based on a few simple checks.

Parameters

- **y_predicted** (*pd.Series*) – Predicted values of length [n_samples]
- **y_true** (*pd.Series*) – Actual class labels of length [n_samples]

Returns None

evalml.objectives.PrecisionMicro



class evalml.objectives.**PrecisionMicro**
Precision score for multiclass classification using micro averaging.

Methods

calculate_percent_difference	Calculate the percent difference between scores.
objective_function	Computes the relative value of the provided predictions compared to the actual labels, according a specified metric
score	Returns a numerical score indicating performance based on the differences between the predicted and actual values.
validate_inputs	Validates the input based on a few simple checks.

evalml.objectives.PrecisionMicro.calculate_percent_difference

classmethod PrecisionMicro.**calculate_percent_difference**(*score*, *baseline_score*)
Calculate the percent difference between scores.

Parameters

- **score** (*float*) – A score. Output of the score method of this objective.
- **baseline_score** (*float*) – A score. Output of the score method of this objective. In practice, this is the score achieved on this objective with a baseline estimator.

Returns

The percent difference between the scores. This will be the difference normalized by the

baseline score.

Return type float

evalml.objectives.PrecisionMicro.objective_function

PrecisionMicro.**objective_function**(*y_true*, *y_predicted*, *X=None*)

Computes the relative value of the provided predictions compared to the actual labels, according a specified metric

Arguments: y_predicted (pd.Series): Predicted values of length [n_samples] y_true (pd.Series): Actual class labels of length [n_samples] X (pd.DataFrame or np.array): Extra data of shape [n_samples, n_features] necessary to calculate score

Returns Numerical value used to calculate score

evalml.objectives.PrecisionMicro.score

PrecisionMicro.**score**(*y_true*, *y_predicted*, *X=None*)

Returns a numerical score indicating performance based on the differences between the predicted and actual values.

Parameters

- **y_predicted** (*pd.Series*) – Predicted values of length [n_samples]
- **y_true** (*pd.Series*) – Actual class labels of length [n_samples]
- **X** (*pd.DataFrame or np.array*) – Extra data of shape [n_samples, n_features] necessary to calculate score

Returns score

evalml.objectives.PrecisionMicro.validate_inputs

PrecisionMicro.**validate_inputs**(*y_true*, *y_predicted*)

Validates the input based on a few simple checks.

Parameters

- **y_predicted** (*pd.Series*) – Predicted values of length [n_samples]
- **y_true** (*pd.Series*) – Actual class labels of length [n_samples]

Returns None

evalml.objectives.PrecisionMacro

class evalml.objectives.**PrecisionMacro**

Precision score for multiclass classification using macro averaging.

Methods

calculate_percent_difference	Calculate the percent difference between scores.
objective_function	Computes the relative value of the provided predictions compared to the actual labels, according a specified metric
score	Returns a numerical score indicating performance based on the differences between the predicted and actual values.
validate_inputs	Validates the input based on a few simple checks.

evalml.objectives.PrecisionMacro.calculate_percent_difference

classmethod PrecisionMacro.**calculate_percent_difference**(*score*, *baseline_score*)

Calculate the percent difference between scores.

Parameters

- **score** (*float*) – A score. Output of the score method of this objective.
- **baseline_score** (*float*) – A score. Output of the score method of this objective. In practice, this is the score achieved on this objective with a baseline estimator.

Returns

The percent difference between the scores. This will be the difference normalized by the baseline score.

Return type float

evalml.objectives.PrecisionMacro.objective_function

PrecisionMacro.**objective_function**(*y_true*, *y_predicted*, *X=None*)

Computes the relative value of the provided predictions compared to the actual labels, according a specified metric

Arguments: y_predicted (pd.Series): Predicted values of length [n_samples] y_true (pd.Series): Actual class labels of length [n_samples] X (pd.DataFrame or np.array): Extra data of shape [n_samples, n_features] necessary to calculate score

Returns Numerical value used to calculate score

evalml.objectives.PrecisionMacro.score

PrecisionMacro.**score**(*y_true*, *y_predicted*, *X=None*)

Returns a numerical score indicating performance based on the differences between the predicted and actual values.

Parameters

- **y_predicted** (*pd.Series*) – Predicted values of length [n_samples]
- **y_true** (*pd.Series*) – Actual class labels of length [n_samples]

- **X** (*pd.DataFrame or np.array*) – Extra data of shape [n_samples, n_features] necessary to calculate score

Returns score

evalml.objectives.PrecisionMacro.validate_inputs

PrecisionMacro.**validate_inputs**(*y_true*, *y_predicted*)

Validates the input based on a few simple checks.

Parameters

- **y_predicted** (*pd.Series*) – Predicted values of length [n_samples]
- **y_true** (*pd.Series*) – Actual class labels of length [n_samples]

Returns None

evalml.objectives.PrecisionWeighted

class evalml.objectives.**PrecisionWeighted**

Precision score for multiclass classification using weighted averaging.

Methods

calculate_percent_difference	Calculate the percent difference between scores.
objective_function	Computes the relative value of the provided predictions compared to the actual labels, according a specified metric
score	Returns a numerical score indicating performance based on the differences between the predicted and actual values.
validate_inputs	Validates the input based on a few simple checks.

evalml.objectives.PrecisionWeighted.calculate_percent_difference

classmethod PrecisionWeighted.**calculate_percent_difference**(*score*, *baseline_score*)

Calculate the percent difference between scores.

Parameters

- **score** (*float*) – A score. Output of the score method of this objective.
- **baseline_score** (*float*) – A score. Output of the score method of this objective. In practice, this is the score achieved on this objective with a baseline estimator.

Returns

The percent difference between the scores. This will be the difference normalized by the

baseline score.

Return type float

evalml.objectives.PrecisionWeighted.objective_function

PrecisionWeighted.**objective_function**(*y_true*, *y_predicted*, *X=None*)

Computes the relative value of the provided predictions compared to the actual labels, according a specified metric

Arguments: y_predicted (pd.Series): Predicted values of length [n_samples] y_true (pd.Series): Actual class labels of length [n_samples] X (pd.DataFrame or np.array): Extra data of shape [n_samples, n_features] necessary to calculate score

Returns Numerical value used to calculate score

evalml.objectives.PrecisionWeighted.score

PrecisionWeighted.**score**(*y_true*, *y_predicted*, *X=None*)

Returns a numerical score indicating performance based on the differences between the predicted and actual values.

Parameters

- **y_predicted** (*pd.Series*) – Predicted values of length [n_samples]
- **y_true** (*pd.Series*) – Actual class labels of length [n_samples]
- **X** (*pd.DataFrame or np.array*) – Extra data of shape [n_samples, n_features] necessary to calculate score

Returns score

evalml.objectives.PrecisionWeighted.validate_inputs

PrecisionWeighted.**validate_inputs**(*y_true*, *y_predicted*)

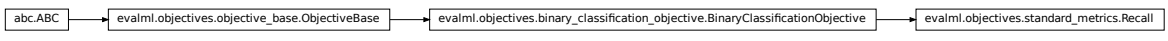
Validates the input based on a few simple checks.

Parameters

- **y_predicted** (*pd.Series*) – Predicted values of length [n_samples]
- **y_true** (*pd.Series*) – Actual class labels of length [n_samples]

Returns None

evalml.objectives.Recall



class evalml.objectives.**Recall**

Recall score for binary classification.

Methods

calculate_percent_difference	Calculate the percent difference between scores.
decision_function	Apply a learned threshold to predicted probabilities to get predicted classes.
objective_function	Computes the relative value of the provided predictions compared to the actual labels, according a specified metric
optimize_threshold	Learn a binary classification threshold which optimizes the current objective.
score	Returns a numerical score indicating performance based on the differences between the predicted and actual values.
validate_inputs	Validates the input based on a few simple checks.

evalml.objectives.Recall.calculate_percent_difference

classmethod Recall.**calculate_percent_difference**(*score*, *baseline_score*)

Calculate the percent difference between scores.

Parameters

- **score** (*float*) – A score. Output of the score method of this objective.
- **baseline_score** (*float*) – A score. Output of the score method of this objective. In practice, this is the score achieved on this objective with a baseline estimator.

Returns

The percent difference between the scores. This will be the difference normalized by the baseline score.

Return type float

evalml.objectives.Recall.decision_function

Recall.**decision_function**(*ypred_proba*, *threshold=0.5*, *X=None*)

Apply a learned threshold to predicted probabilities to get predicted classes.

Parameters

- **ypred_proba** (*list*) – The classifier's predicted probabilities
- **threshold** (*float, optional*) – Threshold used to make a prediction. Defaults to 0.5.
- **X** (*pd.DataFrame, optional*) – Any extra columns that are needed from training data.

Returns predictions

evalml.objectives.Recall.objective_function

Recall.**objective_function**(*y_true*, *y_predicted*, *X=None*)

Computes the relative value of the provided predictions compared to the actual labels, according a specified metric

Arguments: y_predicted (pd.Series): Predicted values of length [n_samples] y_true (pd.Series): Actual class labels of length [n_samples] X (pd.DataFrame or np.array): Extra data of shape [n_samples, n_features] necessary to calculate score

Returns Numerical value used to calculate score

evalml.objectives.Recall.optimize_threshold

Recall.**optimize_threshold**(*ypred_proba*, *y_true*, *X=None*)

Learn a binary classification threshold which optimizes the current objective.

Parameters

- **ypred_proba** (*list*) – The classifier's predicted probabilities
- **y_true** (*list*) – The ground truth for the predictions.
- **X** (*pd.DataFrame, optional*) – Any extra columns that are needed from training data.

Returns Optimal threshold for this objective

evalml.objectives.Recall.score

Recall.**score**(*y_true*, *y_predicted*, *X=None*)

Returns a numerical score indicating performance based on the differences between the predicted and actual values.

Parameters

- **y_predicted** (*pd.Series*) – Predicted values of length [n_samples]
- **y_true** (*pd.Series*) – Actual class labels of length [n_samples]
- **X** (*pd.DataFrame or np.array*) – Extra data of shape [n_samples, n_features] necessary to calculate score

Returns score

evalml.objectives.Recall.validate_inputs

Recall.**validate_inputs**(*y_true*, *y_predicted*)

Validates the input based on a few simple checks.

Parameters

- **y_predicted** (*pd.Series*) – Predicted values of length [n_samples]
- **y_true** (*pd.Series*) – Actual class labels of length [n_samples]

Returns None

evalml.objectives.RecallMicro

class evalml.objectives.**RecallMicro**

Recall score for multiclass classification using micro averaging.

Methods

calculate_percent_difference	Calculate the percent difference between scores.
objective_function	Computes the relative value of the provided predictions compared to the actual labels, according a specified metric
score	Returns a numerical score indicating performance based on the differences between the predicted and actual values.
validate_inputs	Validates the input based on a few simple checks.

evalml.objectives.RecallMicro.calculate_percent_difference

classmethod RecallMicro.**calculate_percent_difference**(*score*, *baseline_score*)

Calculate the percent difference between scores.

Parameters

- **score** (*float*) – A score. Output of the score method of this objective.
- **baseline_score** (*float*) – A score. Output of the score method of this objective. In practice, this is the score achieved on this objective with a baseline estimator.

Returns

The percent difference between the scores. This will be the difference normalized by the baseline score.

Return type float

evalml.objectives.RecallMicro.objective_function

RecallMicro.**objective_function**(*y_true*, *y_predicted*, *X=None*)

Computes the relative value of the provided predictions compared to the actual labels, according a specified metric

Arguments: y_predicted (pd.Series): Predicted values of length [n_samples] y_true (pd.Series): Actual class labels of length [n_samples] X (pd.DataFrame or np.array): Extra data of shape [n_samples, n_features] necessary to calculate score

Returns Numerical value used to calculate score

evalml.objectives.RecallMicro.score

RecallMicro.**score**(*y_true*, *y_predicted*, *X=None*)

Returns a numerical score indicating performance based on the differences between the predicted and actual values.

Parameters

- **y_predicted** (*pd.Series*) – Predicted values of length [n_samples]
- **y_true** (*pd.Series*) – Actual class labels of length [n_samples]
- **X** (*pd.DataFrame or np.array*) – Extra data of shape [n_samples, n_features] necessary to calculate score

Returns score

evalml.objectives.RecallMicro.validate_inputs

RecallMicro.**validate_inputs**(*y_true*, *y_predicted*)

Validates the input based on a few simple checks.

Parameters

- **y_predicted** (*pd.Series*) – Predicted values of length [n_samples]
- **y_true** (*pd.Series*) – Actual class labels of length [n_samples]

Returns None

evalml.objectives.RecallMacro

class evalml.objectives.**RecallMacro**

Recall score for multiclass classification using macro averaging.

Methods

calculate_percent_difference	Calculate the percent difference between scores.
objective_function	Computes the relative value of the provided predictions compared to the actual labels, according a specified metric
score	Returns a numerical score indicating performance based on the differences between the predicted and actual values.
validate_inputs	Validates the input based on a few simple checks.

evalml.objectives.RecallMacro.calculate_percent_difference

classmethod RecallMacro.**calculate_percent_difference**(*score*, *baseline_score*)

Calculate the percent difference between scores.

Parameters

- **score** (*float*) – A score. Output of the score method of this objective.
- **baseline_score** (*float*) – A score. Output of the score method of this objective. In practice, this is the score achieved on this objective with a baseline estimator.

Returns

The percent difference between the scores. This will be the difference normalized by the baseline score.

Return type float

evalml.objectives.RecallMacro.objective_function

RecallMacro.**objective_function**(*y_true*, *y_predicted*, *X=None*)

Computes the relative value of the provided predictions compared to the actual labels, according a specified metric

Arguments: y_predicted (pd.Series): Predicted values of length [n_samples] y_true (pd.Series): Actual class labels of length [n_samples] X (pd.DataFrame or np.array): Extra data of shape [n_samples, n_features] necessary to calculate score

Returns Numerical value used to calculate score

evalml.objectives.RecallMacro.score

RecallMacro.**score**(*y_true*, *y_predicted*, *X=None*)

Returns a numerical score indicating performance based on the differences between the predicted and actual values.

