EvalML Documentation

Release 0.9.0

Feature Labs, Inc.

Apr 27, 2020

GETTING STARTED

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

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

CHAPTER ONE

QUICK START

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

1.1 Load Data

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

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

1.2 Configure search

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

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

```
[3]: automl = AutoClassificationSearch(objective="f1",
                                       max_pipelines=5)
```

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

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

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

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

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

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

Optimizing for F1. Greater score is better.

Searching up to 5 pipelines.
```

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

```
 XGBoost Binary Classification Pipel...    20%|        | Elapsed:00:05
 Random Forest Binary Classification...    40%|      | Elapsed:00:17
 Logistic Regression Binary Pipeline:      60%|    | Elapsed:00:18
 XGBoost Binary Classification Pipel...    80%|  | Elapsed:00:25
 XGBoost Binary Classification Pipel...   100%|| Elapsed:00:31
 Optimization finished                    100%|| Elapsed:00:31
```

1.3 See Pipeline Rankings

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

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

[6]:
```
   id                           pipeline_name     score  \
0   2          Logistic Regression Binary Pipeline  0.969444
1   0        XGBoost Binary Classification Pipeline  0.961592
2   1  Random Forest Binary Classification Pipeline  0.954699

   high_variance_cv                                         parameters
0             False  {'impute_strategy': 'mean', 'penalty': 'l2', '...
1             False  {'impute_strategy': 'most_frequent', 'percent_...
2             False  {'impute_strategy': 'median', 'percent_feature...
```

1.4 Describe pipeline

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

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

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

Problem Type: Binary Classification
Model Family: XGBoost
Number of features: 4

Pipeline Steps
```

(continues on next page)

(continued from previous page)

```
===============
1. One Hot Encoder
         * top_n : 10
2. Simple Imputer
         * impute_strategy : most_frequent
         * fill_value : None
3. RF Classifier Select From Model
         * percent_features : 0.14894727260851873
         * threshold : -inf
4. XGBoost Classifier
         * eta : 0.4736080452737106
         * max_depth : 18
         * min_child_weight : 5.153314260276387
         * n_estimators : 660

Training
========
Training for Binary Classification problems.
Total training time (including CV): 6.4 seconds

Cross Validation
------------
               F1  Accuracy Binary  Balanced Accuracy Binary  Precision  Recall   AUC␣
↪ Log Loss Binary   MCC Binary # Training # Testing
0            0.938           0.921                     0.909      0.919   0.958 0.973␣
↪            0.235       0.831     303.000    152.000
1            0.944           0.928                     0.911      0.912   0.979 0.984␣
↪            0.161       0.846     303.000    152.000
2            0.930           0.914                     0.913      0.946   0.916 0.987␣
↪            0.173       0.818     304.000    151.000
mean         0.938           0.921                     0.911      0.926   0.951 0.981␣
↪            0.190       0.832           -          -
std          0.007           0.007                     0.002      0.018   0.032 0.007␣
↪            0.040       0.014           -          -
coef of var  0.007           0.007                     0.002      0.019   0.034 0.008␣
↪            0.210       0.017           -          -
```

1.5 Select Best pipeline

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

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

[8]:
```
OrderedDict([('F1', 0.9863013698630138)])
```

We can also visualize the structure of our pipeline:

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

[9]:

1.6 Whats next?

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

1.6.1 Install

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

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

Dependencies

Optional Dependencies

EvalML includes several dependencies in `requirements.txt` by default: `xgboost` and `catboost` support pipelines built around those modeling libraries, and `plotly` and `ipywidgets` support plotting functionality in automl searches. These dependencies are recommended but are not required in order to install and use EvalML. To install these additional dependencies run `pip install -r requirements.txt`.

Core Dependencies

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

Windows

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

1.6.2 Objective Functions

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

List of Available Objective Functions

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

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

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

- *Fraud Detection*
- *Lead Scoring*

Build your own objective Functions

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

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

1.6.3 Building a Fraud Prediction Model with EvalML

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

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

Configure "Cost of Fraud"

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

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

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

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

Search for best pipeline

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

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

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

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

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

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

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

print(X.dtypes)
```

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

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

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

automl.search(X_train, y_train)
```

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

Optimizing for Fraud Cost. Lower score is better.

Searching up to 5 pipelines.
```

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

```
 XGBoost Binary Classification Pipel...     20%|        | Elapsed:00:06
 Random Forest Binary Classification...     40%|      | Elapsed:00:21
 Logistic Regression Binary Pipeline:       60%|    | Elapsed:00:23
 XGBoost Binary Classification Pipel...     80%|  | Elapsed:00:32
 XGBoost Binary Classification Pipel...    100%|| Elapsed:00:40
```

(continues on next page)

(continued from previous page)

```
Optimization finished                          100%|| Elapsed:00:40
```

View rankings and select pipeline

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

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

[6]:
```
   id                                   pipeline_name     score  \
0   1  Random Forest Binary Classification Pipeline  0.007838
1   3        XGBoost Binary Classification Pipeline  0.007838
2   2           Logistic Regression Binary Pipeline  0.007847

   high_variance_cv                                         parameters
0             False  {'impute_strategy': 'median', 'percent_feature...
1             False  {'impute_strategy': 'most_frequent', 'percent_...
2             False  {'impute_strategy': 'mean', 'penalty': 'l2', '...
```

to select the best pipeline we can run

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

Describe pipeline

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

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

```
************************************************
* Random Forest Binary Classification Pipeline *
************************************************

Problem Type: Binary Classification
Model Family: Random Forest
Number of features: 1

Pipeline Steps
==============
1. One Hot Encoder
         * top_n : 10
2. Simple Imputer
         * impute_strategy : median
         * fill_value : None
3. RF Classifier Select From Model
         * percent_features : 0.8140470414877383
         * threshold : mean
4. Random Forest Classifier
         * n_estimators : 859
         * max_depth : 6

Training
========
Training for Binary Classification problems.
```

(continues on next page)

(continued from previous page)

```
Objective to optimize binary classification pipeline thresholds for: <evalml.
↪objectives.fraud_cost.FraudCost object at 0x7f292826b208>
Total training time (including CV): 14.7 seconds

Cross Validation
------------
             Fraud Cost   AUC  Recall  Precision # Training # Testing
0                 0.008 0.866   1.000      0.141   1066.000   667.000
1                 0.008 0.846   1.000      0.141   1066.000   667.000
2                 0.008 0.824   1.000      0.141   1067.000   666.000
mean              0.008 0.845   1.000      0.141          -         -
std               0.000 0.021   0.000      0.000          -         -
coef of var       0.007 0.025   0.000      0.001          -         -
```

Evaluate on hold out

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

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

[9]:
```
<evalml.pipelines.classification.random_forest_binary.RFBinaryClassificationPipeline␣
↪at 0x7f28d6c2be10>
```

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', 0.8402823920265778),
             ('Fraud Cost', 0.007766323050145169)])
```

Why optimize for a problem-specific objective?

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

[11]:
```
automl_auc = AutoClassificationSearch(objective='auc',
                                      additional_objectives=['recall', 'precision'],
                                      max_pipelines=5,
                                      optimize_thresholds=True)

automl_auc.search(X_train, y_train)
```

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

Optimizing for AUC. Greater score is better.

Searching up to 5 pipelines.
```

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

```
XGBoost Binary Classification Pipel...     20%|          | Elapsed:00:06
Random Forest Binary Classification...     40%|        | Elapsed:00:19
Logistic Regression Binary Pipeline:       60%|      | Elapsed:00:20
XGBoost Binary Classification Pipel...     80%|    | Elapsed:00:28
XGBoost Binary Classification Pipel...    100%|| Elapsed:00:36
Optimization finished                     100%|| Elapsed:00:36
```

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

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

[12]:
```
   id                                pipeline_name     score  \
0   4         XGBoost Binary Classification Pipeline  0.863982
3   1   Random Forest Binary Classification Pipeline  0.850172
4   2            Logistic Regression Binary Pipeline  0.807842

   high_variance_cv                                         parameters
0             False  {'impute_strategy': 'mean', 'percent_features'...
3             False  {'impute_strategy': 'median', 'percent_feature...
4             False  {'impute_strategy': 'mean', 'penalty': 'l2', '...
```

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

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

[13]:
```
<evalml.pipelines.classification.xgboost_binary.XGBoostBinaryPipeline at␣
↪0x7f28d57f95f8>
```

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

[14]:
```
OrderedDict([('AUC', 0.8268272425249169),
             ('Fraud Cost', 0.007669676738284303)])
```

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

[15]:
```
OrderedDict([('AUC', 0.8402823920265778),
             ('Fraud Cost', 0.007766323050145169)])
```

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

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

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

1.6.4 Building a Lead Scoring Model with EvalML

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

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

Configure LeadScoring

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

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

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

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

Dataset

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

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

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

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

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

     display(X.head())
```

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

Search for best pipeline

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

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

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

     print(X.dtypes)
```

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

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

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

     automl.search(X_train, y_train)
```

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

Optimizing for Lead Scoring. Greater score is better.

Searching up to 5 pipelines.
```

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

```
 XGBoost Binary Classification Pipel...    20%|         | Elapsed:00:08
 Random Forest Binary Classification...    40%|       | Elapsed:00:23
 Logistic Regression Binary Pipeline:      60%|     | Elapsed:00:26
 XGBoost Binary Classification Pipel...    80%|   | Elapsed:00:35
 XGBoost Binary Classification Pipel...   100%|| Elapsed:00:46
 Optimization finished                    100%|| Elapsed:00:46
```

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   3         XGBoost Binary Classification Pipeline  15.095733
3   2            Logistic Regression Binary Pipeline  13.158047
4   1   Random Forest Binary Classification Pipeline  11.239462

   high_variance_cv                                        parameters
0             False  {'impute_strategy': 'most_frequent', 'percent_...
3              True  {'impute_strategy': 'mean', 'penalty': 'l2', '...
4              True  {'impute_strategy': 'median', 'percent_feature...
```

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"])
```

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

Problem Type: Binary Classification
Model Family: XGBoost
Number of features: 1

Pipeline Steps
==============
1. One Hot Encoder
         * top_n : 10
2. Simple Imputer
         * impute_strategy : most_frequent
         * fill_value : None
3. RF Classifier Select From Model
         * percent_features : 0.14894727260851873
         * threshold : -inf
4. XGBoost Classifier
         * eta : 0.4736080452737106
         * max_depth : 18
         * min_child_weight : 5.153314260276387
         * n_estimators : 660

Training
========
Training for Binary Classification problems.
Objective to optimize binary classification pipeline thresholds for: <evalml.
↪objectives.lead_scoring.LeadScoring object at 0x7f3d99b83780>
Total training time (including CV): 9.6 seconds

Cross Validation
----------------
             Lead Scoring   AUC # Training # Testing
0                  15.606 0.519   2479.000  1550.000
1                  14.523 0.502   2479.000  1550.000
```

(continues on next page)

(continued from previous page)

```
2                  15.158 0.536   2480.000  1549.000
mean               15.096 0.519         -         -
std                 0.545 0.017         -         -
coef of var         0.036 0.033         -         -
```

Evaluate on hold out

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

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

[9]:
```
<evalml.pipelines.classification.xgboost_binary.XGBoostBinaryPipeline at␣
↪0x7f3d7c2a9940>
```

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', 0.4454215775158658),
             ('Lead Scoring', 12.218400687876182)])
```

Why optimize for a problem-specific objective?

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

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

automl_auc.search(X_train, y_train)
```

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

Optimizing for AUC. Greater score is better.

Searching up to 5 pipelines.
```

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

```
 XGBoost Binary Classification Pipel...    20%|        | Elapsed:00:05
 Random Forest Binary Classification...    40%|      | Elapsed:00:17
 Logistic Regression Binary Pipeline:      60%|    | Elapsed:00:17
 XGBoost Binary Classification Pipel...    80%|  | Elapsed:00:25
 XGBoost Binary Classification Pipel...   100%|| Elapsed:00:32
 Optimization finished                    100%|| Elapsed:00:32
```

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

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

```
[12]:    id                              pipeline_name     score  \
0   2           Logistic Regression Binary Pipeline  0.695618
1   1  Random Forest Binary Classification Pipeline  0.591495
2   0        XGBoost Binary Classification Pipeline  0.571654

   high_variance_cv                                         parameters
0             False  {'impute_strategy': 'mean', 'penalty': 'l2', '...
1             False  {'impute_strategy': 'median', 'percent_feature...
2             False  {'impute_strategy': 'most_frequent', 'percent_...
```

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

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

```
[13]: <evalml.pipelines.classification.logistic_regression_binary.
↪LogisticRegressionBinaryPipeline at 0x7f3d7c609d30>
```

```
[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', 0.6510350559081293), ('Lead Scoring', 0.0)])
```

When we optimize for AUC, we can see that the AUC score from this pipeline is better than the AUC score from the pipeline optimized for lead scoring. However, the revenue per lead gained was only `$7` per lead when optimized for AUC and was `$45` when optimized for lead scoring. As a result, we would gain up to 6x the amount of revenue if we optimized for lead scoring.

This happens because optimizing for AUC does not take into account the user-specified true_positive (dollar amount to be gained with a successful lead) and false_positive (dollar amount to be lost with an unsuccessful lead) values. Thus, the best pipelines may produce the highest AUC but may not actually generate the most revenue through lead scoring.

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

1.6.5 Custom Objective Functions

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

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

How to Create a Objective Function

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

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

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

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

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


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

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

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

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

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

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

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

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

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

(continues on next page)

(continued from previous page)

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

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

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

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

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

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

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

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

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

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

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

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

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

        loss_per_total_processed = loss / transaction_amount.sum()

        return loss_per_total_processed
```

1.6.6 Setting up pipeline search

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

How it works

EvalML selects and tunes machine learning pipelines built of numerous steps. This includes encoding categorical data, missing value imputation, feature selection, feature scaling, and finally machine learning. As EvalML tunes pipelines, it uses the objective function selected and configured by the user to guide its search.

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

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

Selecting problem type

EvalML supports both classification and regression problems. You select your problem type by importing the appropriate class.

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

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

[2]:
```
<evalml.automl.auto_classification_search.AutoClassificationSearch at 0x7fcdc2b3c5f8>
```

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

[3]:
```
<evalml.automl.auto_regression_search.AutoRegressionSearch at 0x7fcdabb4d5c0>
```

Setting the Objective Function

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

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

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

AutoClassificationSearch(objective=fraud_objective, optimize_thresholds=True)
```

[4]:
```
<evalml.automl.auto_classification_search.AutoClassificationSearch at 0x7fcdabb60b70>
```

Evaluate on Additional Objectives

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

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

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

AutoClassificationSearch(objective='AUC', additional_objectives=[fraud_objective],␣
↪optimize_thresholds=False)
```

[5]:
```
<evalml.automl.auto_classification_search.AutoClassificationSearch at 0x7fcdabb6ae80>
```

Selecting Model Types

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

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

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

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

[7]:
```
[evalml.pipelines.classification.random_forest_binary.RFBinaryClassificationPipeline]
```

you can see a list of all supported models like this

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

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

[9]:
```python
evalml.list_model_families("regression")
```

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

Limiting Search Time

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

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

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

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

[10]:
```
<evalml.automl.auto_classification_search.AutoClassificationSearch at 0x7fcdabb091d0>
```

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

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

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

[12]:
```
<evalml.automl.auto_classification_search.AutoClassificationSearch at 0x7fcdabb1ed68>
```

Early Stopping

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

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

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

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

Optimizing for MSE. Lower score is better.

Searching up to 10 pipelines.
```

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

```
 XGBoost Regression Pipeline:                 10%|          | Elapsed:00:03
 Cat Boost Regression Pipeline:               20%|         | Elapsed:00:12
 Random Forest Regression Pipeline:           30%|        | Elapsed:00:15
 XGBoost Regression Pipeline:                 40%|       | Elapsed:00:20


2 iterations without improvement. Stopping search early...
 Optimization finished                        40%|       | Elapsed:00:20
```

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

[14]:
```
   id                    pipeline_name        score  high_variance_cv  \
0   1       Cat Boost Regression Pipeline  3566.688649             False
1   0         XGBoost Regression Pipeline  4443.846724             False
2   2   Random Forest Regression Pipeline  5615.790436             False

                                          parameters
0  {'impute_strategy': 'most_frequent', 'n_estima...
1  {'impute_strategy': 'most_frequent', 'percent_...
2  {'impute_strategy': 'most_frequent', 'percent_...
```

Control Cross Validation

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

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

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

1.6.7 Exploring search results

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

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

X, y = evalml.demos.load_breast_cancer()

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

automl.search(X, y)
```

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

Optimizing for F1. Greater score is better.

Searching up to 5 pipelines.
```

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

```
 XGBoost Binary Classification Pipel...    20%|        | Elapsed:00:05
 Random Forest Binary Classification...    40%|      | Elapsed:00:18
 Logistic Regression Binary Pipeline:      60%|    | Elapsed:00:19
 XGBoost Binary Classification Pipel...    80%|  | Elapsed:00:26
 XGBoost Binary Classification Pipel...   100%|| Elapsed:00:32
 Optimization finished                    100%|| Elapsed:00:32
```

View Rankings

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

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

```
[2]:    id                              pipeline_name     score  \
0   2           Logistic Regression Binary Pipeline  0.982042
1   0        XGBoost Binary Classification Pipeline  0.976191
2   1  Random Forest Binary Classification Pipeline  0.958032

   high_variance_cv                                         parameters
0             False  {'impute_strategy': 'mean', 'penalty': 'l2', '...
1             False  {'impute_strategy': 'most_frequent', 'percent_...
2             False  {'impute_strategy': 'median', 'percent_feature...
```

Describe Pipeline

Each pipeline is given an `id`. We can get more information about any particular pipeline using that `id`. Here, we will get more information about the pipeline with `id = 0`.

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

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

Problem Type: Binary Classification
Model Family: XGBoost
Number of features: 25

Pipeline Steps
==============
1. One Hot Encoder
	 * top_n : 10
2. Simple Imputer
	 * impute_strategy : most_frequent
	 * fill_value : None
3. RF Classifier Select From Model
	 * percent_features : 0.8487792213962843
	 * threshold : -inf
4. XGBoost Classifier
	 * eta : 0.38438170729269994
	 * max_depth : 7
	 * min_child_weight : 1.5104167958569887
	 * n_estimators : 397

Training
========
Training for Binary Classification problems.
Total training time (including CV): 5.4 seconds

Cross Validation
----------------
               F1  Accuracy Binary  Balanced Accuracy Binary  Precision  Recall   AUC␣
↪ Log Loss Binary  MCC Binary # Training # Testing
```

(continues on next page)

(continued from previous page)

```
0           0.962           0.953                           0.954      0.974    0.950 0.988␣
↪            0.138       0.900     379.000    190.000
1           0.979           0.974                           0.965      0.960    1.000 0.997␣
↪            0.071       0.945     379.000    190.000
2           0.987           0.984                           0.982      0.983    0.992 0.997␣
↪            0.075       0.966     380.000    189.000
mean        0.976           0.970                           0.967      0.972    0.980 0.994␣
↪            0.095       0.937          -          -
std         0.013           0.016                           0.014      0.012    0.027 0.005␣
↪            0.037       0.034          -          -
coef of var 0.013           0.017                           0.015      0.012    0.028 0.006␣
↪            0.395       0.036          -          -
```

Get Pipeline

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

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

[4]:
```
<evalml.pipelines.classification.xgboost_binary.XGBoostBinaryPipeline at␣
↪0x7f5c40ff2a20>
```

Get best pipeline

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

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

[5]:
```
<evalml.pipelines.classification.logistic_regression_binary.
↪LogisticRegressionBinaryPipeline at 0x7f5c4065c358>
```

Feature Importances

We can get the feature importances of the resulting pipeline

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

[6]:

	feature	importance
0	mean concave points	0.465049
1	worst concave points	0.246494
2	worst radius	0.089427
3	worst area	0.045472
4	mean texture	0.029848
5	worst concavity	0.020971
6	area error	0.020298
7	radius error	0.018571
8	worst texture	0.014910
9	worst smoothness	0.010209
10	mean area	0.006383
11	mean concavity	0.004976
12	mean smoothness	0.004681
13	worst perimeter	0.004660

(continues on next page)

(continued from previous page)

```
14           worst symmetry    0.004073
15          concavity error    0.003436
16         mean compactness    0.003422
17  worst fractal dimension    0.002782
18         smoothness error    0.001911
19  fractal dimension error    0.001905
20           symmetry error    0.000420
21          perimeter error    0.000101
22              mean radius    0.000000
23           mean perimeter    0.000000
24        worst compactness    0.000000
```

We can also create a bar plot of the feature importances

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

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

Access raw results

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

[8]:
```
automl.results
```

[8]:
```
{'pipeline_results': {0: {'id': 0,
   'pipeline_name': 'XGBoost Binary Classification Pipeline',
   'pipeline_summary': 'XGBoost Classifier w/ One Hot Encoder + Simple Imputer + RF␣
↪Classifier Select From Model',
   'parameters': {'impute_strategy': 'most_frequent',
    'percent_features': 0.8487792213962843,
    'threshold': -inf,
    'eta': 0.38438170729269994,
    'max_depth': 7,
    'min_child_weight': 1.5104167958569887,
    'n_estimators': 397},
   'score': 0.9761912315723671,
   'high_variance_cv': False,
   'training_time': 5.40069317817688,
   'cv_data': [{'all_objective_scores': OrderedDict([('F1',
                   0.9617021276595743),
                  ('Accuracy Binary', 0.9526315789473684),
                  ('Balanced Accuracy Binary', 0.9536631554030062),
                  ('Precision', 0.9741379310344828),
                  ('Recall', 0.9495798319327731),
                  ('AUC', 0.9876908509882827),
                  ('Log Loss Binary', 0.13808748615334288),
                  ('MCC Binary', 0.9001633057441626),
                  ('# Training', 379),
                  ('# Testing', 190)]),
     'score': 0.9617021276595743},
    {'all_objective_scores': OrderedDict([('F1', 0.9794238683127572),
                  ('Accuracy Binary', 0.9736842105263158),
                  ('Balanced Accuracy Binary', 0.9647887323943662),
```

(continues on next page)

(continued from previous page)

```
                  ('Precision', 0.9596774193548387),
                  ('Recall', 1.0),
                  ('AUC', 0.9973961415552136),
                  ('Log Loss Binary', 0.07131786501827025),
                  ('MCC Binary', 0.9445075449666159),
                  ('# Training', 379),
                  ('# Testing', 190)]),
    'score': 0.9794238683127572},
   {'all_objective_scores': OrderedDict([('F1', 0.9874476987447698),
                  ('Accuracy Binary', 0.9841269841269841),
                  ('Balanced Accuracy Binary', 0.9815126050420169),
                  ('Precision', 0.9833333333333333),
                  ('Recall', 0.9915966386554622),
                  ('AUC', 0.996998799519808),
                  ('Log Loss Binary', 0.07531116866342562),
                  ('MCC Binary', 0.9659285184801715),
                  ('# Training', 380),
                  ('# Testing', 189)]),
    'score': 0.9874476987447698}]},
 1: {'id': 1,
  'pipeline_name': 'Random Forest Binary Classification Pipeline',
  'pipeline_summary': 'Random Forest Classifier w/ One Hot Encoder + Simple Imputer␣
↪+ RF Classifier Select From Model',
  'parameters': {'impute_strategy': 'median',
   'percent_features': 0.8140470414877383,
   'threshold': 'mean',
   'n_estimators': 859,
   'max_depth': 6},
  'score': 0.9580315415303952,
  'high_variance_cv': False,
  'training_time': 12.612428903579712,
  'cv_data': [{'all_objective_scores': OrderedDict([('F1',
                   0.9361702127659575),
                  ('Accuracy Binary', 0.9210526315789473),
                  ('Balanced Accuracy Binary', 0.9199313528228192),
                  ('Precision', 0.9482758620689655),
                  ('Recall', 0.9243697478991597),
                  ('AUC', 0.9766836311989585),
                  ('Log Loss Binary', 0.20455160484518806),
                  ('MCC Binary', 0.833232300751445),
                  ('# Training', 379),
                  ('# Testing', 190)]),
    'score': 0.9361702127659575},
   {'all_objective_scores': OrderedDict([('F1', 0.9672131147540983),
                  ('Accuracy Binary', 0.9578947368421052),
                  ('Balanced Accuracy Binary', 0.9465025446798438),
                  ('Precision', 0.944),
                  ('Recall', 0.9915966386554622),
                  ('AUC', 0.9838442419221209),
                  ('Log Loss Binary', 0.14826817405619716),
                  ('MCC Binary', 0.9106361866954563),
                  ('# Training', 379),
                  ('# Testing', 190)]),
    'score': 0.9672131147540983},
   {'all_objective_scores': OrderedDict([('F1', 0.9707112970711297),
                  ('Accuracy Binary', 0.9629629629629629),
                  ('Balanced Accuracy Binary', 0.9588235294117646),
```

(continues on next page)

(continued from previous page)

```
                   ('Precision', 0.9666666666666667),
                   ('Recall', 0.9747899159663865),
                   ('AUC', 0.9942376950780312),
                   ('Log Loss Binary', 0.10344817959803934),
                   ('MCC Binary', 0.9204135621119959),
                   ('# Training', 380),
                   ('# Testing', 189)]),
     'score': 0.9707112970711297}]},
  2: {'id': 2,
   'pipeline_name': 'Logistic Regression Binary Pipeline',
   'pipeline_summary': 'Logistic Regression Classifier w/ One Hot Encoder + Simple␣
↪Imputer + Standard Scaler',
   'parameters': {'impute_strategy': 'mean',
    'penalty': 'l2',
    'C': 0.21198179042885398},
   'score': 0.9820415596969072,
   'high_variance_cv': False,
   'training_time': 1.3641350269317627,
   'cv_data': [{'all_objective_scores': OrderedDict([('F1', 0.979253112033195),
                   ('Accuracy Binary', 0.9736842105263158),
                   ('Balanced Accuracy Binary', 0.9676293052432241),
                   ('Precision', 0.9672131147540983),
                   ('Recall', 0.9915966386554622),
                   ('AUC', 0.9904130666351048),
                   ('Log Loss Binary', 0.10058063355386729),
                   ('MCC Binary', 0.943843520216036),
                   ('# Training', 379),
                   ('# Testing', 190)]),
     'score': 0.979253112033195},
    {'all_objective_scores': OrderedDict([('F1', 0.9794238683127572),
                   ('Accuracy Binary', 0.9736842105263158),
                   ('Balanced Accuracy Binary', 0.9647887323943662),
                   ('Precision', 0.9596774193548387),
                   ('Recall', 1.0),
                   ('AUC', 0.9989347851816782),
                   ('Log Loss Binary', 0.07682029301742287),
                   ('MCC Binary', 0.9445075449666159),
                   ('# Training', 379),
                   ('# Testing', 190)]),
     'score': 0.9794238683127572},
    {'all_objective_scores': OrderedDict([('F1', 0.9874476987447698),
                   ('Accuracy Binary', 0.9841269841269841),
                   ('Balanced Accuracy Binary', 0.9815126050420169),
                   ('Precision', 0.9833333333333333),
                   ('Recall', 0.9915966386554622),
                   ('AUC', 0.997358943577431),
                   ('Log Loss Binary', 0.08090403408994591),
                   ('MCC Binary', 0.9659285184801715),
                   ('# Training', 380),
                   ('# Testing', 189)]),
     'score': 0.9874476987447698}]},
  3: {'id': 3,
   'pipeline_name': 'XGBoost Binary Classification Pipeline',
   'pipeline_summary': 'XGBoost Classifier w/ One Hot Encoder + Simple Imputer + RF␣
↪Classifier Select From Model',
   'parameters': {'impute_strategy': 'most_frequent',
    'percent_features': 0.14894727260851873,
```

(continues on next page)

(continued from previous page)

```
    'threshold': -inf,
    'eta': 0.4736080452737106,
    'max_depth': 18,
    'min_child_weight': 5.153314260276387,
    'n_estimators': 660},
   'score': 0.941255546698183,
   'high_variance_cv': False,
   'training_time': 6.766803026199341,
   'cv_data': [{'all_objective_scores': OrderedDict([('F1',
                   0.9264069264069265),
                  ('Accuracy Binary', 0.9105263157894737),
                  ('Balanced Accuracy Binary', 0.9143685643271393),
                  ('Precision', 0.9553571428571429),
                  ('Recall', 0.8991596638655462),
                  ('AUC', 0.9715942715114214),
                  ('Log Loss Binary', 0.2351054900534157),
                  ('MCC Binary', 0.8150103776135726),
                  ('# Training', 379),
                  ('# Testing', 190)]),
     'score': 0.9264069264069265},
    {'all_objective_scores': OrderedDict([('F1', 0.9482071713147411),
                  ('Accuracy Binary', 0.9315789473684211),
                  ('Balanced Accuracy Binary', 0.9084507042253521),
                  ('Precision', 0.9015151515151515),
                  ('Recall', 1.0),
                  ('AUC', 0.9784589892294946),
                  ('Log Loss Binary', 0.18131056035061574),
                  ('MCC Binary', 0.858166066103978),
                  ('# Training', 379),
                  ('# Testing', 190)]),
     'score': 0.9482071713147411},
    {'all_objective_scores': OrderedDict([('F1', 0.9491525423728814),
                  ('Accuracy Binary', 0.9365079365079365),
                  ('Balanced Accuracy Binary', 0.9348739495798319),
                  ('Precision', 0.9572649572649573),
                  ('Recall', 0.9411764705882353),
                  ('AUC', 0.9841536614645858),
                  ('Log Loss Binary', 0.16492396169563844),
                  ('MCC Binary', 0.8648817040445186),
                  ('# Training', 380),
                  ('# Testing', 189)]),
     'score': 0.9491525423728814}]},
  4: {'id': 4,
   'pipeline_name': 'XGBoost Binary Classification Pipeline',
   'pipeline_summary': 'XGBoost Classifier w/ One Hot Encoder + Simple Imputer + RF␣
↪Classifier Select From Model',
   'parameters': {'impute_strategy': 'mean',
    'percent_features': 0.6435218111142487,
    'threshold': 'mean',
    'eta': 0.9446689170495841,
    'max_depth': 11,
    'min_child_weight': 4.731957459914713,
    'n_estimators': 676},
   'score': 0.9486606279409701,
   'high_variance_cv': False,
   'training_time': 6.595508098602295,
   'cv_data': [{'all_objective_scores': OrderedDict([('F1',
```

(continues on next page)

(continued from previous page)

```
                    0.9210526315789473),
                   ('Accuracy Binary', 0.9052631578947369),
                   ('Balanced Accuracy Binary', 0.9130074565037283),
                   ('Precision', 0.963302752293578),
                   ('Recall', 0.8823529411764706),
                   ('AUC', 0.975085808971476),
                   ('Log Loss Binary', 0.2385086150043016),
                   ('MCC Binary', 0.8080435814236837),
                   ('# Training', 379),
                   ('# Testing', 190)]),
     'score': 0.9210526315789473},
    {'all_objective_scores': OrderedDict([('F1', 0.9709543568464729),
                   ('Accuracy Binary', 0.9631578947368421),
                   ('Balanced Accuracy Binary', 0.9563853710498283),
                   ('Precision', 0.9590163934426229),
                   ('Recall', 0.9831932773109243),
                   ('AUC', 0.9697597348798673),
                   ('Log Loss Binary', 0.13901819948468505),
                   ('MCC Binary', 0.9211492315750531),
                   ('# Training', 379),
                   ('# Testing', 190)]),
     'score': 0.9709543568464729},
    {'all_objective_scores': OrderedDict([('F1', 0.9539748953974896),
                   ('Accuracy Binary', 0.9417989417989417),
                   ('Balanced Accuracy Binary', 0.9361344537815126),
                   ('Precision', 0.95),
                   ('Recall', 0.957983193277311),
                   ('AUC', 0.9845738295318127),
                   ('Log Loss Binary', 0.13538144654258666),
                   ('MCC Binary', 0.8748986057438203),
                   ('# Training', 380),
                   ('# Testing', 189)]),
     'score': 0.9539748953974896}]}},
 'search_order': [0, 1, 2, 3, 4]}
```

1.6.8 Regression Example

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


X, y = evalml.demos.load_diabetes()

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

automl.search(X, y)
```

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

Optimizing for R2. Greater score is better.
```

(continues on next page)

(continued from previous page)

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

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

```
 XGBoost Regression Pipeline:                 20%|        | Elapsed:00:03
 Cat Boost Regression Pipeline:               40%|      | Elapsed:00:12
 Random Forest Regression Pipeline:           60%|    | Elapsed:00:15
 XGBoost Regression Pipeline:                 80%|  | Elapsed:00:21
 XGBoost Regression Pipeline:                100%|| Elapsed:00:26
 Optimization finished                       100%|| Elapsed:00:26
```

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

[2]:
```
   id                     pipeline_name     score  high_variance_cv  \
0   1       Cat Boost Regression Pipeline  0.397415             False
1   0         XGBoost Regression Pipeline  0.245869              True
3   2   Random Forest Regression Pipeline  0.051449              True

                                          parameters
0  {'impute_strategy': 'most_frequent', 'n_estima...
1  {'impute_strategy': 'most_frequent', 'percent_...
3  {'impute_strategy': 'most_frequent', 'percent_...
```

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

[3]:
```
<evalml.pipelines.regression.catboost.CatBoostRegressionPipeline at 0x7fea77afae48>
```

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

[4]:
```
<evalml.pipelines.regression.xgboost_regression.XGBoostRegressionPipeline at␣
↪0x7fea7832bba8>
```

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

```
*******************************
* XGBoost Regression Pipeline *
*******************************

Problem Type: Regression
Model Family: XGBoost
Number of features: 8

Pipeline Steps
==============
1. One Hot Encoder
         * top_n : 10
2. Simple Imputer
         * impute_strategy : most_frequent
         * fill_value : None
3. RF Regressor Select From Model
         * percent_features : 0.8487792213962843
         * threshold : -inf
4. XGBoost Regressor
         * eta : 0.38438170729269994
         * max_depth : 7
```

(continues on next page)

(continued from previous page)

```
        * min_child_weight : 1.5104167958569887
        * n_estimators : 397

Training
========
Training for Regression problems.
Total training time (including CV): 3.8 seconds

Cross Validation
-----------
Warning! High variance within cross validation scores. Model may not perform as␣
↪estimated on unseen data.
               R2    MAE      MSE  MedianAE  MaxError  ExpVariance # Training #␣
↪Testing
0           0.265 51.909 4204.782    45.175   174.089        0.266    294.000   148.
↪000
1           0.339 50.432 4190.876    40.601   162.048        0.340    295.000   147.
↪000
2           0.134 56.410 4935.882    47.643   206.828        0.135    295.000   147.
↪000
mean        0.246 52.917 4443.847    44.473   180.989        0.247          -        ␣
↪-
std         0.104  3.114  426.172     3.573    23.174        0.104          -        ␣
↪-
coef of var 0.424  0.059    0.096     0.080     0.128        0.419          -        ␣
↪-
```

1.6.9 EvalML Components and Pipelines

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

XGBoost Pipeline

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

Parameters

The parameters dictionary needs to be in the format of a two-layered dictionary where the first key-value pair is the component name and component parameters dictionary. The component parameters dictionary consists of a key value pair of parameter name and parameter values. An example will be shown below and component parameters can be found *here*.