Parameters

- **y_predicted** (*pd.Series*) – Predicted values of length [n_samples]
- **y_true** (*pd.Series*) – Actual class labels of length [n_samples]
- **X** (*pd.DataFrame or np.array*) – Extra data of shape [n_samples, n_features] necessary to calculate score

Returns score

evalml.objectives.RecallMacro.validate_inputs

RecallMacro.**validate_inputs**(*y_true*, *y_predicted*)

Validates the input based on a few simple checks.

Parameters

- **y_predicted** (*pd.Series*) – Predicted values of length [n_samples]
- **y_true** (*pd.Series*) – Actual class labels of length [n_samples]

Returns None

evalml.objectives.RecallWeighted

class evalml.objectives.**RecallWeighted**

Recall score for multiclass classification using weighted averaging.

Methods

calculate_percent_difference	Calculate the percent difference between scores.
objective_function	Computes the relative value of the provided predictions compared to the actual labels, according a specified metric
score	Returns a numerical score indicating performance based on the differences between the predicted and actual values.
validate_inputs	Validates the input based on a few simple checks.

evalml.objectives.RecallWeighted.calculate_percent_difference

classmethod RecallWeighted.**calculate_percent_difference**(*score*, *baseline_score*)

Calculate the percent difference between scores.

Parameters

- **score** (*float*) – A score. Output of the score method of this objective.
- **baseline_score** (*float*) – A score. Output of the score method of this objective. In practice, this is the score achieved on this objective with a baseline estimator.

Returns

The percent difference between the scores. This will be the difference normalized by the baseline score.

Return type float

evalml.objectives.RecallWeighted.objective_function

RecallWeighted.**objective_function**(*y_true*, *y_predicted*, *X=None*)

Computes the relative value of the provided predictions compared to the actual labels, according a specified metric

Arguments: y_predicted (pd.Series): Predicted values of length [n_samples] y_true (pd.Series): Actual class labels of length [n_samples] X (pd.DataFrame or np.array): Extra data of shape [n_samples, n_features] necessary to calculate score

Returns Numerical value used to calculate score

evalml.objectives.RecallWeighted.score

RecallWeighted.**score**(*y_true*, *y_predicted*, *X=None*)

Returns a numerical score indicating performance based on the differences between the predicted and actual values.

Parameters

- **y_predicted** (*pd.Series*) – Predicted values of length [n_samples]
- **y_true** (*pd.Series*) – Actual class labels of length [n_samples]
- **X** (*pd.DataFrame or np.array*) – Extra data of shape [n_samples, n_features] necessary to calculate score

Returns score

evalml.objectives.RecallWeighted.validate_inputs

RecallWeighted.**validate_inputs**(*y_true*, *y_predicted*)

Validates the input based on a few simple checks.

Parameters

- **y_predicted** (*pd.Series*) – Predicted values of length [n_samples]
- **y_true** (*pd.Series*) – Actual class labels of length [n_samples]

Returns None

5.7.4 Regression Objectives

R2	Coefficient of determination for regression.
MAE	Mean absolute error for regression.
MSE	Mean squared error for regression.
MeanSquaredLogError	Mean squared log error for regression.
MedianAE	Median absolute error for regression.
MaxError	Maximum residual error for regression.
ExpVariance	Explained variance score for regression.
RootMeanSquaredError	Root mean squared error for regression.
RootMeanSquaredLogError	Root mean squared log error for regression.

evalml.objectives.R2

class evalml.objectives.**R2**

Coefficient of determination for regression.

Methods

calculate_percent_difference	Calculate the percent difference between scores.
objective_function	Computes the relative value of the provided predictions compared to the actual labels, according a specified metric
score	Returns a numerical score indicating performance based on the differences between the predicted and actual values.
validate_inputs	Validates the input based on a few simple checks.

evalml.objectives.R2.calculate_percent_difference

classmethod R2.**calculate_percent_difference**(*score*, *baseline_score*)

Calculate the percent difference between scores.

Parameters

- **score** (*float*) – A score. Output of the score method of this objective.
- **baseline_score** (*float*) – A score. Output of the score method of this objective. In practice, this is the score achieved on this objective with a baseline estimator.

Returns

The percent difference between the scores. This will be the difference normalized by the baseline score.

Return type float

evalml.objectives.R2.objective_function

R2.**objective_function**(*y_true*, *y_predicted*, *X=None*)

Computes the relative value of the provided predictions compared to the actual labels, according a specified metric

Arguments: y_predicted (pd.Series): Predicted values of length [n_samples] y_true (pd.Series): Actual class labels of length [n_samples] X (pd.DataFrame or np.array): Extra data of shape [n_samples, n_features] necessary to calculate score

Returns Numerical value used to calculate score

evalml.objectives.R2.score

R2.**score**(*y_true*, *y_predicted*, *X=None*)

Returns a numerical score indicating performance based on the differences between the predicted and actual values.

Parameters

- **y_predicted** (*pd.Series*) – Predicted values of length [n_samples]
- **y_true** (*pd.Series*) – Actual class labels of length [n_samples]

- **X** (*pd.DataFrame or np.array*) – Extra data of shape [n_samples, n_features] necessary to calculate score

Returns score

evalml.objectives.R2.validate_inputs

R2.**validate_inputs**(*y_true*, *y_predicted*)
Validates the input based on a few simple checks.

Parameters

- **y_predicted** (*pd.Series*) – Predicted values of length [n_samples]
- **y_true** (*pd.Series*) – Actual class labels of length [n_samples]

Returns None

evalml.objectives.MAE

class evalml.objectives.**MAE**
Mean absolute error for regression.

Methods

calculate_percent_difference	Calculate the percent difference between scores.
objective_function	Computes the relative value of the provided predictions compared to the actual labels, according a specified metric
score	Returns a numerical score indicating performance based on the differences between the predicted and actual values.
validate_inputs	Validates the input based on a few simple checks.

evalml.objectives.MAE.calculate_percent_difference

classmethod MAE.**calculate_percent_difference**(*score*, *baseline_score*)
Calculate the percent difference between scores.

Parameters

- **score** (*float*) – A score. Output of the score method of this objective.
- **baseline_score** (*float*) – A score. Output of the score method of this objective. In practice, this is the score achieved on this objective with a baseline estimator.

Returns

The percent difference between the scores. This will be the difference normalized by the

baseline score.

Return type float

evalml.objectives.MAE.objective_function

MAE.**objective_function**(*y_true*, *y_predicted*, *X=None*)

Computes the relative value of the provided predictions compared to the actual labels, according a specified metric

Arguments: y_predicted (pd.Series): Predicted values of length [n_samples] y_true (pd.Series): Actual class labels of length [n_samples] X (pd.DataFrame or np.array): Extra data of shape [n_samples, n_features] necessary to calculate score

Returns Numerical value used to calculate score

evalml.objectives.MAE.score

MAE.**score**(*y_true*, *y_predicted*, *X=None*)

Returns a numerical score indicating performance based on the differences between the predicted and actual values.

Parameters

- **y_predicted** (*pd.Series*) – Predicted values of length [n_samples]
- **y_true** (*pd.Series*) – Actual class labels of length [n_samples]
- **X** (*pd.DataFrame or np.array*) – Extra data of shape [n_samples, n_features] necessary to calculate score

Returns score

evalml.objectives.MAE.validate_inputs

MAE.**validate_inputs**(*y_true*, *y_predicted*)

Validates the input based on a few simple checks.

Parameters

- **y_predicted** (*pd.Series*) – Predicted values of length [n_samples]
- **y_true** (*pd.Series*) – Actual class labels of length [n_samples]

Returns None

evalml.objectives.MSE

class evalml.objectives.**MSE**

Mean squared error for regression.

Methods

calculate_percent_difference	Calculate the percent difference between scores.
objective_function	Computes the relative value of the provided predictions compared to the actual labels, according a specified metric
score	Returns a numerical score indicating performance based on the differences between the predicted and actual values.
validate_inputs	Validates the input based on a few simple checks.

evalml.objectives.MSE.calculate_percent_difference

classmethod MSE.**calculate_percent_difference**(*score*, *baseline_score*)

Calculate the percent difference between scores.

Parameters

- **score** (*float*) – A score. Output of the score method of this objective.
- **baseline_score** (*float*) – A score. Output of the score method of this objective. In practice, this is the score achieved on this objective with a baseline estimator.

Returns

The percent difference between the scores. This will be the difference normalized by the baseline score.

Return type float

evalml.objectives.MSE.objective_function

MSE.**objective_function**(*y_true*, *y_predicted*, *X=None*)

Computes the relative value of the provided predictions compared to the actual labels, according a specified metric

Arguments: y_predicted (pd.Series): Predicted values of length [n_samples] y_true (pd.Series): Actual class labels of length [n_samples] X (pd.DataFrame or np.array): Extra data of shape [n_samples, n_features] necessary to calculate score

Returns Numerical value used to calculate score

evalml.objectives.MSE.score

MSE.**score**(*y_true*, *y_predicted*, *X=None*)

Returns a numerical score indicating performance based on the differences between the predicted and actual values.

Parameters

- **y_predicted** (*pd.Series*) – Predicted values of length [n_samples]
- **y_true** (*pd.Series*) – Actual class labels of length [n_samples]

- **X** (*pd.DataFrame or np.array*) – Extra data of shape [n_samples, n_features] necessary to calculate score

Returns score

evalml.objectives.MSE.validate_inputs

MSE.**validate_inputs**(*y_true*, *y_predicted*)

Validates the input based on a few simple checks.

Parameters

- **y_predicted** (*pd.Series*) – Predicted values of length [n_samples]
- **y_true** (*pd.Series*) – Actual class labels of length [n_samples]

Returns None

evalml.objectives.MeanSquaredLogError

class evalml.objectives.**MeanSquaredLogError**

Mean squared log error for regression.

Only valid for nonnegative inputs. Otherwise, will throw a ValueError

Methods

calculate_percent_difference	Calculate the percent difference between scores.
objective_function	Computes the relative value of the provided predictions compared to the actual labels, according a specified metric
score	Returns a numerical score indicating performance based on the differences between the predicted and actual values.
validate_inputs	Validates the input based on a few simple checks.

evalml.objectives.MeanSquaredLogError.calculate_percent_difference

classmethod MeanSquaredLogError.**calculate_percent_difference**(*score*, *baseline_score*)

Calculate the percent difference between scores.

Parameters

- **score** (*float*) – A score. Output of the score method of this objective.
- **baseline_score** (*float*) – A score. Output of the score method of this objective. In practice, this is the score achieved on this objective with a baseline estimator.

Returns

The percent difference between the scores. This will be the difference normalized by the baseline score.

Return type float

evalml.objectives.MeanSquaredLogError.objective_function

MeanSquaredLogError.**objective_function**(*y_true*, *y_predicted*, *X=None*)

Computes the relative value of the provided predictions compared to the actual labels, according a specified metric

Arguments: y_predicted (pd.Series): Predicted values of length [n_samples] y_true (pd.Series): Actual class labels of length [n_samples] X (pd.DataFrame or np.array): Extra data of shape [n_samples, n_features] necessary to calculate score

Returns Numerical value used to calculate score

evalml.objectives.MeanSquaredLogError.score

MeanSquaredLogError.**score**(*y_true*, *y_predicted*, *X=None*)

Returns a numerical score indicating performance based on the differences between the predicted and actual values.

Parameters

- **y_predicted** (*pd.Series*) – Predicted values of length [n_samples]
- **y_true** (*pd.Series*) – Actual class labels of length [n_samples]
- **X** (*pd.DataFrame or np.array*) – Extra data of shape [n_samples, n_features] necessary to calculate score

Returns score

evalml.objectives.MeanSquaredLogError.validate_inputs

MeanSquaredLogError.**validate_inputs**(*y_true*, *y_predicted*)

Validates the input based on a few simple checks.

Parameters

- **y_predicted** (*pd.Series*) – Predicted values of length [n_samples]
- **y_true** (*pd.Series*) – Actual class labels of length [n_samples]

Returns None

evalml.objectives.MedianAE

class evalml.objectives.**MedianAE**

Median absolute error for regression.

Methods

calculate_percent_difference	Calculate the percent difference between scores.
objective_function	Computes the relative value of the provided predictions compared to the actual labels, according a specified metric
score	Returns a numerical score indicating performance based on the differences between the predicted and actual values.
validate_inputs	Validates the input based on a few simple checks.

evalml.objectives.MedianAE.calculate_percent_difference

classmethod MedianAE.**calculate_percent_difference**(*score*, *baseline_score*)

Calculate the percent difference between scores.

Parameters

- **score** (*float*) – A score. Output of the score method of this objective.
- **baseline_score** (*float*) – A score. Output of the score method of this objective. In practice, this is the score achieved on this objective with a baseline estimator.

Returns

The percent difference between the scores. This will be the difference normalized by the baseline score.

Return type float

evalml.objectives.MedianAE.objective_function

MedianAE.**objective_function**(*y_true*, *y_predicted*, *X=None*)

Computes the relative value of the provided predictions compared to the actual labels, according a specified metric

Arguments: y_predicted (pd.Series): Predicted values of length [n_samples] y_true (pd.Series): Actual class labels of length [n_samples] X (pd.DataFrame or np.array): Extra data of shape [n_samples, n_features] necessary to calculate score

Returns Numerical value used to calculate score

evalml.objectives.MedianAE.score

MedianAE.**score**(*y_true*, *y_predicted*, *X=None*)

Returns a numerical score indicating performance based on the differences between the predicted and actual values.