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

X, y = load_breast_cancer()

parameters = {
```

(continues on next page)

(continued from previous page)

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

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

[1]:

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

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

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

Problem Type: Binary Classification
Model Family: XGBoost

Pipeline Steps
==============
1. One Hot Encoder
         * top_n : 10
2. Simple Imputer
         * impute_strategy : mean
         * fill_value : None
3. RF Classifier Select From Model
         * percent_features : 0.5
         * threshold : -inf
4. XGBoost Classifier
         * eta : 0.5
         * max_depth : 10
         * min_child_weight : 5
         * n_estimators : 20
```

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

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

[3]:
```
OrderedDict([('Recall', 0.9971988795518207)])
```

1.6.10 EvalML Components

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

Component Classes

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

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

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

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

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

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

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

display(X)
```

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

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

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

clf = LogisticRegressionClassifier()

X = X
y = [1, 0]

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

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

Component Types

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

Transformer Types

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

Estimator Types

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

1.6.11 Custom Pipelines in EvalML

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

Requirements

A custom pipeline must adhere to the following requirements:

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

Pipeline Configuration

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

Custom Name

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

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

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

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

print(CustomPipeline.name)
```

```
A custom pipeline name
```

Custom Hyperparameters

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

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

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

class CustomPipeline(BinaryClassificationPipeline):
    component_graph = ['Simple Imputer', 'Logistic Regression Classifier']
    custom_hyperparameters = {
        'impute_strategy': ['most_frequent']
    }

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

```
Without custom hyperparameters:
{'impute_strategy': ['mean', 'median', 'most_frequent'], 'penalty': ['l2'], 'C':␣
↪Real(low=0.01, high=10, prior='uniform', transform='identity')}

With custom hyperparameters:
{'impute_strategy': ['most_frequent'], 'penalty': ['l2'], 'C': Real(low=0.01, high=10,
↪ prior='uniform', transform='identity')}
```

1.6.12 Guardrails

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

Missing Data

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

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

from evalml.guardrails.utils import detect_highly_null

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

detect_highly_null(X, percent_threshold=0.8)
```

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

Abnormal Data

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

ID Columns

ID columns in your dataset provide little to no benefit to a machine learning pipeline as the pipeline cannot extrapolate useful information from unique identifiers. Thus, detect_id_columns() reminds you if these columns exists.

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

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


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

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

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

Outliers

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

Below we generate a random dataset with some outliers.

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

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

We then utilize `detect_outliers` to rediscover these outliers.

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

detect_outliers(X)
```

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

1.6.13 Avoiding Overfitting

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

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

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

Detecting Label Leakage

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

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

[2]:
```
import pandas as pd

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

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

automl = evalml.AutoClassificationSearch(
    max_pipelines=1,
```

(continues on next page)

(continued from previous page)

```
    allowed_model_families=["linear_model"],
)

automl.search(X, y)
```

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

Optimizing for Log Loss Binary. Lower score is better.

Searching up to 1 pipelines.
WARNING: Possible label leakage: leaked_feature, leaked_feature_2
```

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

```
 Logistic Regression Binary Pipeline:     100%|| Elapsed:00:00
 Optimization finished                    100%|| Elapsed:00:00
```

In the example above, EvalML warned about the input features `leaked_feature` and `leak_feature_2`, which are both very closely correlated with the label we are trying to predict. If you'd like to turn this check off, set `detect_label_leakage=False`.

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

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

[3]:

	feature	importance
0	leaked_feature	-1.789393
1	leaked_feature_2	-1.645127
2	valid_feature	-0.398465

Perform cross-validation for pipeline evaluation

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

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

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

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

X, y = evalml.demos.load_breast_cancer()

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

automl.search(X, y)
```

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

Optimizing for Log Loss Binary. Lower score is better.

Searching up to 1 pipelines.
```

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

```
 XGBoost Binary Classification Pipel...   100%|| Elapsed:00:10
 Optimization finished                    100%|| Elapsed:00:10
```

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

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

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

Problem Type: Binary Classification
Model Family: XGBoost
Number of features: 25

Pipeline Steps
==============
1. One Hot Encoder
         * top_n : 10
2. Simple Imputer
         * impute_strategy : most_frequent
         * fill_value : None
3. RF Classifier Select From Model
         * percent_features : 0.8487792213962843
         * threshold : -inf
4. XGBoost Classifier
         * eta : 0.38438170729269994
         * max_depth : 7
         * min_child_weight : 1.5104167958569887
         * n_estimators : 397

Training
========
Training for Binary Classification problems.
Total training time (including CV): 10.5 seconds

Cross Validation
----------------
Warning! High variance within cross validation scores. Model may not perform as␣
↪estimated on unseen data.
             Log Loss Binary  Accuracy Binary  Balanced Accuracy Binary    F1 ␣
↪Precision  Recall   AUC  MCC Binary # Training # Testing
0                      0.532            0.840                     0.860 0.863      0.
↪953   0.788 0.949       0.691     83.000    81.000
```

(continues on next page)

(continued from previous page)

```
1                       0.045             0.988                       0.988 0.988      1.
↪000    0.977 1.000        0.976    164.000     81.000
2                       0.111             0.975                       0.963 0.982      0.
↪964    1.000 0.990        0.945    245.000     81.000
3                       0.062             0.975                       0.983 0.982      1.
↪000    0.966 0.999        0.942    326.000     81.000
4                       0.083             0.963                       0.977 0.976      1.
↪000    0.953 0.997        0.900    407.000     81.000
5                       0.068             0.963                       0.944 0.975      0.
↪967    0.983 0.998        0.903    488.000     81.000
mean                    0.150             0.951                       0.952 0.961      0.
↪981    0.945 0.989        0.893          -          -
std                     0.189             0.055                       0.048 0.048      0.
↪021    0.078 0.020        0.103          -          -
coef of var             1.256             0.058                       0.050 0.050      0.
↪022    0.083 0.020        0.115          -          -
```

Detect unstable pipelines

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

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

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

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

[6]:
```
   id                            pipeline_name     score  high_variance_cv  \
0   0  XGBoost Binary Classification Pipeline  0.150153              True

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

Create holdout for model validation

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

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

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

[8]:
```
automl = evalml.AutoClassificationSearch(
    objective="recall",
    max_pipelines=3,
    detect_label_leakage=True
)
automl.search(X_train, y_train)
```

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

Optimizing for Recall. Greater score is better.

Searching up to 3 pipelines.
```

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

```
 XGBoost Binary Classification Pipel...     33%|       | Elapsed:00:05
 Random Forest Binary Classification...     67%|    | Elapsed:00:17
 Logistic Regression Binary Pipeline:      100%|| Elapsed:00:17
 Optimization finished                     100%|| Elapsed:00:17
```

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

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

```
[9]: OrderedDict([('Recall', 0.9722222222222222)])
```

1.6.14 Changelog

Future Releases

- Enhancements
- Fixes
- Changes
- **Documentation Changes**
  - Add instructions to freeze *master* on *release.md* #726
- Testing Changes

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
- **Fixes**
  - Removed direct access to *cls.component_graph* #595
  - Add testing files to .gitignore #625

  - Remove circular dependencies from *Makefile* #637
  - Add error case for *normalize_confusion_matrix()* #640
  - Fixed XGBoostClassifier and XGBoostRegressor bug with feature names that contain [,], or < #659
  - Update make_pipeline_graph to not accidentally create empty file when testing if path is valid #649
  - Fix pip installation warning about docsutils version, from boto dependency #664
  - Removed zero division warning for F1/precision/recall metrics #671
  - Fixed *summary* for pipelines without estimators #707
- **Changes**
  - Updated default objective for binary/multiseries classification to log loss #613
  - Created classification and regression pipeline subclasses and removed objective as an attribute of pipeline classes #405
  - Changed the output of *score* to return one dictionary #429
  - Created binary and multiclass objective subclasses #504
  - Updated objectives API #445
  - Removed call to *get_plot_data* from AutoML #615
  - Set *raise_error* to default to True for AutoML classes #638
  - Remove unnecessary "u" prefixes on some unicode strings #641
  - Changed one-hot encoder to return uint8 dtypes instead of ints #653
  - Pipeline *_name* field changed to *custom_name* #650
  - Removed *graphs.py* and moved methods into *PipelineBase* #657, #665
  - Remove s3fs as a dev dependency #664
  - Changed requirements-parser to be a core dependency #673
  - Replace *supported_problem_types* field on pipelines with *problem_type* attribute on base classes #678
  - Changed AutoML to only show best results for a given pipeline template in *rankings*, added *full_rankings* property to show all #682
  - Update *ModelFamily* values: don't list xgboost/catboost as classifiers now that we have regression pipelines for them #677
  - Changed AutoML's *describe_pipeline* to get problem type from pipeline instead #685
  - Standardize *import_or_raise* error messages #683
  - Updated argument order of objectives to align with sklearn's #698
  - Renamed *pipeline.feature_importance_graph* to *pipeline.graph_feature_importances* #700
  - Moved ROC and confusion matrix methods to *evalml.pipelines.plot_utils* #704
  - Renamed *MultiClassificationObjective* to *MulticlassClassificationObjective*, to align with pipeline naming scheme #715
- **Documentation Changes**

- Fixed some sphinx warnings #593
- Fixed docstring for AutoClassificationSearch with correct command #599
- Limit readthedocs formats to pdf, not htmlzip and epub #594 #600
- Clean up objectives API documentation #605
- Fixed function on Exploring search results page #604
- Update release process doc #567
- AutoClassificationSearch and AutoRegressionSearch show inherited methods in API reference #651
- Fixed improperly formatted code in breaking changes for changelog #655
- Added configuration to treat Sphinx warnings as errors #660
- Removed separate plotting section for pipelines in API reference #657, #665
- Have leads example notebook load S3 files using https, so we can delete s3fs dev dependency #664
- Categorized components in API reference and added descriptions for each category #663
- Fixed Sphinx warnings about BalancedAccuracy objective #669
- Updated API reference to include missing components and clean up pipeline docstrings #689
- Reorganize API ref, and clarify pipeline sub-titles #688
- Add and update preprocessing utils in API reference #687
- Added inheritance diagrams to API reference #695
- Documented which default objective AutoML optimizes for #699
- Create seperate install page #701
- Include more utils in API ref, like *import_or_raise* #704
- Add more color to pipeline documentation #705

- **Testing Changes**
  - Matched install commands of *check_latest_dependencies* test and it's GitHub action #578
  - Added Github app to auto assign PR author as assignee #477
  - Removed unneeded conda installation of xgboost in windows checkin tests #618
  - Update graph tests to always use tmpfile dir #649
  - Changelog checkin test workaround for release PRs: If 'future release' section is empty of PR refs, pass check #658
  - Add changelog checkin test exception for *dep-update* branch #723

Warning: Breaking Changes

- Pipelines will now no longer take an objective parameter during instantiation, and will no longer have an objective attribute.
- `fit()` and `predict()` now use an optional `objective` parameter, which is only used in binary classification pipelines to fit for a specific objective.

- `score()` will now use a required `objectives` parameter that is used to determine all the objectives to score on. This differs from the previous behavior, where the pipeline's objective was scored on regardless.
- `score()` will now return one dictionary of all objective scores.
- `ROC` and `ConfusionMatrix` plot methods via `Auto(*).plot` have been removed by #615 and are replaced by `roc_curve` and `confusion_matrix` in *evamlm.pipelines.plot_utils*` in #704
- `normalize_confusion_matrix` has been moved to `evalml.pipelines.plot_utils` #704
- Pipelines `_name` field changed to `custom_name`
- Pipelines `supported_problem_types` field is removed because it is no longer necessary #678
- Updated argument order of objectives' *objective_function* to align with sklearn #698
- *pipeline.feature_importance_graph* has been renamed to *pipeline.graph_feature_importances* in #700
- Removed unsupported `MSLE` objective #704

v0.8.0 Apr. 1, 2020

- **Enhancements**
  - Add normalization option and information to confusion matrix #484
  - Add util function to drop rows with NaN values #487
  - Renamed *PipelineBase.name* as *PipelineBase.summary* and redefined *PipelineBase.name* as class property #491
  - Added access to parameters in Pipelines with *PipelineBase.parameters* (used to be return of *PipelineBase.describe*) #501
  - Added *fill_value* parameter for SimpleImputer #509
  - Added functionality to override component hyperparameters and made pipelines take hyperparemeters from components #516
  - Allow numpy.random.RandomState for random_state parameters #556
- **Fixes**
  - Removed unused dependency *matplotlib*, and move *category_encoders* to test reqs #572
- **Changes**
  - Undo version cap in XGBoost placed in #402 and allowed all released of XGBoost #407
  - Support pandas 1.0.0 #486
  - Made all references to the logger static #503
  - Refactored *model_type* parameter for components and pipelines to *model_family* #507
  - Refactored *problem_types* for pipelines and components into *supported_problem_types* #515
  - Moved *pipelines/utils.save_pipeline* and *pipelines/utils.load_pipeline* to *PipelineBase.save* and *PipelineBase.load* #526
  - Limit number of categories encoded by OneHotEncoder #517
- **Documentation Changes**
  - Updated API reference to remove PipelinePlot and added moved PipelineBase plotting methods #483

  - Add code style and github issue guides #463 #512
  - Updated API reference for to surface class variables for pipelines and components #537
  - Fixed README documentation link #535
  - Unhid PR references in changelog #656
- **Testing Changes**
  - Added automated dependency check PR #482, #505
  - Updated automated dependency check comment #497
  - Have build_docs job use python executor, so that env vars are set properly #547
  - Added simple test to make sure OneHotEncoder's top_n works with large number of categories #552
  - Run windows unit tests on PRs #557

Warning: Breaking Changes

- `AutoClassificationSearch` and `AutoRegressionSearch`'s `model_types` parameter has been refactored into `allowed_model_families`
- `ModelTypes` enum has been changed to `ModelFamily`
- Components and Pipelines now have a `model_family` field instead of `model_type`
- `get_pipelines` utility function now accepts `model_families` as an argument instead of `model_types`
- `PipelineBase.name` no longer returns structure of pipeline and has been replaced by `PipelineBase.summary`
- `PipelineBase.problem_types` and `Estimator.problem_types` has been renamed to `supported_problem_types`
- `pipelines/utils.save_pipeline` and `pipelines/utils.load_pipeline` moved to `PipelineBase.save` and `PipelineBase.load`

v0.7.0 Mar. 9, 2020

- **Enhancements**
  - Added emacs buffers to .gitignore #350
  - Add CatBoost (gradient-boosted trees) classification and regression components and pipelines #247
  - Added Tuner abstract base class #351
  - Added n_jobs as parameter for AutoClassificationSearch and AutoRegressionSearch #403
  - Changed colors of confusion matrix to shades of blue and updated axis order to match scikit-learn's #426
  - Added PipelineBase graph and feature_importance_graph methods, moved from previous location #423
  - Added support for python 3.8 #462
- **Fixes**

  - Fixed ROC and confusion matrix plots not being calculated if user passed own additional_objectives #276
  - Fixed ReadtheDocs FileNotFoundError exception for fraud dataset #439
- **Changes**
  - Added n_estimators as a tunable parameter for XGBoost #307
  - Remove unused parameter ObjectiveBase.fit_needs_proba #320
  - Remove extraneous parameter component_type from all components #361
  - Remove unused rankings.csv file #397
  - Downloaded demo and test datasets so unit tests can run offline #408
  - Remove *_needs_fitting* attribute from Components #398
  - Changed plot.feature_importance to show only non-zero feature importances by default, added optional parameter to show all #413
  - Refactored *PipelineBase* to take in parameter dictionary and moved pipeline metadata to class attribute #421
  - Dropped support for Python 3.5 #438
  - Removed unused *apply.py* file #449
  - Clean up requirements.txt to remove unused deps #451
  - Support installation without all required dependencies #459
- **Documentation Changes**
  - Update release.md with instructions to release to internal license key #354
- **Testing Changes**
  - Added tests for utils (and moved current utils to gen_utils) #297
  - Moved XGBoost install into it's own separate step on Windows using Conda #313
  - Rewind pandas version to before 1.0.0, to diagnose test failures for that version #325
  - Added dependency update checkin test #324
  - Rewind XGBoost version to before 1.0.0 to diagnose test failures for that version #402
  - Update dependency check to use a whitelist #417
  - Update unit test jobs to not install dev deps #455

Warning: Breaking Changes

- Python 3.5 will not be actively supported.

v0.6.0 Dec. 16, 2019

- **Enhancements**
  - Added ability to create a plot of feature importances #133
  - Add early stopping to AutoML using patience and tolerance parameters #241
  - Added ROC and confusion matrix metrics and plot for classification problems and introduce PipelineSearchPlots class #242

  - Enhanced AutoML results with search order #260
  - Added utility function to show system and environment information #300
- **Fixes**
  - Lower botocore requirement #235
  - Fixed decision_function calculation for FraudCost objective #254
  - Fixed return value of Recall metrics #264
  - Components return *self* on fit #289
- **Changes**
  - Renamed automl classes to AutoRegressionSearch and AutoClassificationSearch #287
  - Updating demo datasets to retain column names #223
  - Moving pipeline visualization to PipelinePlots class #228
  - Standarizing inputs as pd.Dataframe / pd.Series #130
  - Enforcing that pipelines must have an estimator as last component #277
  - Added ipywidgets as a dependency in requirements.txt #278
  - Added Random and Grid Search Tuners #240
- **Documentation Changes**
  - Adding class properties to API reference #244
  - Fix and filter FutureWarnings from scikit-learn #249, #257
  - Adding Linear Regression to API reference and cleaning up some Sphinx warnings #227
- **Testing Changes**
  - Added support for testing on Windows with CircleCI #226
  - Added support for doctests #233