Parameters

- **y_predicted** (*pd.Series*) – Predicted values of length [n_samples]

- **y_true** (*pd.Series*) – Actual class labels of length [n_samples]
- **X** (*pd.DataFrame or np.array*) – Extra data of shape [n_samples, n_features] necessary to calculate score

Returns score

evalml.objectives.MedianAE.validate_inputs

MedianAE.**validate_inputs**(*y_true*, *y_predicted*)
Validates the input based on a few simple checks.

Parameters

- **y_predicted** (*pd.Series*) – Predicted values of length [n_samples]
- **y_true** (*pd.Series*) – Actual class labels of length [n_samples]

Returns None

evalml.objectives.MaxError

class evalml.objectives.**MaxError**
Maximum residual error for regression.

Methods

calculate_percent_difference	Calculate the percent difference between scores.
objective_function	Computes the relative value of the provided predictions compared to the actual labels, according a specified metric
score	Returns a numerical score indicating performance based on the differences between the predicted and actual values.
validate_inputs	Validates the input based on a few simple checks.

evalml.objectives.MaxError.calculate_percent_difference

classmethod MaxError.**calculate_percent_difference**(*score*, *baseline_score*)
Calculate the percent difference between scores.

Parameters

- **score** (*float*) – A score. Output of the score method of this objective.
- **baseline_score** (*float*) – A score. Output of the score method of this objective. In practice, this is the score achieved on this objective with a baseline estimator.

Returns

The percent difference between the scores. This will be the difference normalized by the baseline score.

Return type float

evalml.objectives.MaxError.objective_function

MaxError.**objective_function**(*y_true*, *y_predicted*, *X=None*)

Computes the relative value of the provided predictions compared to the actual labels, according a specified metric

Arguments: y_predicted (pd.Series): Predicted values of length [n_samples] y_true (pd.Series): Actual class labels of length [n_samples] X (pd.DataFrame or np.array): Extra data of shape [n_samples, n_features] necessary to calculate score

Returns Numerical value used to calculate score

evalml.objectives.MaxError.score

MaxError.**score**(*y_true*, *y_predicted*, *X=None*)

Returns a numerical score indicating performance based on the differences between the predicted and actual values.

Parameters

- **y_predicted** (*pd.Series*) – Predicted values of length [n_samples]
- **y_true** (*pd.Series*) – Actual class labels of length [n_samples]
- **X** (*pd.DataFrame or np.array*) – Extra data of shape [n_samples, n_features] necessary to calculate score

Returns score

evalml.objectives.MaxError.validate_inputs

MaxError.**validate_inputs**(*y_true*, *y_predicted*)

Validates the input based on a few simple checks.

Parameters

- **y_predicted** (*pd.Series*) – Predicted values of length [n_samples]
- **y_true** (*pd.Series*) – Actual class labels of length [n_samples]

Returns None

evalml.objectives.ExpVariance

class evalml.objectives.**ExpVariance**

Explained variance score for regression.

Methods

calculate_percent_difference	Calculate the percent difference between scores.
objective_function	Computes the relative value of the provided predictions compared to the actual labels, according a specified metric
score	Returns a numerical score indicating performance based on the differences between the predicted and actual values.
validate_inputs	Validates the input based on a few simple checks.

evalml.objectives.ExpVariance.calculate_percent_difference

classmethod ExpVariance.**calculate_percent_difference**(*score*, *baseline_score*)

Calculate the percent difference between scores.

Parameters

- **score** (*float*) – A score. Output of the score method of this objective.
- **baseline_score** (*float*) – A score. Output of the score method of this objective. In practice, this is the score achieved on this objective with a baseline estimator.

Returns

The percent difference between the scores. This will be the difference normalized by the baseline score.

Return type float

evalml.objectives.ExpVariance.objective_function

ExpVariance.**objective_function**(*y_true*, *y_predicted*, *X=None*)

Computes the relative value of the provided predictions compared to the actual labels, according a specified metric

Arguments: y_predicted (pd.Series): Predicted values of length [n_samples] y_true (pd.Series): Actual class labels of length [n_samples] X (pd.DataFrame or np.array): Extra data of shape [n_samples, n_features] necessary to calculate score

Returns Numerical value used to calculate score

evalml.objectives.ExpVariance.score

ExpVariance.**score**(*y_true*, *y_predicted*, *X=None*)

Returns a numerical score indicating performance based on the differences between the predicted and actual values.

Parameters

- **y_predicted** (*pd.Series*) – Predicted values of length [n_samples]

- **y_true** (*pd.Series*) – Actual class labels of length [n_samples]
- **X** (*pd.DataFrame or np.array*) – Extra data of shape [n_samples, n_features] necessary to calculate score

Returns score

evalml.objectives.ExpVariance.validate_inputs

ExpVariance.**validate_inputs**(*y_true*, *y_predicted*)

Validates the input based on a few simple checks.

Parameters

- **y_predicted** (*pd.Series*) – Predicted values of length [n_samples]
- **y_true** (*pd.Series*) – Actual class labels of length [n_samples]

Returns None

evalml.objectives.RootMeanSquaredError

class evalml.objectives.**RootMeanSquaredError**

Root mean squared error for regression.

Methods

calculate_percent_difference	Calculate the percent difference between scores.
objective_function	Computes the relative value of the provided predictions compared to the actual labels, according a specified metric
score	Returns a numerical score indicating performance based on the differences between the predicted and actual values.
validate_inputs	Validates the input based on a few simple checks.

evalml.objectives.RootMeanSquaredError.calculate_percent_difference

classmethod RootMeanSquaredError.**calculate_percent_difference**(*score*, *baseline_score*)

Calculate the percent difference between scores.

Parameters

- **score** (*float*) – A score. Output of the score method of this objective.
- **baseline_score** (*float*) – A score. Output of the score method of this objective. In practice, this is the score achieved on this objective with a baseline estimator.

Returns

The percent difference between the scores. This will be the difference normalized by the baseline score.

Return type float

evalml.objectives.RootMeanSquaredError.objective_function

RootMeanSquaredError.**objective_function**(*y_true*, *y_predicted*, *X=None*)

Computes the relative value of the provided predictions compared to the actual labels, according a specified metric

Arguments: y_predicted (pd.Series): Predicted values of length [n_samples] y_true (pd.Series): Actual class labels of length [n_samples] X (pd.DataFrame or np.array): Extra data of shape [n_samples, n_features] necessary to calculate score

Returns Numerical value used to calculate score

evalml.objectives.RootMeanSquaredError.score

RootMeanSquaredError.**score**(*y_true*, *y_predicted*, *X=None*)

Returns a numerical score indicating performance based on the differences between the predicted and actual values.

Parameters

- **y_predicted** (*pd.Series*) – Predicted values of length [n_samples]
- **y_true** (*pd.Series*) – Actual class labels of length [n_samples]
- **X** (*pd.DataFrame or np.array*) – Extra data of shape [n_samples, n_features] necessary to calculate score

Returns score

evalml.objectives.RootMeanSquaredError.validate_inputs

RootMeanSquaredError.**validate_inputs**(*y_true*, *y_predicted*)

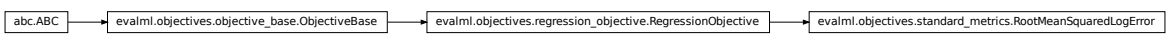
Validates the input based on a few simple checks.

Parameters

- **y_predicted** (*pd.Series*) – Predicted values of length [n_samples]
- **y_true** (*pd.Series*) – Actual class labels of length [n_samples]

Returns None

evalml.objectives.RootMeanSquaredLogError



class evalml.objectives.**RootMeanSquaredLogError**

Root mean squared log error for regression.

Only valid for nonnegative inputs.Otherwise, will throw a ValueError.

Methods

calculate_percent_difference	Calculate the percent difference between scores.
objective_function	Computes the relative value of the provided predictions compared to the actual labels, according a specified metric
score	Returns a numerical score indicating performance based on the differences between the predicted and actual values.
validate_inputs	Validates the input based on a few simple checks.

evalml.objectives.RootMeanSquaredLogError.calculate_percent_difference

classmethod RootMeanSquaredLogError.**calculate_percent_difference**(*score*, *baseline_score*)

Calculate the percent difference between scores.

Parameters

- **score** (*float*) – A score. Output of the score method of this objective.
- **baseline_score** (*float*) – A score. Output of the score method of this objective. In practice, this is the score achieved on this objective with a baseline estimator.

Returns

The percent difference between the scores. This will be the difference normalized by the baseline score.

Return type float

evalml.objectives.RootMeanSquaredLogError.objective_function

RootMeanSquaredLogError.**objective_function**(*y_true*, *y_predicted*, *X=None*)

Computes the relative value of the provided predictions compared to the actual labels, according a specified metric

Arguments: y_predicted (pd.Series): Predicted values of length [n_samples] y_true (pd.Series): Actual class labels of length [n_samples] X (pd.DataFrame or np.array): Extra data of shape [n_samples, n_features] necessary to calculate score

Returns Numerical value used to calculate score

evalml.objectives.RootMeanSquaredLogError.score

RootMeanSquaredLogError.**score**(*y_true*, *y_predicted*, *X=None*)

Returns a numerical score indicating performance based on the differences between the predicted and actual values.

Parameters

- **y_predicted** (*pd.Series*) – Predicted values of length [n_samples]
- **y_true** (*pd.Series*) – Actual class labels of length [n_samples]
- **X** (*pd.DataFrame or np.array*) – Extra data of shape [n_samples, n_features] necessary to calculate score

Returns score

evalml.objectives.RootMeanSquaredLogError.validate_inputs

RootMeanSquaredLogError.**validate_inputs**(*y_true*, *y_predicted*)

Validates the input based on a few simple checks.

Parameters

- **y_predicted** (*pd.Series*) – Predicted values of length [n_samples]
- **y_true** (*pd.Series*) – Actual class labels of length [n_samples]

Returns None

5.7.5 Objective Utils

get_all_objective_names	Get a list of the names of all objectives.
get_core_objectives	Returns all core objective instances associated with the given problem type.
get_core_objective_names	Get a list of all valid core objectives.
get_non_core_objectives	Get non-core objective classes.
get_objective	Returns the Objective class corresponding to a given objective name.

evalml.objectives.get_all_objective_names

evalml.objectives.**get_all_objective_names**()

Get a list of the names of all objectives.

Returns Objective names

Return type list (str)

evalml.objectives.get_core_objectives

evalml.objectives.**get_core_objectives**(*problem_type*)

Returns all core objective instances associated with the given problem type.

Core objectives are designed to work out-of-the-box for any dataset.

Parameters **problem_type** (*str/ProblemTypes*) – Type of problem

Returns List of ObjectiveBase instances

evalml.objectives.get_core_objective_names

evalml.objectives.**get_core_objective_names**()
Get a list of all valid core objectives.

Returns Objective names.

Return type list(str)

evalml.objectives.get_non_core_objectives

evalml.objectives.**get_non_core_objectives**()
Get non-core objective classes.

Non-core objectives are objectives that are domain-specific. Users typically need to configure these objectives before using them in AutoMLSearch.

Returns List of ObjectiveBase classes

evalml.objectives.get_objective

evalml.objectives.**get_objective**(*objective*, *return_instance=False*, ***kwargs*)
Returns the Objective class corresponding to a given objective name.

Parameters

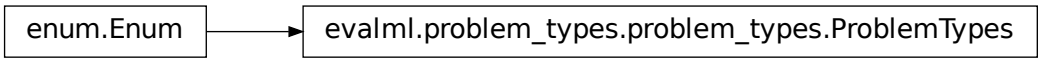
- **objective** (*str or ObjectiveBase*) – Name or instance of the objective class.
- **return_instance** (*bool*) – Whether to return an instance of the objective. This only applies if objective is of type str. Note that the instance will be initialized with default arguments.
- **kwargs** (*Any*) – Any keyword arguments to pass into the objective. Only used when return_instance=True.

Returns ObjectiveBase if the parameter objective is of type ObjectiveBase. If objective is instead a valid objective name, function will return the class corresponding to that name. If return_instance is True, an instance of that objective will be returned.

5.8 Problem Types

ProblemTypes	Enum defining the supported types of machine learning problems.

5.8.1 evalml.problem_types.ProblemTypes



class evalml.problem_types.**ProblemTypes**
Enum defining the supported types of machine learning problems.

handle_problem_types	Handles problem_type by either returning the ProblemTypes or converting from a str.
detect_problem_type	Determine the type of problem is being solved based on the targets (binary vs multiclass classification, regression)

5.8.2 evalml.problem_types.handle_problem_types

evalml.problem_types.**handle_problem_types**(*problem_type*)
Handles problem_type by either returning the ProblemTypes or converting from a str.

Parameters **problem_type** (*str or ProblemTypes*) – Problem type that needs to be handled

Returns ProblemTypes

5.8.3 evalml.problem_types.detect_problem_type

evalml.problem_types.**detect_problem_type**(*y*)

Determine the type of problem is being solved based on the targets (binary vs multiclass classification, regression)
Ignores missing and null data

Parameters **y** (*pd.Series*) – the target labels to predict

Returns ProblemType Enum

Return type ProblemType

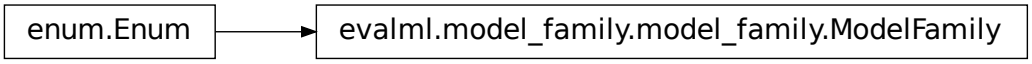
Example

```
>>> y = pd.Series([0, 1, 0, 0, 0, 1, 1, 0, 0, 1, 1])
>>> problem_type = detect_problem_type(y)
>>> assert problem_type == ProblemTypes.BINARY
```

5.9 Model Family

ModelFamily	Enum for family of machine learning models.