Warning: Breaking Changes

- The `fit()` method for `AutoClassifier` and `AutoRegressor` has been renamed to `search()`.
- `AutoClassifier` has been renamed to `AutoClassificationSearch`
- `AutoRegressor` has been renamed to `AutoRegressionSearch`
- `AutoClassificationSearch.results` and `AutoRegressionSearch.results` now is a dictionary with `pipeline_results` and `search_order` keys. `pipeline_results` can be used to access a dictionary that is identical to the old `.results` dictionary. Whereas, `search_order` returns a list of the search order in terms of `pipeline_id`.
- Pipelines now require an estimator as the last component in `component_list`. Slicing pipelines now throws an `NotImplementedError` to avoid returning pipelines without an estimator.

v0.5.2 Nov. 18, 2019

- **Enhancements**
  - Adding basic pipeline structure visualization #211
- **Documentation Changes**

- Added notebooks to build process #212

v0.5.1 Nov. 15, 2019

- **Enhancements**
  - Added basic outlier detection guardrail #151
  - Added basic ID column guardrail #135
  - Added support for unlimited pipelines with a max_time limit #70
  - Updated .readthedocs.yaml to successfully build #188
- **Fixes**
  - Removed MSLE from default additional objectives #203
  - Fixed random_state passed in pipelines #204
  - Fixed slow down in RFRegressor #206
- **Changes**
  - Pulled information for describe_pipeline from pipeline's new describe method #190
  - Refactored pipelines #108
  - Removed guardrails from Auto(*) #202, #208
- **Documentation Changes**
  - Updated documentation to show max_time enhancements #189
  - Updated release instructions for RTD #193
  - Added notebooks to build process #212
  - Added contributing instructions #213
  - Added new content #222

v0.5.0 Oct. 29, 2019

- **Enhancements**
  - Added basic one hot encoding #73
  - Use enums for model_type #110
  - Support for splitting regression datasets #112
  - Auto-infer multiclass classification #99
  - Added support for other units in max_time #125
  - Detect highly null columns #121
  - Added additional regression objectives #100
  - Show an interactive iteration vs. score plot when using fit() #134
- **Fixes**
  - Reordered *describe_pipeline* #94
  - Added type check for model_type #109
  - Fixed *s* units when setting string max_time #132
  - Fix objectives not appearing in API documentation #150

- **Changes**
  - Reorganized tests #93
  - Moved logging to its own module #119
  - Show progress bar history #111
  - Using cloudpickle instead of pickle to allow unloading of custom objectives #113
  - Removed render.py #154
- **Documentation Changes**
  - Update release instructions #140
  - Include additional_objectives parameter #124
  - Added Changelog #136
- **Testing Changes**
  - Code coverage #90
  - Added CircleCI tests for other Python versions #104
  - Added doc notebooks as tests #139
  - Test metadata for CircleCI and 2 core parallelism #137

v0.4.1 Sep. 16, 2019

- **Enhancements**
  - Added AutoML for classification and regressor using Autobase and Skopt #7 #9
  - Implemented standard classification and regression metrics #7
  - Added logistic regression, random forest, and XGBoost pipelines #7
  - Implemented support for custom objectives #15
  - Feature importance for pipelines #18
  - Serialization for pipelines #19
  - Allow fitting on objectives for optimal threshold #27
  - Added detect label leakage #31
  - Implemented callbacks #42
  - Allow for multiclass classification #21
  - Added support for additional objectives #79
- **Fixes**
  - Fixed feature selection in pipelines #13
  - Made random_seed usage consistent #45
- **Documentation Changes**
  - Documentation Changes
  - Added docstrings #6
  - Created notebooks for docs #6
  - Initialized readthedocs EvalML #6

- Added favicon #38

- **Testing Changes**
  - Added testing for loading data #39

v0.2.0 Aug. 13, 2019

- **Enhancements**
  - Created fraud detection objective #4

v0.1.0 July. 31, 2019

- *First Release*
- **Enhancements**
  - Added lead scoring objecitve #1
  - Added basic classifier #1
- **Documentation Changes**
  - Initialized Sphinx for docs #1

1.6.15 API Reference

Demo Datasets

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

evalml.demos.load_fraud

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

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

Parameters

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

Returns X, y

Return type pd.DataFrame, pd.Series

evalml.demos.load_wine

`evalml.demos.load_wine()`

Load wine dataset. Multiclass problem

Returns X, y

Return type pd.DataFrame, pd.Series

evalml.demos.load_breast_cancer

`evalml.demos.load_breast_cancer()`

Load breast cancer dataset. Multiclass problem

Returns X, y

Return type pd.DataFrame, pd.Series

evalml.demos.load_diabetes

`evalml.demos.load_diabetes()`

Load diabetes dataset. Regression problem

Returns X, y

Return type pd.DataFrame, pd.Series

Preprocessing

Utilities to preprocess data before using evalml.

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

evalml.preprocessing.drop_nan_target_rows

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

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

Parameters

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

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

Return type pd.DataFrame

evalml.preprocessing.label_distribution

`evalml.preprocessing.label_distribution(labels)`

Get the label distributions

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

Returns Label values and their frequency distribution as percentages.

Return type pd.Series

evalml.preprocessing.load_data

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

Load features and labels from file.

Parameters

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

Returns features and labels

Return type pd.DataFrame, pd.Series

evalml.preprocessing.number_of_features

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

Get the number of features for specific dtypes

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

Returns dtypes and the number of features for each input type

Return type pd.Series

evalml.preprocessing.split_data

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

Splits data into train and test sets.

Parameters

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

Returns features and labels each split into train and test sets

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

AutoML

AutoClassificationSearch	Automatic pipeline search class for classification problems
AutoRegressionSearch	Automatic pipeline search for regression problems

evalml.automl.AutoClassificationSearch

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

Automatic pipeline search class for classification problems

Methods

__init__	Automated classifier pipeline search
describe_pipeline	Describe a pipeline
get_pipeline	Retrieves trained pipeline
search	Find best classifier

evalml.automl.AutoClassificationSearch.__init__

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

Automated classifier pipeline search

Parameters

- **objective** (*Object*) – The objective to optimize for. Defaults to LogLossBinary for binary classification problems and LogLossMulticlass for multiclass classification problems.

- **multiclass** (*bool*) – If True, expecting multiclass data. Defaults to False.
- **max_pipelines** (*int*) – Maximum number of pipelines to search. If max_pipelines and max_time is not set, then max_pipelines will default to max_pipelines of 5.
- **max_time** (*int, str*) – Maximum time to search for pipelines. This will not start a new pipeline search after the duration has elapsed. If it is an integer, then the time will be in seconds. For strings, time can be specified as seconds, minutes, or hours.
- **patience** (*int*) – Number of iterations without improvement to stop search early. Must be positive. If None, early stopping is disabled. Defaults to None.
- **tolerance** (*float*) – Minimum percentage difference to qualify as score improvement for early stopping. Only applicable if patience is not None. Defaults to None.
- **allowed_model_families** (*list*) – The model families to search. By default, searches over all model families. Run evalml.list_model_families("binary") to see options. Change *binary* to *multiclass* if your problem type is different.
- **cv** – cross-validation method to use. Defaults to StratifiedKFold.
- **tuner** – the tuner class to use. Defaults to scikit-optimize tuner
- **detect_label_leakage** (*bool*) – If True, check input features for label leakage and warn if found. Defaults to true.
- **start_iteration_callback** (*callable*) – function called before each pipeline training iteration. Passed two parameters: pipeline_class, parameters.
- **add_result_callback** (*callable*) – function called after each pipeline training iteration. Passed two parameters: results, trained_pipeline.
- **additional_objectives** (*list*) – Custom set of objectives to score on. Will override default objectives for problem type if not empty.
- **random_state** (*int, np.random.RandomState*) – The random seed/state. Defaults to 0.
- **n_jobs** (*int or None*) – Non-negative integer describing level of parallelism used for pipelines. None and 1 are equivalent. If set to -1, all CPUs are used. For n_jobs below -1, (n_cpus + 1 + n_jobs) are used.
- **verbose** (*boolean*) – If True, turn verbosity on. Defaults to True

evalml.automl.AutoClassificationSearch.describe_pipeline

AutoClassificationSearch.**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.AutoClassificationSearch.get_pipeline

AutoClassificationSearch.**get_pipeline**(*pipeline_id*)

Retrieves trained pipeline

Parameters **pipeline_id** (*int*) – pipeline to retrieve

Returns pipeline associated with id

Return type Pipeline

evalml.automl.AutoClassificationSearch.search

AutoClassificationSearch.**search**(*X*, *y*, *feature_types=None*, *raise_errors=True*, *show_iteration_plot=True*)

Find best classifier

Parameters

- **X** (*pd.DataFrame*) – the input training data of shape [n_samples, n_features]
- **y** (*pd.Series*) – the target training labels of length [n_samples]
- **feature_types** (*list, optional*) – list of feature types, either numerical or categorical. Categorical features will automatically be encoded
- **raise_errors** (*boolean*) – If True, raise errors and exit search if a pipeline errors during fitting. If False, set scores for the errored pipeline to NaN and continue search. Defaults to True.
- **show_iteration_plot** (*boolean, True*) – Shows an iteration vs. score plot in Jupyter notebook. Disabled by default in non-Jupyter enviroments.

Returns self

Attributes

best_pipeline	Returns the best model found
full_rankings	Returns a pandas.DataFrame with scoring results from all pipelines searched
rankings	Returns a pandas.DataFrame with scoring results from the highest-scoring set of parameters used with each pipeline.

evalml.automl.AutoRegressionSearch

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

Automatic pipeline search for regression problems

Methods

`__init__`	Automated regressors pipeline search
`describe_pipeline`	Describe a pipeline
`get_pipeline`	Retrieves trained pipeline
`search`	Find best classifier

evalml.automl.AutoRegressionSearch.__init__

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

Automated regressors pipeline search

Parameters

- **objective** (*Object*) – The objective to optimize for. Defaults to R2.
- **max_pipelines** (*int*) – Maximum number of pipelines to search. If max_pipelines and max_time is not set, then max_pipelines will default to max_pipelines of 5.
- **max_time** (*int, str*) – Maximum time to search for pipelines. This will not start a new pipeline search after the duration has elapsed. If it is an integer, then the time will be in seconds. For strings, time can be specified as seconds, minutes, or hours.
- **allowed_model_families** (*list*) – The model families to search. By default searches over all model families. Run evalml.list_model_families("regression") to see options.
- **patience** (*int*) – Number of iterations without improvement to stop search early. Must be positive. If None, early stopping is disabled. Defaults to None.
- **tolerance** (*float*) – Minimum percentage difference to qualify as score improvement for early stopping. Only applicable if patience is not None. Defaults to None.
- **cv** – cross validation method to use. By default StratifiedKFold
- **tuner** – the tuner class to use. Defaults to scikit-optimize tuner
- **detect_label_leakage** (*bool*) – If True, check input features for label leakage and warn if found. Defaults to true.
- **start_iteration_callback** (*callable*) – function called before each pipeline training iteration. Passed two parameters: pipeline_class, parameters.

- **add_result_callback** (*callable*) – function called after each pipeline training iteration. Passed two parameters: results, trained_pipeline.
- **additional_objectives** (*list*) – Custom set of objectives to score on. Will override default objectives for problem type if not empty.
- **random_state** (*int, np.random.RandomState*) – The random seed/state. Defaults to 0.
- **n_jobs** (*int or None*) – Non-negative integer describing level of parallelism used for pipelines. None and 1 are equivalent. If set to -1, all CPUs are used. For n_jobs below -1, (n_cpus + 1 + n_jobs) are used.
- **verbose** (*boolean*) – If True, turn verbosity on. Defaults to True

evalml.automl.AutoRegressionSearch.describe_pipeline

AutoRegressionSearch.**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.AutoRegressionSearch.get_pipeline

AutoRegressionSearch.**get_pipeline**(*pipeline_id*)
Retrieves trained pipeline

Parameters **pipeline_id** (*int*) – pipeline to retrieve

Returns pipeline associated with id

Return type Pipeline

evalml.automl.AutoRegressionSearch.search

AutoRegressionSearch.**search**(*X*, *y*, *feature_types=None*, *raise_errors=True*, *show_iteration_plot=True*)
Find best classifier

Parameters

- **X** (*pd.DataFrame*) – the input training data of shape [n_samples, n_features]
- **y** (*pd.Series*) – the target training labels of length [n_samples]
- **feature_types** (*list, optional*) – list of feature types, either numerical or categorical. Categorical features will automatically be encoded
- **raise_errors** (*boolean*) – If True, raise errors and exit search if a pipeline errors during fitting. If False, set scores for the errored pipeline to NaN and continue search. Defaults to True.

- **show_iteration_plot** (*boolean, True*) – Shows an iteration vs. score plot in Jupyter notebook. Disabled by default in non-Jupyter enviroments.

Returns self

Attributes

best_pipeline	Returns the best model found
full_rankings	Returns a pandas.DataFrame with scoring results from all pipelines searched
rankings	Returns a pandas.DataFrame with scoring results from the highest-scoring set of parameters used with each pipeline.

Pipelines

Pipeline Base Classes

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

evalml.pipelines.PipelineBase

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

Methods

__init__	Machine learning pipeline made out of transformers and a estimator.
describe	Outputs pipeline details including component parameters
fit	Build a model

Continued on next page

Table 9 – continued from previous page

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 importances
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 objects 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.describe

PipelineBase.**describe**()

Outputs pipeline details including component parameters

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

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

Return type dict

evalml.pipelines.PipelineBase.fit

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

Build a model

Parameters

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

Returns self

evalml.pipelines.PipelineBase.get_component

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

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

Returns component to return

Return type Component

evalml.pipelines.PipelineBase.graph

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

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

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

Return type graphviz.Digraph

evalml.pipelines.PipelineBase.graph_feature_importance

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

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

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

evalml.pipelines.PipelineBase.load

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

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

Returns PipelineBase obj

evalml.pipelines.PipelineBase.predict

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

Parameters

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

Returns estimated labels

Return type pd.Series

evalml.pipelines.PipelineBase.save

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

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

Returns None

evalml.pipelines.PipelineBase.score

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

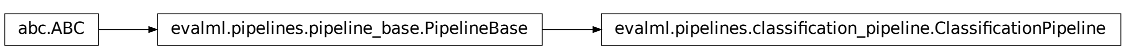
Parameters

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

Returns ordered dictionary of objective scores

Return type dict

evalml.pipelines.ClassificationPipeline



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

Methods

__init__	Machine learning pipeline made out of transformers and a estimator.
describe	Outputs pipeline details including component parameters
fit	Build a model
get_component	Returns component by name
graph	Generate an image representing the pipeline graph
graph_feature_importance	Generate a bar graph of the pipeline's feature importances
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

Continued on next page

Table 10 – continued from previous page

score	Evaluate model performance on current and additional objectives

evalml.pipelines.ClassificationPipeline.__init__

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

Machine learning pipeline made out of transformers and a estimator.

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

`ClassificationPipeline.describe`()

Outputs pipeline details including component parameters

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

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

Return type dict

evalml.pipelines.ClassificationPipeline.fit

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

Build a model

Parameters

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

Returns self

evalml.pipelines.ClassificationPipeline.get_component

`ClassificationPipeline.get_component`(*name*)

Returns component by name

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

Returns component to return

Return type Component

evalml.pipelines.ClassificationPipeline.graph

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

Generate an image representing the pipeline graph

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

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

Return type graphviz.Digraph

evalml.pipelines.ClassificationPipeline.graph_feature_importance

`ClassificationPipeline.`**`graph_feature_importance`**(*show_all_features=False*)

Generate a bar graph of the pipeline's feature importances

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

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

evalml.pipelines.ClassificationPipeline.load

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

Loads pipeline at file path

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

Returns PipelineBase obj

evalml.pipelines.ClassificationPipeline.predict

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

Make predictions using selected features.

Parameters

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

Returns estimated labels

Return type pd.Series

evalml.pipelines.ClassificationPipeline.predict_proba

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

Make probability estimates for labels.

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

Returns probability estimates

Return type pd.DataFrame

evalml.pipelines.ClassificationPipeline.save

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

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

Returns None

evalml.pipelines.ClassificationPipeline.score

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

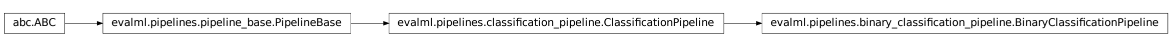
Parameters

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

Returns ordered dictionary of objective scores

Return type dict

evalml.pipelines.BinaryClassificationPipeline



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

Methods

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

evalml.pipelines.BinaryClassificationPipeline.__init__

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

Machine learning pipeline made out of transformers and a estimator.