5.9.1 evalml.model_family.ModelFamily



class evalml.model_family.**ModelFamily**
Enum for family of machine learning models.

handle_model_family	Handles model_family by either returning the ModelFamily or converting from a string

5.9.2 evalml.model_family.handle_model_family

evalml.model_family.**handle_model_family**(*model_family*)
Handles model_family by either returning the ModelFamily or converting from a string

Parameters **model_family** (*str or ModelFamily*) – Model type that needs to be handled

Returns ModelFamily

5.10 Tuners

Tuner	Defines API for Tuners.
SKOptTuner	Bayesian Optimizer.
GridSearchTuner	Grid Search Optimizer.
RandomSearchTuner	Random Search Optimizer.

5.10.1 evalml.tuners.Tuner

class evalml.tuners.**Tuner**(*pipeline_hyperparameter_ranges*, *random_state=0*)

Defines API for Tuners.

Tuners implement different strategies for sampling from a search space. They're used in EvalML to search the space of pipeline hyperparameters.

Methods

__init__	Base Tuner class
add	Register a set of hyperparameters with the score obtained from training a pipeline with those hyperparameters.
is_search_space_exhausted	Optional.
propose	Returns a suggested set of parameters to train and score a pipeline with, based off the search space dimensions and prior samples.

evalml.tuners.Tuner.__init__

Tuner.**__init__**(*pipeline_hyperparameter_ranges*, *random_state=0*)

Base Tuner class

Parameters

- **pipeline_hyperparameter_ranges** (*dict*) – a set of hyperparameter ranges corresponding to a pipeline's parameters
- **random_state** (*int, np.random.RandomState*) – The random state

evalml.tuners.Tuner.add

Tuner.**add**(*pipeline_parameters*, *score*)

Register a set of hyperparameters with the score obtained from training a pipeline with those hyperparameters.

Parameters

- **pipeline_parameters** (*dict*) – a dict of the parameters used to evaluate a pipeline

- **score** (*float*) – the score obtained by evaluating the pipeline with the provided parameters

Returns None

evalml.tuners.Tuner.is_search_space_exhausted

Tuner.**is_search_space_exhausted**()
Optional. If possible search space for tuner is finite, this method indicates whether or not all possible parameters have been scored.

Returns Returns true if all possible parameters in a search space has been scored.

Return type bool

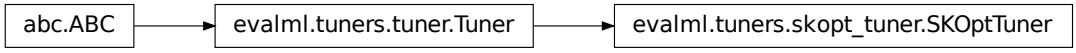
evalml.tuners.Tuner.propose

Tuner.**propose**()
Returns a suggested set of parameters to train and score a pipeline with, based off the search space dimensions and prior samples.

Returns proposed pipeline parameters

Return type dict

5.10.2 evalml.tuners.SKOptTuner



class evalml.tuners.**SKOptTuner**(*pipeline_hyperparameter_ranges*, *random_state=0*)
Bayesian Optimizer.

Methods

__init__	Init SkOptTuner
add	Add score to sample
is_search_space_exhausted	Optional.
propose	Returns a suggested set of parameters to train and score a pipeline with, based off the search space dimensions and prior samples.

evalml.tuners.SKOptTuner.__init__

SKOptTuner.**__init__**(*pipeline_hyperparameter_ranges*, *random_state=0*)
Init SkOptTuner

Parameters

- **pipeline_hyperparameter_ranges** (*dict*) – a set of hyperparameter ranges corresponding to a pipeline's parameters
- **random_state** (*int, np.random.RandomState*) – The random state

evalml.tuners.SKOptTuner.add

SKOptTuner.**add**(*pipeline_parameters*, *score*)

Add score to sample

Parameters

- **pipeline_parameters** (*dict*) – a dict of the parameters used to evaluate a pipeline
- **score** (*float*) – the score obtained by evaluating the pipeline with the provided parameters

Returns None

evalml.tuners.SKOptTuner.is_search_space_exhausted

SKOptTuner.**is_search_space_exhausted**()

Optional. If possible search space for tuner is finite, this method indicates whether or not all possible parameters have been scored.

Returns Returns true if all possible parameters in a search space has been scored.

Return type bool

evalml.tuners.SKOptTuner.propose

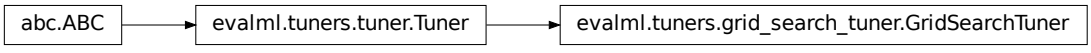
SKOptTuner.**propose**()

Returns a suggested set of parameters to train and score a pipeline with, based off the search space dimensions and prior samples.

Returns proposed pipeline parameters

Return type dict

5.10.3 evalml.tuners.GridSearchTuner



class evalml.tuners.**GridSearchTuner**(*pipeline_hyperparameter_ranges*, *n_points=10*, *random_state=0*)

Grid Search Optimizer.

Example

```
>>> tuner = GridSearchTuner({'My Component': {'param a': [0.0, 10.0], 'param b': [
↪'a', 'b', 'c']}}, n_points=5)
>>> proposal = tuner.propose()
>>> assert proposal.keys() == {'My Component'}
>>> assert proposal['My Component'] == {'param a': 0.0, 'param b': 'a'}
```

Methods

__init__	Generate all of the possible points to search for in the grid
add	Not applicable to grid search tuner as generated parameters are not dependent on scores of previous parameters.
is_search_space_exhausted	Checks if it is possible to generate a set of valid parameters.
propose	Returns parameters from _grid_points iterations

evalml.tuners.GridSearchTuner.__init__

GridSearchTuner.**__init__**(*pipeline_hyperparameter_ranges*, *n_points=10*, *random_state=0*)

Generate all of the possible points to search for in the grid

Parameters

- **pipeline_hyperparameter_ranges** (*dict*) – a set of hyperparameter ranges corresponding to a pipeline's parameters
- **n_points** – The number of points to sample from along each dimension defined in the `space` argument
- **random_state** – Unused in this class

evalml.tuners.GridSearchTuner.add

GridSearchTuner.**add**(*pipeline_parameters*, *score*)

Not applicable to grid search tuner as generated parameters are not dependent on scores of previous parameters.

Parameters

- **pipeline_parameters** (*dict*) – a dict of the parameters used to evaluate a pipeline
- **score** (*float*) – the score obtained by evaluating the pipeline with the provided parameters

evalml.tuners.GridSearchTuner.is_search_space_exhausted

GridSearchTuner.**is_search_space_exhausted**()

Checks if it is possible to generate a set of valid parameters. Stores generated parameters in `self.curr_params` to be returned by `propose()`.

Raises **NoParamsException** – If a search space is exhausted, then this exception is thrown.

Returns If no more valid parameters exists in the search space, return false.

Return type bool

evalml.tuners.GridSearchTuner.propose

GridSearchTuner.**propose**()

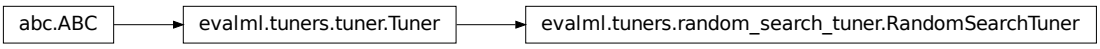
Returns parameters from _grid_points iterations

If all possible combinations of parameters have been scored, then NoParamsException is raised.

Returns proposed pipeline parameters

Return type dict

5.10.4 evalml.tuners.RandomSearchTuner



class evalml.tuners.**RandomSearchTuner**(*pipeline_hyperparameter_ranges*, *random_state=0*, *with_replacement=False*, *replacement_max_attempts=10*)

Random Search Optimizer.

Example

```
>>> tuner = RandomSearchTuner({'My Component': {'param a': [0.0, 10.0], 'param b
↪': ['a', 'b', 'c']}}, random_state=42)
>>> proposal = tuner.propose()
>>> assert proposal.keys() == {'My Component'}
>>> assert proposal['My Component'] == {'param a': 3.7454011884736254, 'param b':
↪'c'}
```

Methods

__init__	Sets up check for duplication if needed.
add	Not applicable to random search tuner as generated parameters are not dependent on scores of previous parameters.
is_search_space_exhausted	Checks if it is possible to generate a set of valid parameters.
propose	Generate a unique set of parameters.

evalml.tuners.RandomSearchTuner.__init__

RandomSearchTuner.**__init__**(*pipeline_hyperparameter_ranges*, *random_state=0*, *with_replacement=False*, *replacement_max_attempts=10*)

Sets up check for duplication if needed.

Parameters

- **pipeline_hyperparameter_ranges** (*dict*) – a set of hyperparameter ranges corresponding to a pipeline's parameters
- **random_state** – Unused in this class
- **with_replacement** – If false, only unique hyperparameters will be shown
- **replacement_max_attempts** – The maximum number of tries to get a unique set of random parameters. Only used if tuner is initalized with with_replacement=True

evalml.tuners.RandomSearchTuner.add

RandomSearchTuner.**add**(*pipeline_parameters*, *score*)

Not applicable to random search tuner as generated parameters are not dependent on scores of previous parameters.

Parameters

- **pipeline_parameters** (*dict*) – a dict of the parameters used to evaluate a pipeline
- **score** (*float*) – the score obtained by evaluating the pipeline with the provided parameters

evalml.tuners.RandomSearchTuner.is_search_space_exhausted

RandomSearchTuner.**is_search_space_exhausted**()

Checks if it is possible to generate a set of valid parameters. Stores generated parameters in `self.curr_params` to be returned by `propose()`.

Raises **NoParamsException** – If a search space is exhausted, then this exception is thrown.

Returns If no more valid parameters exists in the search space, return false.

Return type bool

evalml.tuners.RandomSearchTuner.propose

RandomSearchTuner.**propose**()

Generate a unique set of parameters.

If tuner was initialized with `with_replacement=True` and the tuner is unable to generate a unique set of parameters after `replacement_max_attempts` tries, then `NoParamsException` is raised.

Returns proposed pipeline parameters

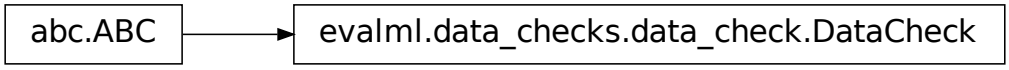
Return type dict

5.11 Data Checks

5.11.1 Data Check Classes

DataCheck	Base class for all data checks.
InvalidTargetDataCheck	Checks if the target data contains missing or invalid values.
HighlyNullDataCheck	Checks if there are any highly-null columns in the input.
IDColumnsDataCheck	Check if any of the features are likely to be ID columns.
LabelLeakageDataCheck	Check if any of the features are highly correlated with the target.
OutliersDataCheck	Checks if there are any outliers in input data by using an Isolation Forest to obtain the anomaly score of each index and then using IQR to determine score anomalies.
NoVarianceDataCheck	Check if the target or any of the features have no variance.
ClassImbalanceDataCheck	Checks if any target labels are imbalanced beyond a threshold.

evalml.data_checks.DataCheck



class evalml.data_checks.**DataCheck**

Base class for all data checks. Data checks are a set of heuristics used to determine if there are problems with input data.

name = 'DataCheck'

Instance attributes

—

Methods:

validate	Inspects and validates the input data, runs any necessary calculations or algorithms, and returns a list of warnings and errors if applicable.

evalml.data_checks.DataCheck.validate

DataCheck.**validate**(*X*, *y=None*)

Inspects and validates the input data, runs any necessary calculations or algorithms, and returns a list of warnings and errors if applicable.

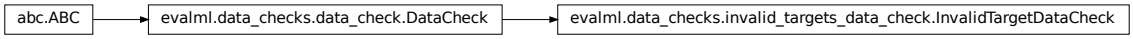
Parameters

- **X** (*pd.DataFrame*) – the input data of shape [n_samples, n_features]
- **y** (*pd.Series, optional*) – the target data of length [n_samples]

Returns list of DataCheckError and DataCheckWarning objects

Return type list (*DataCheckMessage*)

evalml.data_checks.InvalidTargetDataCheck



class evalml.data_checks.**InvalidTargetDataCheck**(*problem_type*)

Checks if the target data contains missing or invalid values.

name = 'InvalidTargetDataCheck'

Instance attributes

—

Methods:

__init__	Initialize self.
validate	Checks if the target data contains missing or invalid values.

evalml.data_checks.InvalidTargetDataCheck.__init__

InvalidTargetDataCheck.**__init__**(*problem_type*)

Initialize self. See help(type(self)) for accurate signature.

evalml.data_checks.InvalidTargetDataCheck.validate

InvalidTargetDataCheck.**validate**(*X*, *y*)

Checks if the target data contains missing or invalid values.

Parameters

- **X** (*pd.DataFrame, pd.Series, np.array, list*) – Features. Ignored.

- **y** – Target data to check for invalid values.

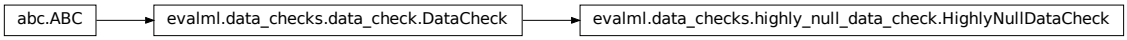
Returns List with DataCheckErrors if any invalid values are found in the target data.

Return type list (*DataCheckError*)

Example

```
>>> X = pd.DataFrame({})
>>> y = pd.Series([0, 1, None, None])
>>> target_check = InvalidTargetDataCheck('binary')
>>> assert target_check.validate(X, y) == [DataCheckError("2 row(s) (50.0%) of target values are null", "InvalidTargetDataCheck")]
```

evalml.data_checks.HighlyNullDataCheck



class evalml.data_checks.**HighlyNullDataCheck**(*pct_null_threshold=0.95*)

Checks if there are any highly-null columns in the input.

name = 'HighlyNullDataCheck'

Instance attributes

—

Methods:

__init__	Checks if there are any highly-null columns in the input.
validate	Checks if there are any highly-null columns in the input.

evalml.data_checks.HighlyNullDataCheck.__init__

HighlyNullDataCheck.**__init__**(*pct_null_threshold=0.95*)

Checks if there are any highly-null columns in the input.

Parameters **pct_null_threshold** (*float*) – If the percentage of NaN values in an input feature exceeds this amount, that feature will be considered highly-null. Defaults to 0.95.

evalml.data_checks.HighlyNullDataCheck.validate

HighlyNullDataCheck.**validate**(*X*, *y=None*)

Checks if there are any highly-null columns in the input.

Parameters

- **X** (*pd.DataFrame, pd.Series, np.array, list*) – Features
- **y** – Ignored.