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

`BinaryClassificationPipeline.describe()`

Outputs pipeline details including component parameters

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

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

Return type dict

evalml.pipelines.BinaryClassificationPipeline.fit

`BinaryClassificationPipeline.fit(X, y)`

Build a model

Parameters

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

Returns self

evalml.pipelines.BinaryClassificationPipeline.get_component

`BinaryClassificationPipeline.get_component(name)`

Returns component by name

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

Returns component to return

Return type Component

evalml.pipelines.BinaryClassificationPipeline.graph

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

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

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

Return type graphviz.Digraph

evalml.pipelines.BinaryClassificationPipeline.graph_feature_importance

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

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

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

evalml.pipelines.BinaryClassificationPipeline.load

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

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

Returns PipelineBase obj

evalml.pipelines.BinaryClassificationPipeline.predict

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

Parameters

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

Returns estimated labels

Return type pd.Series

evalml.pipelines.BinaryClassificationPipeline.predict_proba

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

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

Returns probability estimates

Return type pd.DataFrame

evalml.pipelines.BinaryClassificationPipeline.save

`BinaryClassificationPipeline.`**`save`**(*file_path*)

Saves pipeline at file path

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

Returns None

evalml.pipelines.BinaryClassificationPipeline.score

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

Evaluate model performance on objectives

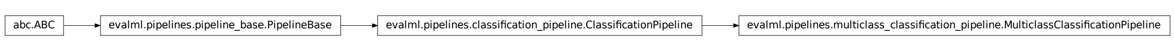
Parameters

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

Returns ordered dictionary of objective scores

Return type dict

evalml.pipelines.MulticlassClassificationPipeline



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

Pipeline subclass for all multiclass classification pipelines.

Methods

`__init__`	Machine learning pipeline made out of transformers and a estimator.
`describe`	Outputs pipeline details including component parameters
`fit`	Build a model
`get_component`	Returns component by name
`graph`	Generate an image representing the pipeline graph
`graph_feature_importance`	Generate a bar graph of the pipeline's feature importances
`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

Continued on next page

Table 12 – continued from previous page

score	Evaluate model performance on current and additional objectives

evalml.pipelines.MulticlassClassificationPipeline.__init__

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

Machine learning pipeline made out of transformers and a estimator.

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

MulticlassClassificationPipeline.**describe**()

Outputs pipeline details including component parameters

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

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

Return type dict

evalml.pipelines.MulticlassClassificationPipeline.fit

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

Build a model

Parameters

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

Returns self

evalml.pipelines.MulticlassClassificationPipeline.get_component

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

Returns component by name

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

Returns component to return

Return type Component

evalml.pipelines.MulticlassClassificationPipeline.graph

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

Generate an image representing the pipeline graph

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

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

Return type graphviz.Digraph

evalml.pipelines.MulticlassClassificationPipeline.graph_feature_importance

`MulticlassClassificationPipeline.`**graph_feature_importance**(*show_all_features=False*)

Generate a bar graph of the pipeline's feature importances

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

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

evalml.pipelines.MulticlassClassificationPipeline.load

static `MulticlassClassificationPipeline.`**load**(*file_path*)

Loads pipeline at file path

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

Returns PipelineBase obj

evalml.pipelines.MulticlassClassificationPipeline.predict

`MulticlassClassificationPipeline.`**predict**(*X, objective=None*)

Make predictions using selected features.

Parameters

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

Returns estimated labels

Return type pd.Series

evalml.pipelines.MulticlassClassificationPipeline.predict_proba

`MulticlassClassificationPipeline.`**predict_proba**(*X*)

Make probability estimates for labels.

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

Returns probability estimates

Return type pd.DataFrame

evalml.pipelines.MulticlassClassificationPipeline.save

`MulticlassClassificationPipeline.`**`save`**(*file_path*)

Saves pipeline at file path

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

Returns None

evalml.pipelines.MulticlassClassificationPipeline.score

`MulticlassClassificationPipeline.`**`score`**(*X*, *y*, *objectives*)

Evaluate model performance on current and additional objectives

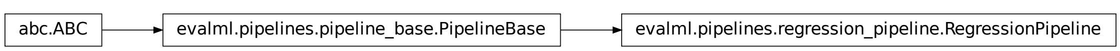
Parameters

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

Returns ordered dictionary of objective scores

Return type dict

evalml.pipelines.RegressionPipeline



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

Pipeline subclass for all regression pipelines.

Methods

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

Continued on next page

Table 13 – continued from previous page

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 objects 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.describe

RegressionPipeline.**describe**()
Outputs pipeline details including component parameters

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

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

Return type dict

evalml.pipelines.RegressionPipeline.fit

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

Parameters

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

Returns self

evalml.pipelines.RegressionPipeline.get_component

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

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

Returns component to return

Return type Component

evalml.pipelines.RegressionPipeline.graph

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

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

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

Return type graphviz.Digraph

evalml.pipelines.RegressionPipeline.graph_feature_importance

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

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

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

evalml.pipelines.RegressionPipeline.load

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

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

Returns PipelineBase obj

evalml.pipelines.RegressionPipeline.predict

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

Parameters

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

Returns estimated labels

Return type pd.Series

evalml.pipelines.RegressionPipeline.save

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

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

Returns None

evalml.pipelines.RegressionPipeline.score

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

Parameters

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

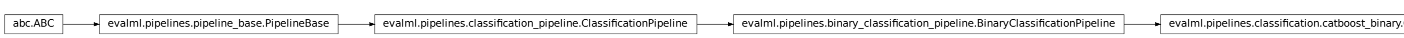
Returns ordered dictionary of objective scores

Return type dict

Classification Pipelines

CatBoostBinaryClassificationPipeline	CatBoost Pipeline for binary classification.
CatBoostMulticlassClassificationPipeline	CatBoost Pipeline for multiclass classification.
LogisticRegressionBinaryPipeline	Logistic Regression Pipeline for binary classification
LogisticRegressionMulticlassPipeline	Logistic Regression Pipeline for multiclass classification
RFBinaryClassificationPipeline	Random Forest Pipeline for binary classification
RFMulticlassClassificationPipeline	Random Forest Pipeline for multiclass classification
XGBoostBinaryPipeline	XGBoost Pipeline for binary classification
XGBoostMulticlassPipeline	XGBoost Pipeline for multiclass classification

evalml.pipelines.CatBoostBinaryClassificationPipeline



class evalml.pipelines.**CatBoostBinaryClassificationPipeline**(*parameters*, *random_state=0*)
CatBoost Pipeline for binary classification. CatBoost is an open-source library and natively supports categorical features.

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

name = 'Cat Boost Binary Classification Pipeline'

custom_name = None

summary = 'CatBoost Classifier w/ Simple Imputer'

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

problem_type = 'binary'

model_family = 'catboost'

hyperparameters = {'eta': Real(low=0, high=1, prior='uniform', transform='identity'),

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

Instance attributes

feature_importances	Return feature importances.
parameters	Returns parameter dictionary for this pipeline
threshold	

Methods:

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

evalml.pipelines.CatBoostBinaryClassificationPipeline.__init__

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

Machine learning pipeline made out of transformers and a estimator.

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

Parameters

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

evalml.pipelines.CatBoostBinaryClassificationPipeline.describe

CatBoostBinaryClassificationPipeline.**describe**()

Outputs pipeline details including component parameters

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

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

Return type dict

evalml.pipelines.CatBoostBinaryClassificationPipeline.fit

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

Parameters

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

Returns self

evalml.pipelines.CatBoostBinaryClassificationPipeline.get_component

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

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

Returns component to return

Return type Component

evalml.pipelines.CatBoostBinaryClassificationPipeline.graph

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

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

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

Return type graphviz.Digraph

evalml.pipelines.CatBoostBinaryClassificationPipeline.graph_feature_importance

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

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

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

evalml.pipelines.CatBoostBinaryClassificationPipeline.load

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

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

Returns PipelineBase obj

evalml.pipelines.CatBoostBinaryClassificationPipeline.predict

`CatBoostBinaryClassificationPipeline.`**`predict`**(*X*, *objective=None*)

Make predictions using selected features.

Parameters

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

Returns estimated labels

Return type pd.Series

evalml.pipelines.CatBoostBinaryClassificationPipeline.predict_proba

`CatBoostBinaryClassificationPipeline.`**`predict_proba`**(*X*)

Make probability estimates for labels.

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

Returns probability estimates

Return type pd.DataFrame

evalml.pipelines.CatBoostBinaryClassificationPipeline.save

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

Saves pipeline at file path

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

Returns None

evalml.pipelines.CatBoostBinaryClassificationPipeline.score

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

Evaluate model performance on objectives

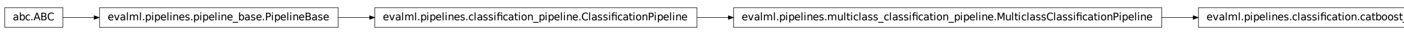
Parameters

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

Returns ordered dictionary of objective scores

Return type dict

evalml.pipelines.CatBoostMulticlassClassificationPipeline



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

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

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

`name = 'Cat Boost Multiclass Classification Pipeline'`

`custom_name = None`

`summary = 'CatBoost Classifier w/ Simple Imputer'`

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

`problem_type = 'multiclass'`

`model_family = 'catboost'`

`hyperparameters = {'eta': Real(low=0, high=1, prior='uniform', transform='identity'),`

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

Instance attributes

`feature_importances`	Return feature importances.
`parameters`	Returns parameter dictionary for this pipeline

Methods:

`__init__`	Machine learning pipeline made out of transformers and a estimator.
`describe`	Outputs pipeline details including component parameters
`fit`	Build a model
`get_component`	Returns component by name
`graph`	Generate an image representing the pipeline graph
`graph_feature_importance`	Generate a bar graph of the pipeline's feature importances
`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 current and additional objectives

evalml.pipelines.CatBoostMulticlassClassificationPipeline.__init__

CatBoostMulticlassClassificationPipeline.**__init__**(*parameters*, *random_state=0*)
Machine learning pipeline made out of transformers and a estimator.

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

Parameters

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

evalml.pipelines.CatBoostMulticlassClassificationPipeline.describe

CatBoostMulticlassClassificationPipeline.**describe**()
Outputs pipeline details including component parameters

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

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

Return type dict

evalml.pipelines.CatBoostMulticlassClassificationPipeline.fit

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

Parameters

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

Returns self

evalml.pipelines.CatBoostMulticlassClassificationPipeline.get_component

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

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

Returns component to return

Return type Component

evalml.pipelines.CatBoostMulticlassClassificationPipeline.graph

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

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

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

Return type graphviz.Digraph

evalml.pipelines.CatBoostMulticlassClassificationPipeline.graph_feature_importance

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

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

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

evalml.pipelines.CatBoostMulticlassClassificationPipeline.load

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

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

Returns PipelineBase obj

evalml.pipelines.CatBoostMulticlassClassificationPipeline.predict

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

Parameters

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

Returns estimated labels

Return type pd.Series

evalml.pipelines.CatBoostMulticlassClassificationPipeline.predict_proba

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

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

Returns probability estimates

Return type pd.DataFrame

evalml.pipelines.CatBoostMulticlassClassificationPipeline.save

`CatBoostMulticlassClassificationPipeline.`**save**(*file_path*)

Saves pipeline at file path

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

Returns None

evalml.pipelines.CatBoostMulticlassClassificationPipeline.score

`CatBoostMulticlassClassificationPipeline.`**score**(*X*, *y*, *objectives*)

Evaluate model performance on current and additional objectives

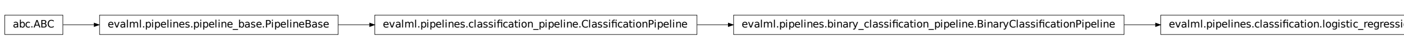
Parameters

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

Returns ordered dictionary of objective scores

Return type dict

evalml.pipelines.LogisticRegressionBinaryPipeline



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

Logistic Regression Pipeline for binary classification

name = 'Logistic Regression Binary Pipeline'

custom_name = None

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

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

problem_type = 'binary'

model_family = 'linear_model'

hyperparameters = {'C': Real(low=0.01, high=10, prior='uniform', transform='identity')

custom_hyperparameters = None

Instance attributes

`feature_importances`	Return feature importances.
`parameters`	Returns parameter dictionary for this pipeline

Continued on next page

Table 19 – continued from previous page

threshold

Methods:

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

evalml.pipelines.LogisticRegressionBinaryPipeline.__init__

`LogisticRegressionBinaryPipeline.__init__`(*parameters*, *random_state=0*)

Machine learning pipeline made out of transformers and a estimator.

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

Parameters

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

evalml.pipelines.LogisticRegressionBinaryPipeline.describe

`LogisticRegressionBinaryPipeline.describe`()

Outputs pipeline details including component parameters

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

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

Return type dict

evalml.pipelines.LogisticRegressionBinaryPipeline.fit

`LogisticRegressionBinaryPipeline.fit`(*X*, *y*)

Build a model

Parameters

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

Returns self

evalml.pipelines.LogisticRegressionBinaryPipeline.get_component

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

Returns component by name

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

Returns component to return

Return type Component

evalml.pipelines.LogisticRegressionBinaryPipeline.graph

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

Generate an image representing the pipeline graph

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

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

Return type graphviz.Digraph

evalml.pipelines.LogisticRegressionBinaryPipeline.graph_feature_importance

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

Generate a bar graph of the pipeline's feature importances

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

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

evalml.pipelines.LogisticRegressionBinaryPipeline.load

static LogisticRegressionBinaryPipeline.**load**(*file_path*)

Loads pipeline at file path

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

Returns PipelineBase obj

evalml.pipelines.LogisticRegressionBinaryPipeline.predict

`LogisticRegressionBinaryPipeline.predict(X, objective=None)`
Make predictions using selected features.

Parameters

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

Returns estimated labels

Return type pd.Series

evalml.pipelines.LogisticRegressionBinaryPipeline.predict_proba

`LogisticRegressionBinaryPipeline.predict_proba(X)`
Make probability estimates for labels.

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

Returns probability estimates

Return type pd.DataFrame

evalml.pipelines.LogisticRegressionBinaryPipeline.save

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

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

Returns None

evalml.pipelines.LogisticRegressionBinaryPipeline.score

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

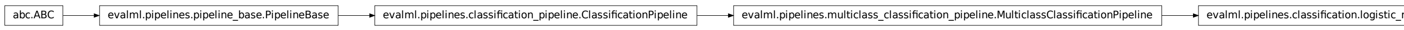
Parameters

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

Returns ordered dictionary of objective scores

Return type dict

evalml.pipelines.LogisticRegressionMulticlassPipeline



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

Logistic Regression Pipeline for multiclass classification

name = 'Logistic Regression Multiclass Pipeline'

custom_name = None

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

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

problem_type = 'multiclass'

model_family = 'linear_model'

hyperparameters = {'C': Real(low=0.01, high=10, prior='uniform', transform='identity')

custom_hyperparameters = None

Instance attributes

feature_importances	Return feature importances.
parameters	Returns parameter dictionary for this pipeline

Methods:

__init__	Machine learning pipeline made out of transformers and a estimator.
describe	Outputs pipeline details including component parameters
fit	Build a model
get_component	Returns component by name
graph	Generate an image representing the pipeline graph
graph_feature_importance	Generate a bar graph of the pipeline's feature importances
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 current and additional objectives

evalml.pipelines.LogisticRegressionMulticlassPipeline.__init__

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

Machine learning pipeline made out of transformers and a estimator.

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

Parameters

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

evalml.pipelines.LogisticRegressionMulticlassPipeline.describe

LogisticRegressionMulticlassPipeline.**describe**()

Outputs pipeline details including component parameters

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

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

Return type dict

evalml.pipelines.LogisticRegressionMulticlassPipeline.fit

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

Build a model

Parameters

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

Returns self

evalml.pipelines.LogisticRegressionMulticlassPipeline.get_component

LogisticRegressionMulticlassPipeline.**get_component**(*name*)

Returns component by name

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

Returns component to return

Return type Component

evalml.pipelines.LogisticRegressionMulticlassPipeline.graph

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

Generate an image representing the pipeline graph

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

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

Return type graphviz.Digraph

evalml.pipelines.LogisticRegressionMulticlassPipeline.graph_feature_importance

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

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

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

evalml.pipelines.LogisticRegressionMulticlassPipeline.load

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

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

Returns PipelineBase obj

evalml.pipelines.LogisticRegressionMulticlassPipeline.predict

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

Parameters

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

Returns estimated labels

Return type pd.Series

evalml.pipelines.LogisticRegressionMulticlassPipeline.predict_proba

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

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

Returns probability estimates

Return type pd.DataFrame

evalml.pipelines.LogisticRegressionMulticlassPipeline.save

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

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

Returns None

evalml.pipelines.LogisticRegressionMulticlassPipeline.score

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

Evaluate model performance on current and additional objectives

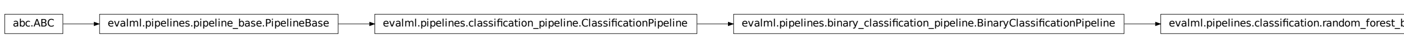
Parameters

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

Returns ordered dictionary of objective scores

Return type dict

evalml.pipelines.RFBinaryClassificationPipeline



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

Random Forest Pipeline for binary classification

name = 'Random Forest Binary Classification Pipeline'

custom_name = 'Random Forest Binary Classification Pipeline'

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

component_graph = ['One Hot Encoder', 'Simple Imputer', 'RF Classifier Select From Mod

problem_type = 'binary'

model_family = 'random_forest'

hyperparameters = {'impute_strategy': ['mean', 'median', 'most_frequent'], 'max_depth

custom_hyperparameters = None

Instance attributes

`feature_importances`	Return feature importances.
`parameters`	Returns parameter dictionary for this pipeline
`threshold`	

Methods:

`__init__`	Machine learning pipeline made out of transformers and a estimator.
`describe`	Outputs pipeline details including component parameters

Continued on next page

Table 24 – 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 importances
load	Loads pipeline at file path
predict	Make predictions using selected features.
predict_proba	Make probability estimates for labels.
save	Saves pipeline at file path
score	Evaluate model performance on objectives

evalml.pipelines.RFBinaryClassificationPipeline.__init__

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

Machine learning pipeline made out of transformers and a estimator.