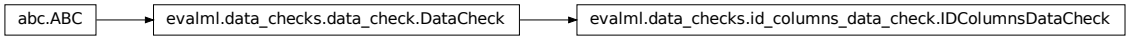
Returns List with a DataCheckWarning if there are any highly-null columns.

Return type list (*DataCheckWarning*)

Example

```
>>> df = pd.DataFrame({
...     'lots_of_null': [None, None, None, None, 5],
...     'no_null': [1, 2, 3, 4, 5]
... })
>>> null_check = HighlyNullDataCheck(pct_null_threshold=0.8)
>>> assert null_check.validate(df) == [DataCheckWarning("Column 'lots_of_null
↪' is 80.0% or more null", "HighlyNullDataCheck")]
```

evalml.data_checks.IDColumnsDataCheck



class evalml.data_checks.**IDColumnsDataCheck**(*id_threshold=1.0*)

Check if any of the features are likely to be ID columns.

name = 'IDColumnsDataCheck'

Instance attributes

—

Methods:

__init__	Check if any of the features are likely to be ID columns.
validate	Check if any of the features are likely to be ID columns.

evalml.data_checks.IDColumnsDataCheck.__init__

IDColumnsDataCheck.**__init__**(*id_threshold=1.0*)

Check if any of the features are likely to be ID columns.

Parameters **id_threshold** (*float*) – The probability threshold to be considered an ID column. Defaults to 1.0.

evalml.data_checks.IDColumnsDataCheck.validate

IDColumnsDataCheck.**validate**(*X*, *y=None*)

Check if any of the features are likely to be ID columns. Currently performs these simple checks:

- column name is "id"
- column name ends in "_id"
- column contains all unique values (and is not float / boolean)

Parameters

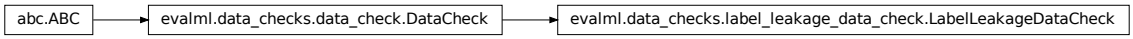
- **X** (*pd.DataFrame*) – The input features to check
- **threshold** (*float*) – The probability threshold to be considered an ID column. Defaults to 1.0

Returns A dictionary of features with column name or index and their probability of being ID columns

Example

```
>>> df = pd.DataFrame({
...     'df_id': [0, 1, 2, 3, 4],
...     'x': [10, 42, 31, 51, 61],
...     'y': [42, 54, 12, 64, 12]
... })
>>> id_col_check = IDColumnsDataCheck()
>>> assert id_col_check.validate(df) == [DataCheckWarning("Column 'df_id' is␣
↪100.0% or more likely to be an ID column", "IDColumnsDataCheck")]
```

evalml.data_checks.LabelLeakageDataCheck



class evalml.data_checks.**LabelLeakageDataCheck**(*pct_corr_threshold=0.95*)

Check if any of the features are highly correlated with the target.

name = 'LabelLeakageDataCheck'

Instance attributes

—

Methods:

__init__	Check if any of the features are highly correlated with the target.
validate	Check if any of the features are highly correlated with the target.

evalml.data_checks.LabelLeakageDataCheck.__init__

LabelLeakageDataCheck.**__init__**(*pct_corr_threshold=0.95*)

Check if any of the features are highly correlated with the target.

Currently only supports binary and numeric targets and features.

Parameters **pct_corr_threshold** (*float*) – The correlation threshold to be considered leakage. Defaults to 0.95.

evalml.data_checks.LabelLeakageDataCheck.validate

LabelLeakageDataCheck.**validate**(*X*, *y*)

Check if any of the features are highly correlated with the target.

Currently only supports binary and numeric targets and features.

Parameters

- **X** (*pd.DataFrame*) – The input features to check
- **y** (*pd.Series*) – The target data

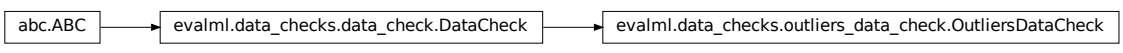
Returns List with a DataCheckWarning if there is label leakage detected.

Return type list (*DataCheckWarning*)

Example

```
>>> X = pd.DataFrame({
...     'leak': [10, 42, 31, 51, 61],
...     'x': [42, 54, 12, 64, 12],
...     'y': [12, 5, 13, 74, 24],
... })
>>> y = pd.Series([10, 42, 31, 51, 40])
>>> label_leakage_check = LabelLeakageDataCheck(pct_corr_threshold=0.8)
>>> assert label_leakage_check.validate(X, y) == [DataCheckWarning("Column
↪'leak' is 80.0% or more correlated with the target", "LabelLeakageDataCheck
↪")]
```

evalml.data_checks.OutliersDataCheck



class evalml.data_checks.**OutliersDataCheck**(*random_state=0*)

Checks if there are any outliers in input data by using an Isolation Forest to obtain the anomaly score of each index and then using IQR to determine score anomalies. Indices with score anomalies are considered outliers.

name = 'OutliersDataCheck'

Instance attributes

—

Methods:

__init__	Checks if there are any outliers in the input data.
validate	Checks if there are any outliers in a dataframe by using an Isolation Forest to obtain the anomaly score of each index and then using IQR to determine score anomalies.

evalml.data_checks.OutliersDataCheck.__init__

OutliersDataCheck.**__init__**(*random_state=0*)

Checks if there are any outliers in the input data.

Parameters **random_state** (*int, np.random.RandomState*) – The random seed/state. Defaults to 0.

evalml.data_checks.OutliersDataCheck.validate

OutliersDataCheck.**validate**(*X*, *y=None*)

Checks if there are any outliers in a dataframe by using an Isolation Forest to obtain the anomaly score of each index and then using IQR to determine score anomalies. Indices with score anomalies are considered outliers.

Parameters

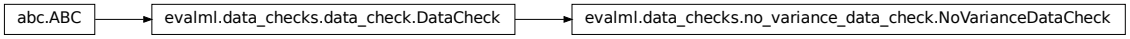
- **X** (*pd.DataFrame*) – Features
- **y** – Ignored.

Returns A set of indices that may have outlier data.

Example

```
>>> df = pd.DataFrame({
...     'x': [1, 2, 3, 40, 5],
...     'y': [6, 7, 8, 990, 10],
...     'z': [-1, -2, -3, -1201, -4]
... })
>>> outliers_check = OutliersDataCheck()
>>> assert outliers_check.validate(df) == [DataCheckWarning("Row '3' is␣
↪likely to have outlier data", "OutliersDataCheck")]
```

evalml.data_checks.NoVarianceDataCheck



class evalml.data_checks.**NoVarianceDataCheck**(*count_nan_as_value=False*)

Check if the target or any of the features have no variance.

name = 'NoVarianceDataCheck'

Instance attributes

—

Methods:

__init__	Check if the target or any of the features have no variance.
validate	Check if the target or any of the features have no variance (1 unique value).

evalml.data_checks.NoVarianceDataCheck.__init__

NoVarianceDataCheck.**__init__**(*count_nan_as_value=False*)

Check if the target or any of the features have no variance.

Parameters **count_nan_as_value** (*bool*) – If True, missing values will be counted as their own unique value. If set to True, a feature that has one unique value and all other data is missing, a DataCheckWarning will be returned instead of an error. Defaults to False.

evalml.data_checks.NoVarianceDataCheck.validate

NoVarianceDataCheck.**validate**(*X*, *y*)

Check if the target or any of the features have no variance (1 unique value).

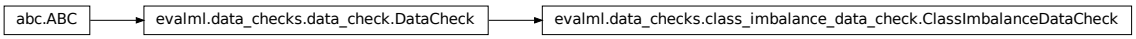
Parameters

- **X** (*pd.DataFrame*) – The input features.
- **y** (*pd.Series*) – The target data.

Returns List of warnings/errors corresponding to features or target with no variance.

Return type list (*DataCheckWarning* or *DataCheckError*)

evalml.data_checks.ClassImbalanceDataCheck



class evalml.data_checks.**ClassImbalanceDataCheck**(*threshold=0.1*)

Checks if any target labels are imbalanced beyond a threshold. Use for classification problems

name = 'ClassImbalanceDataCheck'

Instance attributes

—

Methods:

__init__	Check if any of the features are likely to be ID columns.
validate	Checks if any target labels are imbalanced beyond a threshold for binary and multiclass problems Ignores nan values in target labels if they appear

evalml.data_checks.ClassImbalanceDataCheck.__init__

ClassImbalanceDataCheck.**__init__**(*threshold=0.1*)

Check if any of the features are likely to be ID columns.

Parameters **threshold** (*float*) – The minimum threshold allowed for class imbalance before a warning is raised. A perfectly balanced dataset would have a threshold of (1/n_classes), ie 0.50 for binary classes. Defaults to 0.10

evalml.data_checks.ClassImbalanceDataCheck.validate

ClassImbalanceDataCheck.**validate**(*X*, *y*)

Checks if any target labels are imbalanced beyond a threshold for binary and multiclass problems Ignores nan values in target labels if they appear

Parameters

- **X** (*pd.DataFrame, pd.Series, np.array, list*) – Features. Ignored.
- **y** – Target labels to check for imbalanced data.

Returns list with DataCheckWarnings if imbalance in classes is less than the threshold.

Return type list (*DataCheckWarning*)

Example

```
>>> X = pd.DataFrame({})
>>> y = pd.Series([0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1])
>>> target_check = ClassImbalanceDataCheck(threshold=0.10)
>>> assert target_check.validate(X, y) == [DataCheckWarning("The following␣
↪labels fall below 10% of the target: [0]", "ClassImbalanceDataCheck")]
```

DataChecks	A collection of data checks.
DefaultDataChecks	A collection of basic data checks that is used by AutoML by default.

evalml.data_checks.DataChecks

evalml.data_checks.data_checks.DataChecks

class evalml.data_checks.**DataChecks**(*data_checks=None*, *data_check_params=None*)

A collection of data checks.

Methods

__init__	A collection of data checks.
validate	Inspects and validates the input data against data checks and returns a list of warnings and errors if applicable.

evalml.data_checks.DataChecks.__init__

DataChecks.**__init__**(*data_checks=None*, *data_check_params=None*)

A collection of data checks.

Parameters **data_checks** (*list (DataCheck)*) – List of DataCheck objects

evalml.data_checks.DataChecks.validate

DataChecks.**validate**(*X*, *y=None*)

Inspects and validates the input data against data checks and returns a list of warnings and errors if applicable.

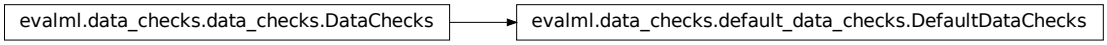
Parameters

- **X** (*pd.DataFrame*) – The input data of shape [n_samples, n_features]
- **y** (*pd.Series*) – The target data of length [n_samples]

Returns List containing DataCheckMessage objects

Return type list (*DataCheckMessage*)

evalml.data_checks.DefaultDataChecks



class evalml.data_checks.**DefaultDataChecks**(*problem_type*)

A collection of basic data checks that is used by AutoML by default. Includes HighlyNullDataCheck, IDColumnsDataCheck, LabelLeakageDataCheck, InvalidTargetDataCheck, and NoVarianceDataCheck.

Methods

__init__	A collection of basic data checks.
validate	Inspects and validates the input data against data checks and returns a list of warnings and errors if applicable.

evalml.data_checks.DefaultDataChecks.__init__

DefaultDataChecks.**__init__**(*problem_type*)

A collection of basic data checks. :param problem_type: The problem type that is being validated. Can be regression, binary, or multiclass. :type problem_type: str

evalml.data_checks.DefaultDataChecks.validate

DefaultDataChecks.**validate**(*X*, *y=None*)

Inspects and validates the input data against data checks and returns a list of warnings and errors if applicable.

Parameters

- **X** (*pd.DataFrame*) – The input data of shape [n_samples, n_features]
- **y** (*pd.Series*) – The target data of length [n_samples]

Returns List containing DataCheckMessage objects

Return type list (*DataCheckMessage*)

5.11.2 Data Check Messages

DataCheckMessage	Base class for all DataCheckMessages.
DataCheckError	DataCheckMessage subclass for errors returned by data checks.
DataCheckWarning	DataCheckMessage subclass for warnings returned by data checks.

evalml.data_checks.DataCheckMessage

evalml.data_checks.data_check_message.DataCheckMessage

class evalml.data_checks.**DataCheckMessage**(*message*, *data_check_name*)

Base class for all DataCheckMessages.

message_type = None

Methods:

__init__	Message returned by a DataCheck, tagged by name."
__str__	String representation of data check message, equivalent to self.message attribute.
__eq__	Checks for equality.

evalml.data_checks.DataCheckMessage.__init__

DataCheckMessage.**__init__**(*message*, *data_check_name*)

Message returned by a DataCheck, tagged by name."

Parameters

- **message** (*str*) – Message string
- **data_check_name** (*str*) – Name of data check

evalml.data_checks.DataCheckMessage.__str__

DataCheckMessage.**__str__**()

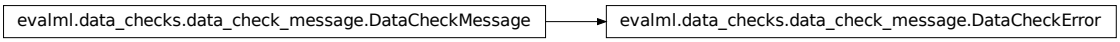
String representation of data check message, equivalent to self.message attribute.

evalml.data_checks.DataCheckMessage.__eq__

DataCheckMessage.**__eq__**(*other*)

Checks for equality. Two DataCheckMessage objs are considered equivalent if their message type and message are equivalent.

evalml.data_checks.DataCheckError



class evalml.data_checks.**DataCheckError**(*message*, *data_check_name*)

DataCheckMessage subclass for errors returned by data checks.

message_type = 'error'

Methods:

__init__	Message returned by a DataCheck, tagged by name.”
__str__	String representation of data check message, equivalent to self.message attribute.
__eq__	Checks for equality.

evalml.data_checks.DataCheckError.__init__

DataCheckError.**__init__**(*message*, *data_check_name*)

Message returned by a DataCheck, tagged by name.”

Parameters

- **message** (*str*) – Message string
- **data_check_name** (*str*) – Name of data check

evalml.data_checks.DataCheckError.__str__

DataCheckError.**__str__**()

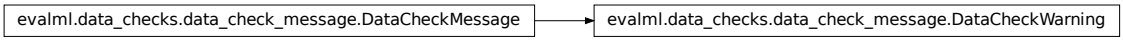
String representation of data check message, equivalent to self.message attribute.

evalml.data_checks.DataCheckError.__eq__

DataCheckError.**__eq__**(*other*)

Checks for equality. Two DataCheckMessage objs are considered equivalent if their message type and message are equivalent.

evalml.data_checks.DataCheckWarning



class evalml.data_checks.**DataCheckWarning**(*message*, *data_check_name*)

DataCheckMessage subclass for warnings returned by data checks.

message_type = 'warning'

Methods:

__init__	Message returned by a DataCheck, tagged by name."
__str__	String representation of data check message, equivalent to self.message attribute.
__eq__	Checks for equality.

evalml.data_checks.DataCheckWarning.__init__

DataCheckWarning.**__init__**(*message*, *data_check_name*)

Message returned by a DataCheck, tagged by name."

Parameters

- **message** (*str*) – Message string
- **data_check_name** (*str*) – Name of data check

evalml.data_checks.DataCheckWarning.__str__

DataCheckWarning.**__str__**()

String representation of data check message, equivalent to self.message attribute.

evalml.data_checks.DataCheckWarning.__eq__

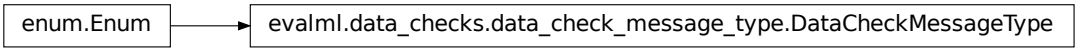
DataCheckWarning.**__eq__**(*other*)

Checks for equality. Two DataCheckMessage objs are considered equivalent if their message type and message are equivalent.

5.11.3 Data Check Message Types

DataCheckMessageType	Enum for type of data check message: WARNING or ERROR.

evalml.data_checks.DataCheckMessageType



class evalml.data_checks.**DataCheckMessageType**
Enum for type of data check message: WARNING or ERROR.

5.12 Utils

5.12.1 General Utils

import_or_raise	Attempts to import the requested library by name.
convert_to_seconds	Converts a string describing a length of time to its length in seconds.
get_random_state	Generates a numpy.random.RandomState instance using seed.
get_random_seed	Given a numpy.random.RandomState object, generate an int representing a seed value for another random number generator.

evalml.utils.import_or_raise

evalml.utils.**import_or_raise**(*library*, *error_msg=None*, *warning=False*)
Attempts to import the requested library by name. If the import fails, raises an ImportError or warning.

Parameters

- **library** (*str*) – the name of the library
- **error_msg** (*str*) – error message to return if the import fails
- **warning** (*bool*) – if True, import_or_raise gives a warning instead of ImportError. Defaults to False.

evalml.utils.convert_to_seconds

evalml.utils.**convert_to_seconds**(*input_str*)
Converts a string describing a length of time to its length in seconds.

evalml.utils.get_random_state

evalml.utils.**get_random_state**(*seed*)
Generates a numpy.random.RandomState instance using seed.

Parameters **seed** (*None, int, np.random.RandomState object*) – seed to use to generate numpy.random.RandomState. Must be between SEED_BOUNDS.min_bound and SEED_BOUNDS.max_bound, inclusive. Otherwise, an exception will be thrown.

evalml.utils.get_random_seed

evalml.utils.**get_random_seed**(*random_state*, *min_bound=0*, *max_bound=2147483647*)

Given a numpy.random.RandomState object, generate an int representing a seed value for another random number generator. Or, if given an int, return that int.

To protect against invalid input to a particular library's random number generator, if an int value is provided, and it is outside the bounds "[min_bound, max_bound)", the value will be projected into the range between the min_bound (inclusive) and max_bound (exclusive) using modular arithmetic.

Parameters

- **random_state** (*int, numpy.random.RandomState*) – random state
- **min_bound** (*None, int*) – if not default of None, will be min bound when generating seed (inclusive). Must be less than max_bound.
- **max_bound** (*None, int*) – if not default of None, will be max bound when generating seed (exclusive). Must be greater than min_bound.

Returns seed for random number generator

Return type int

CHAPTER

SIX

RELEASE NOTES

Future Releases

- Enhancements
- Fixes
- Changes
- Documentation Changes
- Testing Changes

v0.14.1 Sep. 29, 2020

- **Enhancements**
  - Updated partial dependence methods to support calculating numeric columns in a dataset with non-numeric columns #1150
  - Added *get_feature_names* on *OneHotEncoder* #1193
  - Added *detect_problem_type* to *problem_type/utils.py* to automatically detect the problem type given targets #1194
  - Added LightGBM to AutoMLSearch #1199
  - Updates scikit-learn and scikit-optimize to use latest versions - 0.23.2 and 0.8.1 respectively #1141
  - Added *__str__* and *__repr__* for pipelines and components #1218
  - Included internal target check for both training and validation data in AutoMLSearch #1226
  - Add *ProblemTypes.all_problem_types* helper to get list of supported problem types #1219
  - Added *DecisionTreeClassifier* and *DecisionTreeRegressor* classes #1223
  - Added *ProblemTypes.all_problem_types* helper to get list of supported problem types #1219
  - *DataChecks* can now be parametrized by passing a list of *DataCheck* classes and a parameter dictionary #1167
  - Added first CV fold score as validation score in *AutoMLSearch.rankings* #1221
  - Updated *flake8* configuration to enable linting on *__init__.py* files #1234
  - Refined *make_pipeline_from_components* implementation #1204
- **Fixes**
  - Updated GitHub URL after migration to Alteryx GitHub org #1207
  - Changed Problem Type enum to be more similar to the string name #1208

  - Wrapped call to scikit-learn's partial dependence method in a *try/finally* block #1232

- **Changes**
  - Added *allow_writing_files* as a named argument to CatBoost estimators. #1202
  - Added *solver* and *multi_class* as named arguments to LogisticRegressionClassifier #1202
  - Replaced pipeline's *._transform* method to evaluate all the preprocessing steps of a pipeline with *.compute_estimator_features* #1231
  - Changed default large dataset train/test splitting behavior #1205
- **Documentation Changes**
  - Included description of how to access the component instances and features for pipeline user guide #1163
  - Updated API docs to refer to target as "target" instead of "labels" for non-classification tasks and minor docs cleanup #1160
  - Added Class Imbalance Data Check to *api_reference.rst* #1190 #1200
  - Added pipeline properties to API reference #1209
  - Clarified what the objective parameter in AutoML is used for in AutoML API reference and AutoML user guide #1222
  - Updated API docs to include *skopt.space.Categorical* option for component hyperparameter range definition #1228
  - Added install documentation for *libomp* in order to use LightGBM on Mac #1233
  - Improved description of *max_iterations* in documentation #1212
  - Removed unused code from sphinx conf #1235
- Testing Changes

Warning:

Breaking Changes

- *DefaultDataChecks* now accepts a *problem_type* parameter that must be specified #1167
- Pipeline's *._transform* method to evaluate all the preprocessing steps of a pipeline has been replaced with *.compute_estimator_features* #1231
- *get_objectives* has been renamed to *get_core_objectives*. This function will now return a list of valid objective instances #1230

v0.13.2 Sep. 17, 2020

- **Enhancements**
  - Added *output_format* field to explain predictions functions #1107
  - Modified *get_objective* and *get_objectives* to be able to return any objective in *evalml.objectives* #1132
  - Added a *return_instance* boolean parameter to *get_objective* #1132
  - Added *ClassImbalanceDataCheck* to determine whether target imbalance falls below a given threshold #1135
  - Added label encoder to lightGBM for binary classification #1152

  - Added labels for the row index of confusion matrix #1154
  - Added AutoMLSearch object as another parameter in search callbacks #1156
  - Added the corresponding probability threshold for each point displayed in *graph_roc_curve* #1161
  - Added *__eq__* for *ComponentBase* and *PipelineBase* #1178
  - Added support for multiclass classification for *roc_curve* #1164
  - Added *categories* accessor to *OneHotEncoder* for listing the categories associated with a feature #1182
  - Added utility function to create pipeline instances from a list of component instances #1176
- **Fixes**
  - Fixed XGBoost column names for partial dependence methods #1104
  - Removed dead code validating column type from *TextFeaturizer* #1122
  - Fixed issue where Imputer cannot fit when there is None in a categorical or boolean column #1144
  - OneHotEncoder preserves the custom index in the input data #1146
  - Fixed representation for *ModelFamily* #1165
  - Removed duplicate *nbsphinx* dependency in *dev-requirements.txt* #1168
  - Users can now pass in any valid kwargs to all estimators #1157
  - Remove broken accessor *OneHotEncoder.get_feature_names* and unneeded base class #1179
  - Removed LightGBM Estimator from AutoML models #1186
- **Changes**
  - Pinned scikit-optimize version to 0.7.4 #1136
  - Removed tqdm as a dependency #1177
  - Added lightgbm version 3.0.0 to latest_dependency_versions.txt #1185
  - Rename *max_pipelines* to *max_iterations* #1169
- **Documentation Changes**
  - Fixed API docs for *AutoMLSearch add_result_callback* #1113
  - Added a step to our release process for pushing our latest version to conda-forge #1118
  - Added warning for missing ipywidgets dependency for using *PipelineSearchPlots* on Jupyterlab #1145
  - Updated README.md example to load demo dataset #1151
  - Swapped mapping of breast cancer targets in *model_understanding.ipynb* #1170
- **Testing Changes**
  - Added test confirming *TextFeaturizer* never outputs null values #1122
  - Changed Python version of *Update Dependencies* action to 3.8.x #1137
  - Fixed release notes check-in test for *Update Dependencies* actions #1172

Warning:

Breaking Changes

- *get_objective* will now return a class definition rather than an instance by default #1132
- Deleted *OPTIONS* dictionary in *evalml.objectives.utils.py* #1132
- If specifying an objective by string, the string must now match the objective's name field, case-insensitive #1132
- **Passing "Cost Benefit Matrix", "Fraud Cost", "Lead Scoring", "Mean Squared Log Error",** "Recall", "Recall Macro", "Recall Micro", "Recall Weighted", or "Root Mean Squared Log Error" to *AutoMLSearch* will now result in a *ValueError* rather than an *ObjectiveNotFoundError* #1132
- Search callbacks *start_iteration_callback* and *add_results_callback* have changed to include a copy of the AutoMLSearch object as a third parameter #1156
- Deleted *OneHotEncoder.get_feature_names* method which had been broken for a while, in favor of pipelines' *input_feature_names* #1179
- Deleted empty base class *CategoricalEncoder* which *OneHotEncoder* component was inheriting from #1176
- Results from *roc_curve* will now return as a list of dictionaries with each dictionary representing a class #1164
- *max_pipelines* now raises a *DeprecationWarning' and will be removed in the next release. 'max_iterations* should be used instead. #1169

v0.13.1 Aug. 25, 2020

- **Enhancements**
  - Added Cost-Benefit Matrix objective for binary classification #1038
  - Split *fill_value* into *categorical_fill_value* and *numeric_fill_value* for Imputer #1019
  - Added *explain_predictions* and *explain_predictions_best_worst* for explaining multiple predictions with SHAP #1016
  - Added new LSA component for text featurization #1022
  - Added guide on installing with conda #1041
  - Added a "cost-benefit curve" util method to graph cost-benefit matrix scores vs. binary classification thresholds #1081
  - Standardized error when calling transform/predict before fit for pipelines #1048
  - Added *percent_better_than_baseline* to Automl search rankings and full rankings table #1050
  - Added one-way partial dependence and partial dependence plots #1079
  - Added "Feature Value" column to prediction explanation reports. #1064
  - Added LightGBM classification estimator #1082, #1114
  - Added *max_batches* parameter to AutoMLSearch #1087
- **Fixes**
  - Updated TextFeaturizer component to no longer require an internet connection to run #1022
  - Fixed non-deterministic element of TextFeaturizer transformations #1022

  - Added a StandardScaler to all ElasticNet pipelines #1065
  - Updated cost-benefit matrix to normalize score #1099
  - Fixed logic in *calculate_percent_difference* so that it can handle negative values #1100
- **Changes**
  - Added *needs_fitting* property to ComponentBase #1044
  - Updated references to data types to use datatype lists defined in *evalml.utils.gen_utils* #1039
  - Remove maximum version limit for SciPy dependency #1051
  - Moved *all_components* and other component importers into runtime methods #1045
  - Consolidated graphing utility methods under *evalml.utils.graph_utils* #1060
  - Made slight tweaks to how TextFeaturizer uses featuretools, and did some refactoring of that and of LSA #1090
  - Changed *show_all_features* parameter into *importance_threshold*, which allows for thresholding feature importance #1097, #1103
- **Documentation Changes**
  - Update setup.py URL to point to the github repo #1037
  - Added tutorial for using the cost-benefit matrix objective #1088
  - Updated *model_understanding.ipynb* to include documentation for using plotly on Jupyter Lab #1108
- **Testing Changes**
  - Refactor CircleCI tests to use matrix jobs (#1043)
  - Added a test to check that all test directories are included in evalml package #1054

Warning:

Breaking Changes

- `confusion_matrix` and `normalize_confusion_matrix` have been moved to *evalml.utils* #1038
- All graph utility methods previously under `evalml.pipelines.graph_utils` have been moved to `evalml.utils.graph_utils` #1060

v0.12.2 Aug. 6, 2020

- **Enhancements**
  - Add save/load method to components #1023
  - Expose pickle *protocol* as optional arg to save/load #1023
  - Updated estimators used in AutoML to include ExtraTrees and ElasticNet estimators #1030
- Fixes
- **Changes**
  - Removed DeprecationWarning for SimpleImputer #1018
- **Documentation Changes**