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

Parameters

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

evalml.pipelines.RFBinaryClassificationPipeline.describe

RFBinaryClassificationPipeline.**describe**()

Outputs pipeline details including component parameters

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

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

Return type dict

evalml.pipelines.RFBinaryClassificationPipeline.fit

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

Build a model

Parameters

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

Returns self

evalml.pipelines.RFBinaryClassificationPipeline.get_component

`RFBinaryClassificationPipeline.`**`get_component`**`(`*name*`)`

Returns component by name

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

Returns component to return

Return type Component

evalml.pipelines.RFBinaryClassificationPipeline.graph

`RFBinaryClassificationPipeline.`**`graph`**`(`*filepath=None*`)`

Generate an image representing the pipeline graph

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

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

Return type graphviz.Digraph

evalml.pipelines.RFBinaryClassificationPipeline.graph_feature_importance

`RFBinaryClassificationPipeline.`**`graph_feature_importance`**`(`*show_all_features=False*`)`

Generate a bar graph of the pipeline's feature importances

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

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

evalml.pipelines.RFBinaryClassificationPipeline.load

`static` `RFBinaryClassificationPipeline.`**`load`**`(`*file_path*`)`

Loads pipeline at file path

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

Returns PipelineBase obj

evalml.pipelines.RFBinaryClassificationPipeline.predict

`RFBinaryClassificationPipeline.`**`predict`**`(`*X*, *objective=None*`)`

Make predictions using selected features.

Parameters

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

Returns estimated labels

Return type pd.Series

evalml.pipelines.RFBinaryClassificationPipeline.predict_proba

`RFBinaryClassificationPipeline.`**`predict_proba`**(*X*)

Make probability estimates for labels.

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

Returns probability estimates

Return type pd.DataFrame

evalml.pipelines.RFBinaryClassificationPipeline.save

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

Saves pipeline at file path

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

Returns None

evalml.pipelines.RFBinaryClassificationPipeline.score

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

Evaluate model performance on objectives

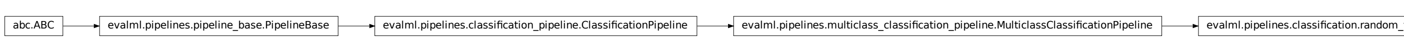
Parameters

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

Returns ordered dictionary of objective scores

Return type dict

evalml.pipelines.RFMulticlassClassificationPipeline



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

Random Forest Pipeline for multiclass classification

name = 'Random Forest Multiclass Classification Pipeline'

custom_name = 'Random Forest Multiclass Classification Pipeline'

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

component_graph = ['One Hot Encoder', 'Simple Imputer', 'RF Classifier Select From Mod

problem_type = 'multiclass'

model_family = 'random_forest'

hyperparameters = {'impute_strategy': ['mean', 'median', 'most_frequent'], 'max_depth

custom_hyperparameters = None

Instance attributes

feature_importances	Return feature importances.
parameters	Returns parameter dictionary for this pipeline

Methods:

__init__	Machine learning pipeline made out of transformers and a estimator.
describe	Outputs pipeline details including component parameters
fit	Build a model
get_component	Returns component by name
graph	Generate an image representing the pipeline graph
graph_feature_importance	Generate a bar graph of the pipeline's feature importances
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 current and additional objectives

evalml.pipelines.RFMulticlassClassificationPipeline.__init__

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

Machine learning pipeline made out of transformers and a estimator.

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

Parameters

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

evalml.pipelines.RFMulticlassClassificationPipeline.describe

RFMulticlassClassificationPipeline.**describe**()

Outputs pipeline details including component parameters

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

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

Return type dict

evalml.pipelines.RFMulticlassClassificationPipeline.fit

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

Build a model

Parameters

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

Returns self

evalml.pipelines.RFMulticlassClassificationPipeline.get_component

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

Returns component by name

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

Returns component to return

Return type Component

evalml.pipelines.RFMulticlassClassificationPipeline.graph

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

Generate an image representing the pipeline graph

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

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

Return type graphviz.Digraph

evalml.pipelines.RFMulticlassClassificationPipeline.graph_feature_importance

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

Generate a bar graph of the pipeline's feature importances

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

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

evalml.pipelines.RFMulticlassClassificationPipeline.load

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

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

Returns PipelineBase obj

evalml.pipelines.RFMulticlassClassificationPipeline.predict

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

Parameters

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

Returns estimated labels

Return type pd.Series

evalml.pipelines.RFMulticlassClassificationPipeline.predict_proba

`RFMulticlassClassificationPipeline.`**predict_proba**(*X*)
Make probability estimates for labels.

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

Returns probability estimates

Return type pd.DataFrame

evalml.pipelines.RFMulticlassClassificationPipeline.save

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

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

Returns None

evalml.pipelines.RFMulticlassClassificationPipeline.score

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

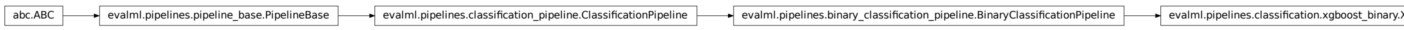
Parameters

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

Returns ordered dictionary of objective scores

Return type dict

evalml.pipelines.XGBoostBinaryPipeline



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

XGBoost Pipeline for binary classification

`name = 'XGBoost Binary Classification Pipeline'`

`custom_name = 'XGBoost Binary Classification Pipeline'`

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

`component_graph = ['One Hot Encoder', 'Simple Imputer', 'RF Classifier Select From Mod`

`problem_type = 'binary'`

`model_family = 'xgboost'`

`hyperparameters = {'eta': Real(low=0, high=1, prior='uniform', transform='identity'),`

`custom_hyperparameters = None`

Instance attributes

`feature_importances`	Return feature importances.
`parameters`	Returns parameter dictionary for this pipeline
`threshold`	

Methods:

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

evalml.pipelines.XGBoostBinaryPipeline.__init__

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

Machine learning pipeline made out of transformers and a estimator.

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

Parameters

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

evalml.pipelines.XGBoostBinaryPipeline.describe

XGBoostBinaryPipeline.**describe**()

Outputs pipeline details including component parameters

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

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

Return type dict

evalml.pipelines.XGBoostBinaryPipeline.fit

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

Build a model

Parameters

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

Returns self

evalml.pipelines.XGBoostBinaryPipeline.get_component

XGBoostBinaryPipeline.**get_component**(*name*)

Returns component by name

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

Returns component to return

Return type Component

evalml.pipelines.XGBoostBinaryPipeline.graph

`XGBoostBinaryPipeline.graph(filepath=None)`
Generate an image representing the pipeline graph

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

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

Return type graphviz.Digraph

evalml.pipelines.XGBoostBinaryPipeline.graph_feature_importance

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

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

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

evalml.pipelines.XGBoostBinaryPipeline.load

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

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

Returns PipelineBase obj

evalml.pipelines.XGBoostBinaryPipeline.predict

`XGBoostBinaryPipeline.predict(X, objective=None)`
Make predictions using selected features.

Parameters

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

Returns estimated labels

Return type pd.Series

evalml.pipelines.XGBoostBinaryPipeline.predict_proba

`XGBoostBinaryPipeline.predict_proba(X)`
Make probability estimates for labels.

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

Returns probability estimates

Return type pd.DataFrame

evalml.pipelines.XGBoostBinaryPipeline.save

XGBoostBinaryPipeline.**save**(*file_path*)

Saves pipeline at file path

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

Returns None

evalml.pipelines.XGBoostBinaryPipeline.score

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

Evaluate model performance on objectives

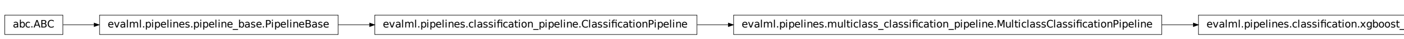
Parameters

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

Returns ordered dictionary of objective scores

Return type dict

evalml.pipelines.XGBoostMulticlassPipeline



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

XGBoost Pipeline for multiclass classification

name = 'XGBoost Multiclass Classification Pipeline'

custom_name = 'XGBoost Multiclass Classification Pipeline'

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

component_graph = ['One Hot Encoder', 'Simple Imputer', 'RF Classifier Select From Mode

problem_type = 'multiclass'

model_family = 'xgboost'

hyperparameters = {'eta': Real(low=0, high=1, prior='uniform', transform='identity'),

custom_hyperparameters = None

Instance attributes

feature_importances	Return feature importances.
parameters	Returns parameter dictionary for this pipeline

Methods:

`__init__`	Machine learning pipeline made out of transformers and a estimator.
`describe`	Outputs pipeline details including component parameters
`fit`	Build a model
`get_component`	Returns component by name
`graph`	Generate an image representing the pipeline graph
`graph_feature_importance`	Generate a bar graph of the pipeline's feature importances
`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 current and additional objectives

evalml.pipelines.XGBoostMulticlassPipeline.__init__

`XGBoostMulticlassPipeline.__init__`(*parameters*, *random_state=0*)

Machine learning pipeline made out of transformers and a estimator.

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

Parameters

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

evalml.pipelines.XGBoostMulticlassPipeline.describe

`XGBoostMulticlassPipeline.describe`()

Outputs pipeline details including component parameters

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

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

Return type dict

evalml.pipelines.XGBoostMulticlassPipeline.fit

`XGBoostMulticlassPipeline.fit`(*X*, *y*)

Build a model

Parameters

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

Returns self

evalml.pipelines.XGBoostMulticlassPipeline.get_component

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

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

Returns component to return

Return type Component

evalml.pipelines.XGBoostMulticlassPipeline.graph

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

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

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

Return type graphviz.Digraph

evalml.pipelines.XGBoostMulticlassPipeline.graph_feature_importance

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

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

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

evalml.pipelines.XGBoostMulticlassPipeline.load

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

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

Returns PipelineBase obj

evalml.pipelines.XGBoostMulticlassPipeline.predict

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

Parameters

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

Returns estimated labels

Return type pd.Series

evalml.pipelines.XGBoostMulticlassPipeline.predict_proba

XGBoostMulticlassPipeline.**predict_proba**(*X*)

Make probability estimates for labels.

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

Returns probability estimates

Return type pd.DataFrame

evalml.pipelines.XGBoostMulticlassPipeline.save

XGBoostMulticlassPipeline.**save**(*file_path*)

Saves pipeline at file path

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

Returns None

evalml.pipelines.XGBoostMulticlassPipeline.score

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

Evaluate model performance on current and additional objectives

Parameters

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

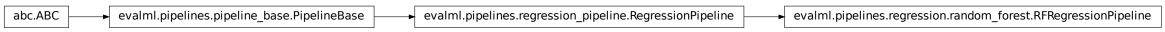
Returns ordered dictionary of objective scores

Return type dict

Regression Pipelines

RFRegressionPipeline	Random Forest Pipeline for regression problems
CatBoostRegressionPipeline	CatBoost Pipeline for regression problems.
LinearRegressionPipeline	Linear Regression Pipeline for regression problems
XGBoostRegressionPipeline	XGBoost Pipeline for regression problems

evalml.pipelines.RFRegressionPipeline



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

Random Forest Pipeline for regression problems

name = 'Random Forest Regression Pipeline'

custom_name = 'Random Forest Regression Pipeline'

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

component_graph = ['One Hot Encoder', 'Simple Imputer', 'RF Regressor Select From Mode

problem_type = 'regression'

model_family = 'random_forest'

hyperparameters = {'impute_strategy': ['mean', 'median', 'most_frequent'], 'max_depth

custom_hyperparameters = None

Instance attributes

feature_importances	Return feature importances.
parameters	Returns parameter dictionary for this pipeline

Methods:

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

evalml.pipelines.RFRegressionPipeline.__init__

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

Machine learning pipeline made out of transformers and a estimator.

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

ComponentBase objects in the list

Parameters

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

evalml.pipelines.RFRegressionPipeline.describe

`RFRegressionPipeline.describe()`

Outputs pipeline details including component parameters

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

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

Return type dict

evalml.pipelines.RFRegressionPipeline.fit

`RFRegressionPipeline.fit(X, y)`

Build a model

Parameters

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

Returns self

evalml.pipelines.RFRegressionPipeline.get_component

`RFRegressionPipeline.get_component(name)`

Returns component by name

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

Returns component to return

Return type Component

evalml.pipelines.RFRegressionPipeline.graph

`RFRegressionPipeline.graph(filepath=None)`

Generate an image representing the pipeline graph

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

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

Return type graphviz.Digraph

evalml.pipelines.RFRegressionPipeline.graph_feature_importance

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

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

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

evalml.pipelines.RFRegressionPipeline.load

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

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

Returns PipelineBase obj

evalml.pipelines.RFRegressionPipeline.predict

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

Parameters

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

Returns estimated labels

Return type pd.Series

evalml.pipelines.RFRegressionPipeline.save

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

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

Returns None

evalml.pipelines.RFRegressionPipeline.score

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

Parameters

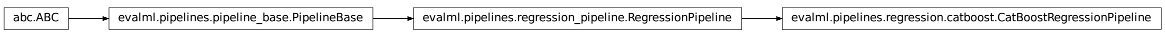
- **X** (*pd.DataFrame or np.array*) – data of shape [n_samples, n_features]
- **y** (*pd.Series*) – true labels of length [n_samples]

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

Returns ordered dictionary of objective scores

Return type dict

evalml.pipelines.CatBoostRegressionPipeline



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

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

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

Note: impute_strategy must support both string and numeric data

name = 'Cat Boost Regression Pipeline'

custom_name = None

summary = 'CatBoost Regressor w/ Simple Imputer'

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

problem_type = 'regression'

model_family = 'catboost'

hyperparameters = {'eta': Real(low=0, high=1, prior='uniform', transform='identity'),

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

Instance attributes

feature_importances	Return feature importances.
parameters	Returns parameter dictionary for this pipeline

Methods:

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

Continued on next page

Table 35 – continued from previous page

save	Saves pipeline at file path
score	Evaluate model performance on current and additional objectives

evalml.pipelines.CatBoostRegressionPipeline.__init__

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

Machine learning pipeline made out of transformers and a estimator.

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

Parameters

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

evalml.pipelines.CatBoostRegressionPipeline.describe

CatBoostRegressionPipeline.**describe**()

Outputs pipeline details including component parameters

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

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

Return type dict

evalml.pipelines.CatBoostRegressionPipeline.fit

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

Build a model

Parameters

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

Returns self

evalml.pipelines.CatBoostRegressionPipeline.get_component

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

Returns component by name

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

Returns component to return

Return type Component

evalml.pipelines.CatBoostRegressionPipeline.graph

`CatBoostRegressionPipeline.`**`graph`**(*filepath=None*)

Generate an image representing the pipeline graph

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

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

Return type graphviz.Digraph

evalml.pipelines.CatBoostRegressionPipeline.graph_feature_importance

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

Generate a bar graph of the pipeline's feature importances

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

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

evalml.pipelines.CatBoostRegressionPipeline.load

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

Loads pipeline at file path

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

Returns PipelineBase obj

evalml.pipelines.CatBoostRegressionPipeline.predict

`CatBoostRegressionPipeline.`**`predict`**(*X, objective=None*)

Make predictions using selected features.

Parameters

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

Returns estimated labels

Return type pd.Series

evalml.pipelines.CatBoostRegressionPipeline.save

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

Saves pipeline at file path

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

Returns None

evalml.pipelines.CatBoostRegressionPipeline.score

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

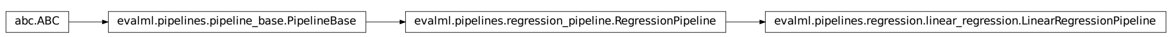
Parameters

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

Returns ordered dictionary of objective scores

Return type dict

evalml.pipelines.LinearRegressionPipeline



class evalml.pipelines.**LinearRegressionPipeline**(*parameters*, *random_state=0*)
Linear Regression Pipeline for regression problems

name = 'Linear Regression Pipeline'

custom_name = None

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

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

problem_type = 'regression'

model_family = 'linear_model'

hyperparameters = {'fit_intercept': [True, False], 'impute_strategy': ['mean', 'medi

custom_hyperparameters = None

Instance attributes

feature_importances	Return feature importances.
parameters	Returns parameter dictionary for this pipeline

Methods:

__init__	Machine learning pipeline made out of transformers and a estimator.
describe	Outputs pipeline details including component parameters
fit	Build a model
get_component	Returns component by name

Continued on next page

Table 37 – continued from previous page

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

evalml.pipelines.LinearRegressionPipeline.__init__

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

Machine learning pipeline made out of transformers and a estimator.

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

Parameters

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

evalml.pipelines.LinearRegressionPipeline.describe

LinearRegressionPipeline.**describe**()

Outputs pipeline details including component parameters

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

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

Return type dict

evalml.pipelines.LinearRegressionPipeline.fit

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

Build a model

Parameters

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

Returns self

evalml.pipelines.LinearRegressionPipeline.get_component

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

Returns component by name

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

Returns component to return

Return type Component

evalml.pipelines.LinearRegressionPipeline.graph

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

Generate an image representing the pipeline graph

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

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

Return type graphviz.Digraph

evalml.pipelines.LinearRegressionPipeline.graph_feature_importance

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

Generate a bar graph of the pipeline's feature importances

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

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

evalml.pipelines.LinearRegressionPipeline.load

static LinearRegressionPipeline.**load**(*file_path*)

Loads pipeline at file path

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

Returns PipelineBase obj

evalml.pipelines.LinearRegressionPipeline.predict

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

Make predictions using selected features.

Parameters

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

Returns estimated labels

Return type pd.Series

evalml.pipelines.LinearRegressionPipeline.save

`LinearRegressionPipeline.`**`save`**(*file_path*)

Saves pipeline at file path

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

Returns None

evalml.pipelines.LinearRegressionPipeline.score

`LinearRegressionPipeline.`**`score`**(*X*, *y*, *objectives*)

Evaluate model performance on current and additional objectives

Parameters

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

Returns ordered dictionary of objective scores

Return type dict

evalml.pipelines.XGBoostRegressionPipeline

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

XGBoost Pipeline for regression problems

`name = 'XGBoost Regression Pipeline'`

`custom_name = None`

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

`component_graph = ['One Hot Encoder', 'Simple Imputer', 'RF Regressor Select From Mode`

`problem_type = 'regression'`

`model_family = 'xgboost'`

`hyperparameters = {'eta': Real(low=0, high=1, prior='uniform', transform='identity'),`

`custom_hyperparameters = None`

Instance attributes

`feature_importances`	Return feature importances.
`parameters`	Returns parameter dictionary for this pipeline

Methods:

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

evalml.pipelines.XGBoostRegressionPipeline.__init__

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

Machine learning pipeline made out of transformers and a estimator.

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

Parameters

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

evalml.pipelines.XGBoostRegressionPipeline.describe

XGBoostRegressionPipeline.**describe**()

Outputs pipeline details including component parameters

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

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

Return type dict

evalml.pipelines.XGBoostRegressionPipeline.fit

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

Build a model

Parameters

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

Returns self

evalml.pipelines.XGBoostRegressionPipeline.get_component

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

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

Returns component to return

Return type Component

evalml.pipelines.XGBoostRegressionPipeline.graph

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

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

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

Return type graphviz.Digraph

evalml.pipelines.XGBoostRegressionPipeline.graph_feature_importance

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

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

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

evalml.pipelines.XGBoostRegressionPipeline.load

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

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

Returns PipelineBase obj

evalml.pipelines.XGBoostRegressionPipeline.predict

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

Parameters

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

Returns estimated labels

Return type pd.Series

evalml.pipelines.XGBoostRegressionPipeline.save

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

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

Returns None

evalml.pipelines.XGBoostRegressionPipeline.score

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

Parameters

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

Returns ordered dictionary of objective scores

Return type dict

Pipeline Utils

all_pipelines	Returns a complete list of all supported pipeline classes.
get_pipelines	Returns the pipelines allowed for a particular problem type.
list_model_families	List model type for a particular problem type

evalml.pipelines.all_pipelines

evalml.pipelines.**all_pipelines**()
Returns a complete list of all supported pipeline classes.

Returns a list of pipeline classes

Return type list[*PipelineBase*]

evalml.pipelines.get_pipelines

evalml.pipelines.**get_pipelines**(*problem_type*, *model_families=None*)
Returns the pipelines allowed for a particular problem type.

Can also optionally filter by a list of model types.

Arguments:

Returns a list of pipeline classes

Return type list[*PipelineBase*]

evalml.pipelines.list_model_families

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

List model type for a particular problem type

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

Returns a list of model families

Return type list[*ModelFamily*]

Pipeline Plot Utils

roc_curve	Receiver Operating Characteristic score for binary classification.
confusion_matrix	Confusion matrix for binary and multiclass classification.
normalize_confusion_matrix	Normalizes a confusion matrix.

evalml.pipelines.roc_curve

evalml.pipelines.**roc_curve**(*y_true*, *y_pred_proba*)

Receiver Operating Characteristic score for binary classification.

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.

Returns false positive rates, true positive rates, and threshold values used to produce each pair of true/false positive rates.

Return type (np.array, np.array, np.array)

evalml.pipelines.confusion_matrix

evalml.pipelines.**confusion_matrix**(*y_true*, *y_predicted*)

Confusion matrix for binary and multiclass classification.

Parameters

- **y_true** (*pd.Series or np.array*) – true binary labels.
- **y_predicted** (*pd.Series or np.array*) – predictions from a binary classifier, before thresholding has been applied.

Returns confusion matrix

Return type np.array

evalml.pipelines.normalize_confusion_matrix

evalml.pipelines.**normalize_confusion_matrix**(*conf_mat*, *option='true'*)
Normalizes a confusion matrix.

Parameters

- **conf_mat** (*pd.DataFrame or np.array*) – confusion matrix to normalize
- **option** (*{'true', 'pred', 'all'}*) – Option to normalize over the rows ('true'), columns ('pred') or all ('all') values. Defaults to 'true'.

Returns A normalized version of the input confusion matrix.

Components

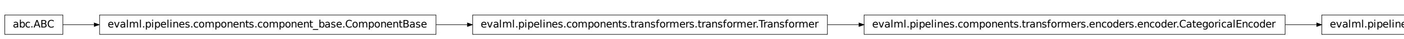
Transformers

Encoders

Encoders convert categorical or non-numerical features into numerical features.

OneHotEncoder	One-hot encoder to encode non-numeric data

evalml.pipelines.components.OneHotEncoder



class evalml.pipelines.components.**OneHotEncoder**(*top_n=10*, *random_state=0*)
One-hot encoder to encode non-numeric data

name = 'One Hot Encoder'

model_family = 'none'

hyperparameter_ranges = {}

Instance attributes

—

Methods:

__init__	Initalizes self.
describe	Describe a component and its parameters
fit	Fits component to data

Continued on next page

Table 44 – continued from previous page

fit_transform	Fits on X and transforms X
get_feature_names	Returns names of transformed and added columns
transform	One-hot encode the input DataFrame.

evalml.pipelines.components.OneHotEncoder.__init__

OneHotEncoder.**__init__**(*top_n=10*, *random_state=0*)
Initalizes self.

evalml.pipelines.components.OneHotEncoder.describe

OneHotEncoder.**describe**(*print_name=False*, *return_dict=False*)
Describe a component and its parameters

Parameters

- **print_name** (*bool, optional*) – whether to print name of component
- **return_dict** (*bool, optional*) – whether to return description as dictionary in the format {"name": name, "parameters": parameters}

Returns prints and returns dictionary

Return type None or dict

evalml.pipelines.components.OneHotEncoder.fit

OneHotEncoder.**fit**(*X*, *y=None*)
Fits component to data

Parameters

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

Returns self

evalml.pipelines.components.OneHotEncoder.fit_transform

OneHotEncoder.**fit_transform**(*X*, *y=None*)
Fits on X and transforms X

Parameters

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

Returns Transformed X

Return type pd.DataFrame

evalml.pipelines.components.OneHotEncoder.get_feature_names

`OneHotEncoder.get_feature_names()`

Returns names of transformed and added columns

Returns list of feature names not including dropped features

Return type list

evalml.pipelines.components.OneHotEncoder.transform

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

One-hot encode the input DataFrame.

Parameters

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

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

Imputers

Imputers fill in missing data.

SimpleImputer	Imputes missing data according to a specified imputation strategy

evalml.pipelines.components.SimpleImputer

class `evalml.pipelines.components.SimpleImputer(impute_strategy='most_frequent', fill_value=None, random_state=0)`

Imputes missing data according to a specified imputation strategy

name = 'Simple Imputer'

model_family = 'none'

hyperparameter_ranges = {'impute_strategy': ['mean', 'median', 'most_frequent']}

Instance attributes

—

Methods:

`__init__`	Initalizes an transformer that imputes missing data according to the specified imputation strategy."
`describe`	Describe a component and its parameters
`fit`	Fits component to data
`fit_transform`	Fits imputer on data X then imputes missing values in X
`transform`	Transforms data X by imputing missing values

evalml.pipelines.components.SimpleImputer.__init__

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

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.describe

SimpleImputer.**describe**(*print_name=False*, *return_dict=False*)

Describe a component and its parameters

Parameters

- **print_name** (*bool, optional*) – whether to print name of component
- **return_dict** (*bool, optional*) – whether to return description as dictionary in the format {"name": name, "parameters": parameters}

Returns prints and returns dictionary

Return type None or dict

evalml.pipelines.components.SimpleImputer.fit

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

Fits component to data

Parameters

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

Returns self

evalml.pipelines.components.SimpleImputer.fit_transform

`SimpleImputer.fit_transform(X, y=None)`
Fits imputer on data X then imputes missing values in X

Parameters

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

Returns Transformed X

Return type pd.DataFrame

evalml.pipelines.components.SimpleImputer.transform

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

Parameters

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

Returns Transformed X

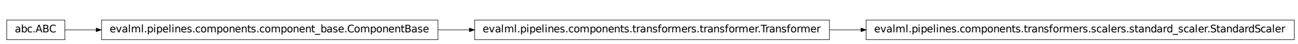
Return type pd.DataFrame

Scalers

Scalers transform and standardize the range of data.

StandardScaler	Standardize features: removes mean and scales to unit variance

evalml.pipelines.components.StandardScaler



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

name = 'Standard Scaler'

model_family = 'none'

hyperparameter_ranges = {}

Instance attributes

—

Methods:

`__init__`	Initialize self.
`describe`	Describe a component and its parameters
`fit`	Fits component to data
`fit_transform`	Fits on X and transforms X
`transform`	Transforms data X

evalml.pipelines.components.StandardScaler.__init__

StandardScaler.**__init__**(*random_state=0*)

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

evalml.pipelines.components.StandardScaler.describe

StandardScaler.**describe**(*print_name=False*, *return_dict=False*)

Describe a component and its parameters

Parameters

- **print_name** (*bool, optional*) – whether to print name of component
- **return_dict** (*bool, optional*) – whether to return description as dictionary in the format {"name": name, "parameters": parameters}

Returns prints and returns dictionary

Return type None or dict

evalml.pipelines.components.StandardScaler.fit

StandardScaler.**fit**(*X*, *y=None*)

Fits component to data

Parameters

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

Returns self

evalml.pipelines.components.StandardScaler.fit_transform

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

Fits on X and transforms X

Parameters

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

Returns Transformed X

Return type pd.DataFrame

evalml.pipelines.components.StandardScaler.transform

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

Transforms data X

Parameters

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

Returns Transformed X

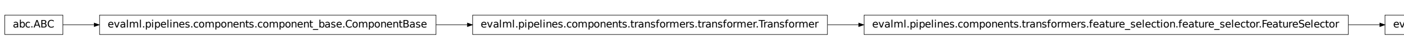
Return type pd.DataFrame

Feature Selectors

Feature selectors select a subset of relevant features for the model.

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

evalml.pipelines.components.RFRegressorSelectFromModel



class `evalml.pipelines.components.`**`RFRegressorSelectFromModel`**(*number_features=None*, *n_estimators=10*, *max_depth=None*, *percent_features=0.5*, *threshold=-inf*, *n_jobs=-1*, *random_state=0*)

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

name = 'RF Regressor Select From Model'

model_family = 'none'

hyperparameter_ranges = {'percent_features': Real(low=0.01, high=1, prior='uniform',

Instance attributes

—

Methods:

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

evalml.pipelines.components.RFRegressorSelectFromModel.__init__

`RFRegressorSelectFromModel.`**`__init__`**(*number_features=None*, *n_estimators=10*, *max_depth=None*, *percent_features=0.5*, *threshold=-inf*, *n_jobs=-1*, *random_state=0*)

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

evalml.pipelines.components.RFRegressorSelectFromModel.describe

`RFRegressorSelectFromModel.`**`describe`**(*print_name=False*, *return_dict=False*)

Describe a component and its parameters

Parameters

- **`print_name`** (*`bool, optional`*) – whether to print name of component
- **`return_dict`** (*`bool, optional`*) – whether to return description as dictionary in the format {“name”: name, “parameters”: parameters}

Returns prints and returns dictionary

Return type None or dict

evalml.pipelines.components.RFRegressorSelectFromModel.fit

`RFRegressorSelectFromModel.`**`fit`**(*X*, *y=None*)

Fits component to data

Parameters

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

Returns self

evalml.pipelines.components.RFRegressorSelectFromModel.fit_transform

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

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

Parameters

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

Returns Transformed X

Return type pd.DataFrame

evalml.pipelines.components.RFRegressorSelectFromModel.get_indices

RFRegressorSelectFromModel.**get_indices**()

Get integer index of features selected

Returns list of indices

Return type list

evalml.pipelines.components.RFRegressorSelectFromModel.get_names

RFRegressorSelectFromModel.**get_names**()

Get names of selected features.

Returns list of the names of features selected

evalml.pipelines.components.RFRegressorSelectFromModel.transform

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

Transforms data X by selecting features

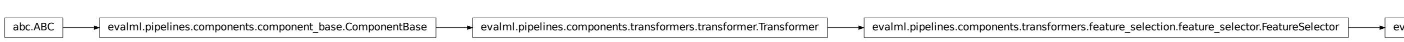
Parameters

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

Returns Transformed X

Return type pd.DataFrame

evalml.pipelines.components.RFClassifierSelectFromModel



class evalml.pipelines.components.**RFClassifierSelectFromModel**(*number_features=None*, *n_estimators=10*, *max_depth=None*, *percent_features=0.5*, *threshold=-inf*, *n_jobs=-1*, *random_state=0*)

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

name = 'RF Classifier Select From Model'

model_family = 'none'

hyperparameter_ranges = {'percent_features': Real(low=0.01, high=1, prior='uniform', t

Instance attributes

—

Methods:

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

evalml.pipelines.components.RFClassifierSelectFromModel.__init__

RFClassifierSelectFromModel.**__init__**(*number_features=None*, *n_estimators=10*, *max_depth=None*, *percent_features=0.5*, *threshold=-inf*, *n_jobs=-1*, *random_state=0*)

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

evalml.pipelines.components.RFClassifierSelectFromModel.describe

RFClassifierSelectFromModel.**describe**(*print_name=False*, *return_dict=False*)

Describe a component and its parameters

Parameters

- **print_name** (*bool, optional*) – whether to print name of component
- **return_dict** (*bool, optional*) – whether to return description as dictionary in the format {"name": name, "parameters": parameters}

Returns prints and returns dictionary

Return type None or dict

evalml.pipelines.components.RFClassifierSelectFromModel.fit

RFClassifierSelectFromModel.**fit**(*X*, *y=None*)

Fits component to data

Parameters

- **X** (*pd.DataFrame or np.array*) – the input training data of shape [n_samples, n_features]

- **y** (*pd.Series, optional*) – the target training labels of length [n_samples]

Returns self

evalml.pipelines.components.RFClassifierSelectFromModel.fit_transform

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

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

Parameters

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

Returns Transformed X

Return type pd.DataFrame

evalml.pipelines.components.RFClassifierSelectFromModel.get_indices

`RFClassifierSelectFromModel.`**`get_indices`**`()`

Get integer index of features selected

Returns list of indices

Return type list

evalml.pipelines.components.RFClassifierSelectFromModel.get_names

`RFClassifierSelectFromModel.`**`get_names`**`()`

Get names of selected features.

Returns list of the names of features selected

evalml.pipelines.components.RFClassifierSelectFromModel.transform

`RFClassifierSelectFromModel.`**`transform`**`(`*X*, *y=None*`)`

Transforms data X by selecting features

Parameters

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

Returns Transformed X

Return type pd.DataFrame

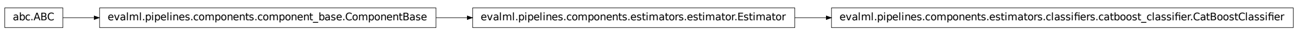
Estimators

Classifiers

Classifiers are models which can be trained to predict a class label from input data.

CatBoostClassifier	CatBoost Classifier, a classifier that uses gradient-boosting on decision trees.
RandomForestClassifier	Random Forest Classifier
LogisticRegressionClassifier	Logistic Regression Classifier
XGBoostClassifier	XGBoost Classifier

evalml.pipelines.components.CatBoostClassifier



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

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=0, high=1, prior='uniform', transform='ident

Instance attributes

SEED_MAX	
SEED_MIN	
feature_importances	Returns feature importances.

Methods:

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

evalml.pipelines.components.CatBoostClassifier.__init__

CatBoostClassifier.**__init__**(*n_estimators=1000*, *eta=0.03*, *max_depth=6*, *bootstrap_type=None*, *random_state=0*)

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

evalml.pipelines.components.CatBoostClassifier.describe

`CatBoostClassifier.describe(print_name=False, return_dict=False)`

Describe a component and its parameters

Parameters

- **print_name** (*bool, optional*) – whether to print name of component
- **return_dict** (*bool, optional*) – whether to return description as dictionary in the format {"name": name, "parameters": parameters}

Returns prints and returns dictionary

Return type None or dict

evalml.pipelines.components.CatBoostClassifier.fit

`CatBoostClassifier.fit(X, y=None)`

Fits component to data

Parameters

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

Returns self

evalml.pipelines.components.CatBoostClassifier.predict

`CatBoostClassifier.predict(X)`

Make predictions using selected features.

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

Returns estimated labels

Return type pd.Series

evalml.pipelines.components.CatBoostClassifier.predict_proba

`CatBoostClassifier.predict_proba(X)`

Make probability estimates for labels.

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

Returns probability estimates

Return type pd.DataFrame

evalml.pipelines.components.RandomForestClassifier

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

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=32, prior='uniform', transfo

Instance attributes

feature_importances	Returns feature importances.