  - Add note about version numbers to release process docs #1034

- **Testing Changes**

  - Test files are now included in the evalml package #1029

v0.12.0 Aug. 3, 2020

- **Enhancements**

  - Added string and categorical targets support for binary and multiclass pipelines and check for numeric targets for *DetectLabelLeakage* data check #932
  - Added clear exception for regression pipelines if target datatype is string or categorical #960
  - Added target column names and class labels in *predict* and *predict_proba* output for pipelines #951
  - Added *_compute_shap_values* and *normalize_values* to *pipelines/explanations* module #958
  - Added *explain_prediction* feature which explains single predictions with SHAP #974
  - Added Imputer to allow different imputation strategies for numerical and categorical dtypes #991
  - Added support for configuring logfile path using env var, and don't create logger if there are filesystem errors #975
  - Updated catboost estimators' default parameters and automl hyperparameter ranges to speed up fit time #998

- **Fixes**

  - Fixed ReadtheDocs warning failure regarding embedded gif #943
  - Removed incorrect parameter passed to pipeline classes in *_add_baseline_pipelines* #941
  - Added universal error for calling *predict*, *predict_proba*, *transform*, and *feature_importances* before fitting #969, #994
  - Made *TextFeaturizer* component and pip dependencies *featuretools* and *nlp_primitives* optional #976
  - Updated imputation strategy in automl to no longer limit impute strategy to *most_frequent* for all features if there are any categorical columns #991
  - Fixed UnboundLocalError for‘cv_pipeline‘ when automl search errors #996
  - Fixed *Imputer* to reset dataframe index to preserve behavior expected from *SimpleImputer* #1009

- **Changes**

  - Moved *get_estimators ‘ to ‘evalml.pipelines.components.utils* #934
  - Modified Pipelines to raise *PipelineScoreError* when they encounter an error during scoring #936
  - Moved *evalml.model_families.list_model_families* to *evalml.pipelines.components.allowed_model_families* #959
  - Renamed *DateTimeFeaturization* to *DateTimeFeaturizer* #977
  - Added check to stop search and raise an error if all pipelines in a batch return NaN scores #1015

- **Documentation Changes**

  - Update README.md #963
  - Reworded message when errors are returned from data checks in search #982
  - Added section on understanding model predictions with *explain_prediction* to User Guide #981

  - Added a section to the user guide and api reference about how XGBoost and CatBoost are not fully supported. #992
  - Added custom components section in user guide #993
  - Update FAQ section formatting #997
  - Update release process documentation #1003
- **Testing Changes**
  - Moved *predict_proba* and *predict* tests regarding string / categorical targets to *test_pipelines.py* #972
  - Fix dependency update bot by updating python version to 3.7 to avoid frequent github version updates #1002

Warning:

Breaking Changes

- `get_estimators` has been moved to `evalml.pipelines.components.utils` (previously was under `evalml.pipelines.utils`) #934
- Removed the `raise_errors` flag in AutoML search. All errors during pipeline evaluation will be caught and logged. #936
- `evalml.model_families.list_model_families` has been moved to *evalml.pipelines.components.allowed_model_families* #959
- `TextFeaturizer`: the `featuretools` and `nlp_primitives` packages must be installed after installing evalml in order to use this component #976
- Renamed `DateTimeFeaturization` to `DateTimeFeaturizer` #977

v0.11.2 July 16, 2020

- **Enhancements**
  - Added *NoVarianceDataCheck* to *DefaultDataChecks* #893
  - Added text processing and featurization component *TextFeaturizer* #913, #924
  - Added additional checks to InvalidTargetDataCheck to handle invalid target data types #929
  - AutoMLSearch will now handle KeyboardInterrupt and prompt user for confirmation #915
- **Fixes**
  - Makes automl results a read-only property #919
- **Changes**
  - Deleted static pipelines and refactored tests involving static pipelines, removed *all_pipelines()* and *get_pipelines()* #904
  - Moved *list_model_families* to *evalml.model_family.utils* #903
  - Updated *all_pipelines*, *all_estimators*, *all_components* to use the same mechanism for dynamically generating their elements #898
  - Rename *master* branch to *main* #918
  - Add pypi release github action #923
  - Updated AutoMLSearch.search stdout output and logging and removed tqdm progress bar #921

  - Moved automl config checks previously in *search()* to init #933

- **Documentation Changes**
  - Reorganized and rewrote documentation #937
  - Updated to use pydata sphinx theme #937
  - Updated docs to use *release_notes* instead of *changelog* #942
- **Testing Changes**
  - Cleaned up fixture names and usages in tests #895

Warning:

Breaking Changes

- `list_model_families` has been moved to `evalml.model_family.utils` (previously was under `evalml.pipelines.utils`) #903
- `get_estimators` has been moved to `evalml.pipelines.components.utils` (previously was under `evalml.pipelines.utils`) #934
- Static pipeline definitions have been removed, but similar pipelines can still be constructed via creating an instance of PipelineBase #904
- `all_pipelines()` and `get_pipelines()` utility methods have been removed #904

v0.11.0 June 30, 2020

- **Enhancements**
  - Added multiclass support for ROC curve graphing #832
  - Added preprocessing component to drop features whose percentage of NaN values exceeds a specified threshold #834
  - Added data check to check for problematic target labels #814
  - Added PerColumnImputer that allows imputation strategies per column #824
  - Added transformer to drop specific columns #827
  - Added support for *categories*, *handle_error*, and *drop* parameters in *OneHotEncoder* #830 #897
  - Added preprocessing component to handle DateTime columns featurization #838
  - Added ability to clone pipelines and components #842
  - Define getter method for component *parameters* #847
  - Added utility methods to calculate and graph permutation importances #860, #880
  - Added new utility functions necessary for generating dynamic preprocessing pipelines #852
  - Added kwargs to all components #863
  - Updated *AutoSearchBase* to use dynamically generated preprocessing pipelines #870
  - Added SelectColumns transformer #873
  - Added ability to evaluate additional pipelines for automl search #874
  - Added *default_parameters* class property to components and pipelines #879
  - Added better support for disabling data checks in automl search #892

  - Added ability to save and load AutoML objects to file #888
  - Updated *AutoSearchBase.get_pipelines* to return an untrained pipeline instance #876
  - Saved learned binary classification thresholds in automl results cv data dict #876
- **Fixes**
  - Fixed bug where SimpleImputer cannot handle dropped columns #846
  - Fixed bug where PerColumnImputer cannot handle dropped columns #855
  - Enforce requirement that builtin components save all inputted values in their parameters dict #847
  - Don't list base classes in *all_components* output #847
  - Standardize all components to output pandas data structures, and accept either pandas or numpy #853
  - Fixed rankings and full_rankings error when search has not been run #894
- **Changes**
  - Update *all_pipelines* and *all_components* to try initializing pipelines/components, and on failure exclude them #849
  - Refactor *handle_components* to *handle_components_class*, standardize to *ComponentBase* subclass instead of instance #850
  - Refactor "blacklist"/"whitelist" to "allow"/"exclude" lists #854
  - Replaced *AutoClassificationSearch* and *AutoRegressionSearch* with *AutoMLSearch* #871
  - Renamed feature_importances and permutation_importances methods to use singular names (feature_importance and permutation_importance) #883
  - Updated *automl* default data splitter to train/validation split for large datasets #877
  - Added open source license, update some repo metadata #887
  - Removed dead code in *_get_preprocessing_components* #896
- **Documentation Changes**
  - Fix some typos and update the EvalML logo #872
- **Testing Changes**
  - Update the changelog check job to expect the new branching pattern for the deps update bot #836
  - Check that all components output pandas datastructures, and can accept either pandas or numpy #853
  - Replaced *AutoClassificationSearch* and *AutoRegressionSearch* with *AutoMLSearch* #871

Warning:

Breaking Changes

- Pipelines' static `component_graph` field must contain either `ComponentBase` subclasses or `str`, instead of `ComponentBase` subclass instances #850
- Rename `handle_component` to `handle_component_class`. Now standardizes to `ComponentBase` subclasses instead of `ComponentBase` subclass instances #850
- Renamed automl's `cv` argument to `data_split` #877

- Pipelines' and classifiers' `feature_importances` is renamed *feature_importance*, *graph_feature_importances* is renamed *graph_feature_importance* #883
- Passing `data_checks=None` to automl search will not perform any data checks as opposed to default checks. #892
- Pipelines to search for in AutoML are now determined automatically, rather than using the statically-defined pipeline classes. #870
- Updated `AutoSearchBase.get_pipelines` to return an untrained pipeline instance, instead of one which happened to be trained on the final cross-validation fold #876

v0.10.0 May 29, 2020

- **Enhancements**
  - Added baseline models for classification and regression, add functionality to calculate baseline models before searching in AutoML #746
  - Port over highly-null guardrail as a data check and define *DefaultDataChecks* and *DisableDataChecks* classes #745
  - Update *Tuner* classes to work directly with pipeline parameters dicts instead of flat parameter lists #779
  - Add Elastic Net as a pipeline option #812
  - Added new Pipeline option *ExtraTrees* #790
  - Added precicion-recall curve metrics and plot for binary classification problems in *evalml.pipeline.graph_utils* #794
  - Update the default automl algorithm to search in batches, starting with default parameters for each pipeline and iterating from there #793
  - Added *AutoMLAlgorithm* class and *IterativeAlgorithm* impl, separated from *AutoSearchBase* #793
- **Fixes**
  - Update pipeline *score* to return *nan* score for any objective which throws an exception during scoring #787
  - Fixed bug introduced in #787 where binary classification metrics requiring predicted probabilities error in scoring #798
  - CatBoost and XGBoost classifiers and regressors can no longer have a learning rate of 0 #795
- **Changes**
  - Cleanup pipeline *score* code, and cleanup codecov #711
  - Remove *pass* for abstract methods for codecov #730
  - Added __str__ for AutoSearch object #675
  - Add util methods to graph ROC and confusion matrix #720
  - Refactor *AutoBase* to *AutoSearchBase* #758
  - Updated AutoBase with *data_checks* parameter, removed previous *detect_label_leakage* parameter, and added functionality to run data checks before search in AutoML #765
  - Updated our logger to use Python's logging utils #763

  - Refactor most of *AutoSearchBase._do_iteration* impl into *AutoSearchBase._evaluate* #762
  - Port over all guardrails to use the new DataCheck API #789
  - Expanded *import_or_raise* to catch all exceptions #759
  - Adds RMSE, MSLE, RMSLE as standard metrics #788
  - Don't allow *Recall* to be used as an objective for AutoML #784
  - Removed feature selection from pipelines #819
  - Update default estimator parameters to make automl search faster and more accurate #793
- **Documentation Changes**
  - Add instructions to freeze *master* on *release.md* #726
  - Update release instructions with more details #727 #733
  - Add objective base classes to API reference #736
  - Fix components API to match other modules #747
- **Testing Changes**
  - Delete codecov yml, use codecov.io's default #732
  - Added unit tests for fraud cost, lead scoring, and standard metric objectives #741
  - Update codecov client #782
  - Updated AutoBase __str__ test to include no parameters case #783
  - Added unit tests for *ExtraTrees* pipeline #790
  - If codecov fails to upload, fail build #810
  - Updated Python version of dependency action #816
  - Update the dependency update bot to use a suffix when creating branches #817

Warning:

Breaking Changes

- The `detect_label_leakage` parameter for AutoML classes has been removed and replaced by a `data_checks` parameter #765
- Moved ROC and confusion matrix methods from `evalml.pipeline.plot_utils` to `evalml.pipeline.graph_utils` #720
- `Tuner` classes require a pipeline hyperparameter range dict as an init arg instead of a space definition #779
- `Tuner.propose` and `Tuner.add` work directly with pipeline parameters dicts instead of flat parameter lists #779
- `PipelineBase.hyperparameters` and `custom_hyperparameters` use pipeline parameters dict format instead of being represented as a flat list #779
- All guardrail functions previously under `evalml.guardrails.utils` will be removed and replaced by data checks #789
- *Recall* disallowed as an objective for AutoML #784
- `AutoSearchBase` parameter `tuner` has been renamed to `tuner_class` #793

- `AutoSearchBase` parameter `possible_pipelines` and `possible_model_families` have been renamed to `allowed_pipelines` and `allowed_model_families` #793

v0.9.0 Apr. 27, 2020

- **Enhancements**
  - Added accuracy as an standard objective #624
  - Added verbose parameter to load_fraud #560
  - Added Balanced Accuracy metric for binary, multiclass #612 #661
  - Added XGBoost regressor and XGBoost regression pipeline #666
  - Added Accuracy metric for multiclass #672
  - Added objective name in *AutoBase.describe_pipeline* #686
  - Added *DataCheck* and *DataChecks*, *Message* classes and relevant subclasses #739
- **Fixes**
  - Removed direct access to *cls.component_graph* #595
  - Add testing files to .gitignore #625
  - Remove circular dependencies from *Makefile* #637
  - Add error case for *normalize_confusion_matrix()* #640
  - Fixed XGBoostClassifier and XGBoostRegressor bug with feature names that contain [,], or < #659
  - Update make_pipeline_graph to not accidentally create empty file when testing if path is valid #649
  - Fix pip installation warning about docsutils version, from boto dependency #664
  - Removed zero division warning for F1/precision/recall metrics #671
  - Fixed *summary* for pipelines without estimators #707
- **Changes**
  - Updated default objective for binary/multiseries classification to log loss #613
  - Created classification and regression pipeline subclasses and removed objective as an attribute of pipeline classes #405
  - Changed the output of *score* to return one dictionary #429
  - Created binary and multiclass objective subclasses #504
  - Updated objectives API #445
  - Removed call to *get_plot_data* from AutoML #615
  - Set *raise_error* to default to True for AutoML classes #638
  - Remove unnecessary "u" prefixes on some unicode strings #641
  - Changed one-hot encoder to return uint8 dtypes instead of ints #653
  - Pipeline *_name* field changed to *custom_name* #650
  - Removed *graphs.py* and moved methods into *PipelineBase* #657, #665

  - Remove s3fs as a dev dependency #664
  - Changed requirements-parser to be a core dependency #673
  - Replace *supported_problem_types* field on pipelines with *problem_type* attribute on base classes #678
  - Changed AutoML to only show best results for a given pipeline template in *rankings*, added *full_rankings* property to show all #682
  - Update *ModelFamily* values: don't list xgboost/catboost as classifiers now that we have regression pipelines for them #677
  - Changed AutoML's *describe_pipeline* to get problem type from pipeline instead #685
  - Standardize *import_or_raise* error messages #683
  - Updated argument order of objectives to align with sklearn's #698
  - Renamed *pipeline.feature_importance_graph* to *pipeline.graph_feature_importances* #700
  - Moved ROC and confusion matrix methods to *evalml.pipelines.plot_utils* #704
  - Renamed *MultiClassificationObjective* to *MulticlassClassificationObjective*, to align with pipeline naming scheme #715

- **Documentation Changes**
  - Fixed some sphinx warnings #593
  - Fixed docstring for AutoClassificationSearch with correct command #599
  - Limit readthedocs formats to pdf, not htmlzip and epub #594 #600
  - Clean up objectives API documentation #605
  - Fixed function on Exploring search results page #604
  - Update release process doc #567
  - AutoClassificationSearch and AutoRegressionSearch show inherited methods in API reference #651
  - Fixed improperly formatted code in breaking changes for changelog #655
  - Added configuration to treat Sphinx warnings as errors #660
  - Removed separate plotting section for pipelines in API reference #657, #665
  - Have leads example notebook load S3 files using https, so we can delete s3fs dev dependency #664
  - Categorized components in API reference and added descriptions for each category #663
  - Fixed Sphinx warnings about BalancedAccuracy objective #669
  - Updated API reference to include missing components and clean up pipeline docstrings #689
  - Reorganize API ref, and clarify pipeline sub-titles #688
  - Add and update preprocessing utils in API reference #687
  - Added inheritance diagrams to API reference #695
  - Documented which default objective AutoML optimizes for #699
  - Create seperate install page #701
  - Include more utils in API ref, like *import_or_raise* #704

  - Add more color to pipeline documentation #705

- **Testing Changes**
  - Matched install commands of *check_latest_dependencies* test and it's GitHub action #578
  - Added Github app to auto assign PR author as assignee #477
  - Removed unneeded conda installation of xgboost in windows checkin tests #618
  - Update graph tests to always use tmpfile dir #649
  - Changelog checkin test workaround for release PRs: If 'future release' section is empty of PR refs, pass check #658
  - Add changelog checkin test exception for *dep-update* branch #723

Warning: Breaking Changes

- Pipelines will now no longer take an objective parameter during instantiation, and will no longer have an objective attribute.
- `fit()` and `predict()` now use an optional `objective` parameter, which is only used in binary classification pipelines to fit for a specific objective.
- `score()` will now use a required `objectives` parameter that is used to determine all the objectives to score on. This differs from the previous behavior, where the pipeline's objective was scored on regardless.
- `score()` will now return one dictionary of all objective scores.
- `ROC` and `ConfusionMatrix` plot methods via `Auto(*).plot` have been removed by #615 and are replaced by `roc_curve` and `confusion_matrix` in *evamlm.pipelines.plot_utils'* in #704
- `normalize_confusion_matrix` has been moved to `evalml.pipelines.plot_utils` #704
- Pipelines `_name` field changed to `custom_name`
- Pipelines `supported_problem_types` field is removed because it is no longer necessary #678
- Updated argument order of objectives' *objective_function* to align with sklearn #698
- *pipeline.feature_importance_graph* has been renamed to *pipeline.graph_feature_importances* in #700
- Removed unsupported `MSLE` objective #704

v0.8.0 Apr. 1, 2020

- **Enhancements**
  - Add normalization option and information to confusion matrix #484
  - Add util function to drop rows with NaN values #487
  - Renamed *PipelineBase.name* as *PipelineBase.summary* and redefined *PipelineBase.name* as class property #491
  - Added access to parameters in Pipelines with *PipelineBase.parameters* (used to be return of *PipelineBase.describe*) #501
  - Added *fill_value* parameter for SimpleImputer #509
  - Added functionality to override component hyperparameters and made pipelines take hyperparemeters from components #516
  - Allow numpy.random.RandomState for random_state parameters #556

- **Fixes**
  - Removed unused dependency *matplotlib*, and move *category_encoders* to test reqs #572
- **Changes**
  - Undo version cap in XGBoost placed in #402 and allowed all released of XGBoost #407
  - Support pandas 1.0.0 #486
  - Made all references to the logger static #503
  - Refactored *model_type* parameter for components and pipelines to *model_family* #507
  - Refactored *problem_types* for pipelines and components into *supported_problem_types* #515
  - Moved *pipelines/utils.save_pipeline* and *pipelines/utils.load_pipeline* to *PipelineBase.save* and *PipelineBase.load* #526
  - Limit number of categories encoded by OneHotEncoder #517
- **Documentation Changes**
  - Updated API reference to remove PipelinePlot and added moved PipelineBase plotting methods #483
  - Add code style and github issue guides #463 #512
  - Updated API reference for to surface class variables for pipelines and components #537
  - Fixed README documentation link #535
  - Unhid PR references in changelog #656
- **Testing Changes**
  - Added automated dependency check PR #482, #505
  - Updated automated dependency check comment #497
  - Have build_docs job use python executor, so that env vars are set properly #547
  - Added simple test to make sure OneHotEncoder's top_n works with large number of categories #552
  - Run windows unit tests on PRs #557

Warning: Breaking Changes

- `AutoClassificationSearch` and `AutoRegressionSearch`'s `model_types` parameter has been refactored into `allowed_model_families`
- `ModelTypes` enum has been changed to `ModelFamily`
- Components and Pipelines now have a `model_family` field instead of `model_type`
- `get_pipelines` utility function now accepts `model_families` as an argument instead of `model_types`
- `PipelineBase.name` no longer returns structure of pipeline and has been replaced by `PipelineBase.summary`
- `PipelineBase.problem_types` and `Estimator.problem_types` has been renamed to `supported_problem_types`
- `pipelines/utils.save_pipeline` and `pipelines/utils.load_pipeline` moved to `PipelineBase.save` and `PipelineBase.load`

v0.7.0 Mar. 9, 2020

- **Enhancements**
  - Added emacs buffers to .gitignore #350
  - Add CatBoost (gradient-boosted trees) classification and regression components and pipelines #247
  - Added Tuner abstract base class #351
  - Added n_jobs as parameter for AutoClassificationSearch and AutoRegressionSearch #403
  - Changed colors of confusion matrix to shades of blue and updated axis order to match scikit-learn's #426
  - Added PipelineBase graph and feature_importance_graph methods, moved from previous location #423
  - Added support for python 3.8 #462
- **Fixes**
  - Fixed ROC and confusion matrix plots not being calculated if user passed own additional_objectives #276
  - Fixed ReadtheDocs FileNotFoundError exception for fraud dataset #439
- **Changes**
  - Added n_estimators as a tunable parameter for XGBoost #307
  - Remove unused parameter ObjectiveBase.fit_needs_proba #320
  - Remove extraneous parameter component_type from all components #361
  - Remove unused rankings.csv file #397
  - Downloaded demo and test datasets so unit tests can run offline #408
  - Remove *_needs_fitting* attribute from Components #398
  - Changed plot.feature_importance to show only non-zero feature importances by default, added optional parameter to show all #413
  - Refactored *PipelineBase* to take in parameter dictionary and moved pipeline metadata to class attribute #421
  - Dropped support for Python 3.5 #438
  - Removed unused *apply.py* file #449
  - Clean up requirements.txt to remove unused deps #451
  - Support installation without all required dependencies #459
- **Documentation Changes**
  - Update release.md with instructions to release to internal license key #354
- **Testing Changes**
  - Added tests for utils (and moved current utils to gen_utils) #297
  - Moved XGBoost install into it's own separate step on Windows using Conda #313
  - Rewind pandas version to before 1.0.0, to diagnose test failures for that version #325

  - Added dependency update checkin test #324
  - Rewind XGBoost version to before 1.0.0 to diagnose test failures for that version #402
  - Update dependency check to use a whitelist #417
  - Update unit test jobs to not install dev deps #455

Warning: Breaking Changes

- Python 3.5 will not be actively supported.

v0.6.0 Dec. 16, 2019

- **Enhancements**
  - Added ability to create a plot of feature importances #133
  - Add early stopping to AutoML using patience and tolerance parameters #241
  - Added ROC and confusion matrix metrics and plot for classification problems and introduce PipelineSearchPlots class #242
  - Enhanced AutoML results with search order #260
  - Added utility function to show system and environment information #300
- **Fixes**
  - Lower botocore requirement #235
  - Fixed decision_function calculation for FraudCost objective #254
  - Fixed return value of Recall metrics #264
  - Components return *self* on fit #289
- **Changes**
  - Renamed automl classes to AutoRegressionSearch and AutoClassificationSearch #287
  - Updating demo datasets to retain column names #223
  - Moving pipeline visualization to PipelinePlots class #228
  - Standarizing inputs as pd.Dataframe / pd.Series #130
  - Enforcing that pipelines must have an estimator as last component #277
  - Added ipywidgets as a dependency in requirements.txt #278
  - Added Random and Grid Search Tuners #240
- **Documentation Changes**
  - Adding class properties to API reference #244
  - Fix and filter FutureWarnings from scikit-learn #249, #257
  - Adding Linear Regression to API reference and cleaning up some Sphinx warnings #227
- **Testing Changes**
  - Added support for testing on Windows with CircleCI #226
  - Added support for doctests #233

Warning: Breaking Changes

- The `fit()` method for `AutoClassifier` and `AutoRegressor` has been renamed to `search()`.
- `AutoClassifier` has been renamed to `AutoClassificationSearch`
- `AutoRegressor` has been renamed to `AutoRegressionSearch`
- `AutoClassificationSearch.results` and `AutoRegressionSearch.results` now is a dictionary with `pipeline_results` and `search_order` keys. `pipeline_results` can be used to access a dictionary that is identical to the old `.results` dictionary. Whereas, `search_order` returns a list of the search order in terms of `pipeline_id`.
- Pipelines now require an estimator as the last component in `component_list`. Slicing pipelines now throws an `NotImplementedError` to avoid returning pipelines without an estimator.

v0.5.2 Nov. 18, 2019

- **Enhancements**
  - Adding basic pipeline structure visualization #211
- **Documentation Changes**
  - Added notebooks to build process #212

v0.5.1 Nov. 15, 2019

- **Enhancements**
  - Added basic outlier detection guardrail #151
  - Added basic ID column guardrail #135
  - Added support for unlimited pipelines with a max_time limit #70
  - Updated .readthedocs.yaml to successfully build #188
- **Fixes**
  - Removed MSLE from default additional objectives #203
  - Fixed random_state passed in pipelines #204
  - Fixed slow down in RFRegressor #206
- **Changes**
  - Pulled information for describe_pipeline from pipeline's new describe method #190
  - Refactored pipelines #108
  - Removed guardrails from Auto(*) #202, #208
- **Documentation Changes**
  - Updated documentation to show max_time enhancements #189
  - Updated release instructions for RTD #193
  - Added notebooks to build process #212
  - Added contributing instructions #213
  - Added new content #222

v0.5.0 Oct. 29, 2019

- **Enhancements**
  - Added basic one hot encoding #73
  - Use enums for model_type #110
  - Support for splitting regression datasets #112
  - Auto-infer multiclass classification #99
  - Added support for other units in max_time #125
  - Detect highly null columns #121
  - Added additional regression objectives #100
  - Show an interactive iteration vs. score plot when using fit() #134
- **Fixes**
  - Reordered *describe_pipeline* #94
  - Added type check for model_type #109
  - Fixed *s* units when setting string max_time #132
  - Fix objectives not appearing in API documentation #150
- **Changes**
  - Reorganized tests #93
  - Moved logging to its own module #119
  - Show progress bar history #111
  - Using cloudpickle instead of pickle to allow unloading of custom objectives #113
  - Removed render.py #154
- **Documentation Changes**
  - Update release instructions #140
  - Include additional_objectives parameter #124
  - Added Changelog #136
- **Testing Changes**
  - Code coverage #90
  - Added CircleCI tests for other Python versions #104
  - Added doc notebooks as tests #139
  - Test metadata for CircleCI and 2 core parallelism #137

v0.4.1 Sep. 16, 2019

- **Enhancements**
  - Added AutoML for classification and regressor using Autobase and Skopt #7 #9
  - Implemented standard classification and regression metrics #7
  - Added logistic regression, random forest, and XGBoost pipelines #7
  - Implemented support for custom objectives #15
  - Feature importance for pipelines #18

  - Serialization for pipelines #19
  - Allow fitting on objectives for optimal threshold #27
  - Added detect label leakage #31
  - Implemented callbacks #42
  - Allow for multiclass classification #21
  - Added support for additional objectives #79
- **Fixes**
  - Fixed feature selection in pipelines #13
  - Made random_seed usage consistent #45
- **Documentation Changes**
  - Documentation Changes
  - Added docstrings #6
  - Created notebooks for docs #6
  - Initialized readthedocs EvalML #6
  - Added favicon #38
- **Testing Changes**
  - Added testing for loading data #39

v0.2.0 Aug. 13, 2019

- **Enhancements**
  - Created fraud detection objective #4

v0.1.0 July. 31, 2019

- *First Release*
- **Enhancements**
  - Added lead scoring objecitve #1
  - Added basic classifier #1
- **Documentation Changes**
  - Initialized Sphinx for docs #1

INDEX

Symbols

A

B

C

D

E

F

G

H

I

L

M

N

O

P

R

S

T

V

X