Methods:

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

evalml.pipelines.components.RandomForestClassifier.__init__

RandomForestClassifier.**__init__**(*n_estimators=10*, *max_depth=None*, *n_jobs=-1*, *random_state=0*)

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

evalml.pipelines.components.RandomForestClassifier.describe

RandomForestClassifier.**describe**(*print_name=False*, *return_dict=False*)

Describe a component and its parameters

Parameters

- **print_name** (*bool, optional*) – whether to print name of component
- **return_dict** (*bool, optional*) – whether to return description as dictionary in the format {"name": name, "parameters": parameters}

Returns prints and returns dictionary

Return type None or dict

evalml.pipelines.components.RandomForestClassifier.fit

RandomForestClassifier.**fit**(*X*, *y=None*)
Fits component to data

Parameters

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

Returns self

evalml.pipelines.components.RandomForestClassifier.predict

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

Parameters X (*pd.DataFrame*) – features

Returns estimated labels

Return type pd.Series

evalml.pipelines.components.RandomForestClassifier.predict_proba

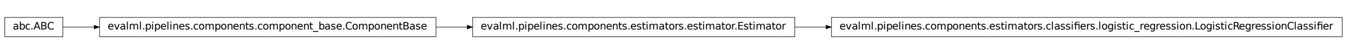
RandomForestClassifier.**predict_proba**(*X*)
Make probability estimates for labels.

Parameters X (*pd.DataFrame*) – features

Returns probability estimates

Return type pd.DataFrame

evalml.pipelines.components.LogisticRegressionClassifier



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

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

Instance attributes

`feature_importances`	Returns feature importances.

Methods:

`__init__`	Initialize self.
`describe`	Describe a component and its parameters
`fit`	Fits component to data
`predict`	Make predictions using selected features.
`predict_proba`	Make probability estimates for labels.

evalml.pipelines.components.LogisticRegressionClassifier.__init__

LogisticRegressionClassifier.**__init__**(*penalty='l2'*, *C=1.0*, *n_jobs=-1*, *random_state=0*)

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

evalml.pipelines.components.LogisticRegressionClassifier.describe

LogisticRegressionClassifier.**describe**(*print_name=False*, *return_dict=False*)

Describe a component and its parameters

Parameters

- **print_name** (*bool, optional*) – whether to print name of component
- **return_dict** (*bool, optional*) – whether to return description as dictionary in the format {"name": name, "parameters": parameters}

Returns prints and returns dictionary

Return type None or dict

evalml.pipelines.components.LogisticRegressionClassifier.fit

LogisticRegressionClassifier.**fit**(*X*, *y=None*)

Fits component to data

Parameters

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

Returns self

evalml.pipelines.components.LogisticRegressionClassifier.predict

LogisticRegressionClassifier.**predict**(*X*)

Make predictions using selected features.

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

Returns estimated labels

Return type pd.Series

evalml.pipelines.components.LogisticRegressionClassifier.predict_proba

`LogisticRegressionClassifier.`**`predict_proba`**`(`*X*`)`

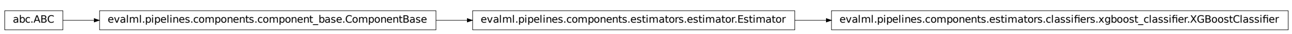
Make probability estimates for labels.

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

Returns probability estimates

Return type pd.DataFrame

evalml.pipelines.components.XGBoostClassifier



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

XGBoost Classifier

name = 'XGBoost Classifier'

model_family = 'xgboost'

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

hyperparameter_ranges = {'eta': Real(low=0, high=1, prior='uniform', transform='ident

Instance attributes

SEED_MAX	
SEED_MIN	
feature_importances	Returns feature importances.

Methods:

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

evalml.pipelines.components.XGBoostClassifier.__init__

XGBoostClassifier.**__init__**(*eta=0.1*, *max_depth=3*, *min_child_weight=1*, *n_estimators=100*, *random_state=0*)

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

evalml.pipelines.components.XGBoostClassifier.describe

XGBoostClassifier.**describe**(*print_name=False*, *return_dict=False*)

Describe a component and its parameters

Parameters

- **print_name** (*bool, optional*) – whether to print name of component
- **return_dict** (*bool, optional*) – whether to return description as dictionary in the format {"name": name, "parameters": parameters}

Returns prints and returns dictionary

Return type None or dict

evalml.pipelines.components.XGBoostClassifier.fit

XGBoostClassifier.**fit**(*X*, *y=None*)

Fits component to data

Parameters

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

Returns self

evalml.pipelines.components.XGBoostClassifier.predict

XGBoostClassifier.**predict**(*X*)

Make predictions using selected features.

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

Returns estimated labels

Return type pd.Series

evalml.pipelines.components.XGBoostClassifier.predict_proba

XGBoostClassifier.**predict_proba**(*X*)

Make probability estimates for labels.

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

Returns probability estimates

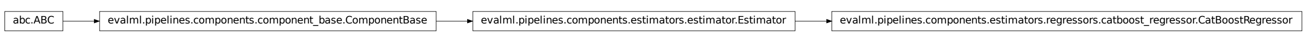
Return type pd.DataFrame

Regressors

Regressors are models which can be trained to predict a target value from input data.

CatBoostRegressor	CatBoost Regressor, a regressor that uses gradient-boosting on decision trees.
LinearRegressor	Linear Regressor
RandomForestRegressor	Random Forest Regressor
XGBoostRegressor	XGBoost Regressor

evalml.pipelines.components.CatBoostRegressor



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

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=0, high=1, prior='uniform', transform='ident

Instance attributes

SEED_MAX	
SEED_MIN	
feature_importances	Returns feature importances.

Methods:

__init__	Initialize self.
describe	Describe a component and its parameters
fit	Build a model
predict	Make predictions using selected features.
predict_proba	Make probability estimates for labels.

evalml.pipelines.components.CatBoostRegressor.__init__

CatBoostRegressor.**__init__**(*n_estimators=1000*, *eta=0.03*, *max_depth=6*, *bootstrap_type=None*, *random_state=0*)
Initialize self. See help(type(self)) for accurate signature.

evalml.pipelines.components.CatBoostRegressor.describe

CatBoostRegressor.**describe**(*print_name=False*, *return_dict=False*)
Describe a component and its parameters

Parameters

- **print_name** (*bool, optional*) – whether to print name of component
- **return_dict** (*bool, optional*) – whether to return description as dictionary in the format {"name": name, "parameters": parameters}

Returns prints and returns dictionary

Return type None or dict

evalml.pipelines.components.CatBoostRegressor.fit

CatBoostRegressor.**fit**(*X*, *y=None*)
Build a model

Parameters

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

Returns self

evalml.pipelines.components.CatBoostRegressor.predict

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

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

Returns estimated labels

Return type pd.Series

evalml.pipelines.components.CatBoostRegressor.predict_proba

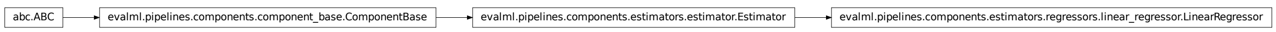
CatBoostRegressor.**predict_proba**(*X*)
Make probability estimates for labels.

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

Returns probability estimates

Return type pd.DataFrame

evalml.pipelines.components.LinearRegressor



class evalml.pipelines.components.**LinearRegressor**(*fit_intercept=True*, *normalize=False*, *n_jobs=-1*, *random_state=0*)

Linear Regressor

name = 'Linear Regressor'

model_family = 'linear_model'

supported_problem_types = [<ProblemTypes.REGRESSION: 'regression'>]

hyperparameter_ranges = {'fit_intercept': [True, False], 'normalize': [True, False]}

Instance attributes

feature_importances	Returns feature importances.

Methods:

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

evalml.pipelines.components.LinearRegressor.__init__

LinearRegressor.**__init__**(*fit_intercept=True*, *normalize=False*, *n_jobs=-1*, *random_state=0*)

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

evalml.pipelines.components.LinearRegressor.describe

LinearRegressor.**describe**(*print_name=False*, *return_dict=False*)

Describe a component and its parameters

Parameters

- **print_name** (*bool, optional*) – whether to print name of component
- **return_dict** (*bool, optional*) – whether to return description as dictionary in the format {"name": name, "parameters": parameters}

Returns prints and returns dictionary

Return type None or dict

evalml.pipelines.components.LinearRegressor.fit

`LinearRegressor.fit(X, y=None)`

Fits component to data

Parameters

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

Returns self

evalml.pipelines.components.LinearRegressor.predict

`LinearRegressor.predict(X)`

Make predictions using selected features.

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

Returns estimated labels

Return type pd.Series

evalml.pipelines.components.LinearRegressor.predict_proba

`LinearRegressor.predict_proba(X)`

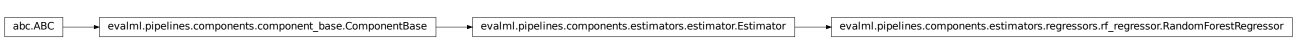
Make probability estimates for labels.

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

Returns probability estimates

Return type pd.DataFrame

evalml.pipelines.components.RandomForestRegressor



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

Random Forest Regressor

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

Instance attributes

feature_importances	Returns feature importances.

Methods:

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

evalml.pipelines.components.RandomForestRegressor.__init__

RandomForestRegressor.**__init__**(*n_estimators=10*, *max_depth=None*, *n_jobs=-1*, *random_state=0*)

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

evalml.pipelines.components.RandomForestRegressor.describe

RandomForestRegressor.**describe**(*print_name=False*, *return_dict=False*)

Describe a component and its parameters

Parameters

- **print_name** (*bool, optional*) – whether to print name of component
- **return_dict** (*bool, optional*) – whether to return description as dictionary in the format {"name": name, "parameters": parameters}

Returns prints and returns dictionary

Return type None or dict

evalml.pipelines.components.RandomForestRegressor.fit

RandomForestRegressor.**fit**(*X*, *y=None*)

Fits component to data

Parameters

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

Returns self

evalml.pipelines.components.RandomForestRegressor.predict

RandomForestRegressor.**predict**(*X*)

Make predictions using selected features.

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

Returns estimated labels

Return type pd.Series

evalml.pipelines.components.RandomForestRegressor.predict_proba

RandomForestRegressor.**predict_proba**(*X*)

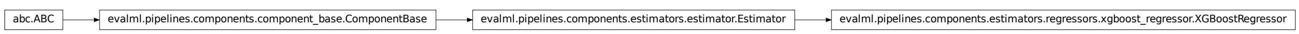
Make probability estimates for labels.

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

Returns probability estimates

Return type pd.DataFrame

evalml.pipelines.components.XGBoostRegressor



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

XGBoost Regressor

name = 'XGBoost Regressor'

model_family = 'xgboost'

supported_problem_types = [<ProblemTypes.REGRESSION: 'regression'>]

hyperparameter_ranges = {'eta': Real(low=0, high=1, prior='uniform', transform='ident

Instance attributes

SEED_MAX	
SEED_MIN	
feature_importances	Returns feature importances.

Methods:

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

evalml.pipelines.components.XGBoostRegressor.__init__

XGBoostRegressor.**__init__**(*eta=0.1*, *max_depth=3*, *min_child_weight=1*, *n_estimators=100*, *random_state=0*)

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

evalml.pipelines.components.XGBoostRegressor.describe

XGBoostRegressor.**describe**(*print_name=False*, *return_dict=False*)

Describe a component and its parameters

Parameters

- **print_name** (*bool, optional*) – whether to print name of component
- **return_dict** (*bool, optional*) – whether to return description as dictionary in the format {"name": name, "parameters": parameters}

Returns prints and returns dictionary

Return type None or dict

evalml.pipelines.components.XGBoostRegressor.fit

XGBoostRegressor.**fit**(*X*, *y=None*)

Fits component to data

Parameters

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

Returns self

evalml.pipelines.components.XGBoostRegressor.predict

XGBoostRegressor.**predict**(*X*)

Make predictions using selected features.

Parameters X (*pd.DataFrame*) – features

Returns estimated labels

Return type pd.Series

evalml.pipelines.components.XGBoostRegressor.predict_proba

XGBoostRegressor.**predict_proba**(*X*)

Make probability estimates for labels.

Parameters X (*pd.DataFrame*) – features

Returns probability estimates

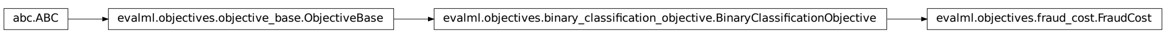
Return type pd.DataFrame

Objective Functions

Domain-Specific Objectives

FraudCost	Score the percentage of money lost of the total transaction amount process due to fraud
LeadScoring	Lead scoring

evalml.objectives.FraudCost



class evalml.objectives.**FraudCost**(*retry_percentage=0.5*, *interchange_fee=0.02*, *fraud_payout_percentage=1.0*, *amount_col='amount'*)

Score the percentage of money lost of the total transaction amount process due to fraud

Methods

__init__	Create instance of FraudCost
decision_function	Determine if a transaction is fraud given predicted probabilities, threshold, and dataframe with transaction amount
objective_function	Calculate amount lost to fraud per transaction given predictions, true values, and dataframe with transaction amount
optimize_threshold	Learn a binary classification threshold which optimizes the current objective.
score	Returns a numerical score indicating performance based on the differences between the predicted and actual values.

evalml.objectives.FraudCost.__init__

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

Create instance of FraudCost

Parameters

- **retry_percentage** (*float*) – What percentage of customers that will retry a transaction if it is declined. Between 0 and 1. Defaults to .5
- **interchange_fee** (*float*) – How much of each successful transaction you can collect. Between 0 and 1. Defaults to .02
- **fraud_payout_percentage** (*float*) – Percentage of fraud you will not be able to collect. Between 0 and 1. Defaults to 1.0
- **amount_col** (*str*) – Name of column in data that contains the amount. Defaults to "amount"

evalml.objectives.FraudCost.decision_function

FraudCost.**decision_function**(*ypred_proba*, *threshold=0.0*, *X=None*)

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

Parameters

- **ypred_proba** (*pd.Series*) – Predicted probablities
- **X** (*pd.DataFrame*) – Dataframe containing transaction amount
- **threshold** (*float*) – Dollar threshold to determine if transaction is fraud

Returns Series of predicted fraud labels using X and threshold

Return type pd.Series

evalml.objectives.FraudCost.objective_function

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

Calculate amount lost to fraud per transaction given predictions, true values, and dataframe with transaction amount

Parameters

- **y_predicted** (*pd.Series*) – predicted fraud labels
- **y_true** (*pd.Series*) – true fraud labels
- **X** (*pd.DataFrame*) – dataframe with transaction amounts

Returns amount lost to fraud per transaction

Return type float

evalml.objectives.FraudCost.optimize_threshold

FraudCost.**optimize_threshold**(*ypred_proba*, *y_true*, *X=None*)

Learn a binary classification threshold which optimizes the current objective.

Parameters

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

Returns Optimal threshold for this objective

evalml.objectives.FraudCost.score

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

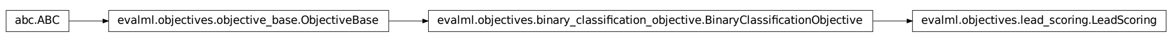
Returns a numerical score indicating performance based on the differences between the predicted and actual values.

Parameters

- **y_predicted** (*pd.Series*) – predicted values of length [n_samples]
- **y_true** (*pd.Series*) – actual class labels of length [n_samples]
- **X** (*pd.DataFrame or np.array*) – extra data of shape [n_samples, n_features] necessary to calculate score

Returns score

evalml.objectives.LeadScoring



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

Lead scoring

Methods

`__init__`	Create instance.
`decision_function`	Apply a learned threshold to predicted probabilities to get predicted classes.
`objective_function`	Calculate the profit per lead.
`optimize_threshold`	Learn a binary classification threshold which optimizes the current objective.
`score`	Returns a numerical score indicating performance based on the differences between the predicted and actual values.

evalml.objectives.LeadScoring.__init__

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

Create instance.

Parameters

- **true_positives** (*int*) – reward for a true positive
- **false_positives** (*int*) – cost for a false positive. Should be negative.

evalml.objectives.LeadScoring.decision_function

LeadScoring.**decision_function**(*ypred_proba*, *threshold=0.5*, *X=None*)

Apply a learned threshold to predicted probabilities to get predicted classes.

Parameters

- **ypred_proba** (*list*) – The classifier's predicted probabilities
- **threshold** (*float, optional*) – Threshold used to make a prediction. Defaults to 0.5.

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

Returns predictions

evalml.objectives.LeadScoring.objective_function

LeadScoring.**objective_function**(*y_true*, *y_predicted*, *X=None*)

Calculate the profit per lead.

Parameters

- **y_predicted** (*pd.Series*) – predicted labels
- **y_true** (*pd.Series*) – true labels
- **X** (*pd.DataFrame*) – None, not used.

Returns profit per lead

Return type float

evalml.objectives.LeadScoring.optimize_threshold

LeadScoring.**optimize_threshold**(*ypred_proba*, *y_true*, *X=None*)

Learn a binary classification threshold which optimizes the current objective.

Parameters

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

Returns Optimal threshold for this objective

evalml.objectives.LeadScoring.score

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

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

Parameters

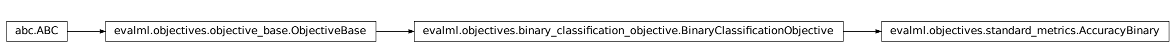
- **y_predicted** (*pd.Series*) – predicted values of length [n_samples]
- **y_true** (*pd.Series*) – actual class labels of length [n_samples]
- **X** (*pd.DataFrame or np.array*) – extra data of shape [n_samples, n_features] necessary to calculate score

Returns score

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
RecallWeighted	Recall score for multiclass classification using weighted averaging

evalml.objectives.AccuracyBinary



class evalml.objectives.**AccuracyBinary**

Accuracy score for binary classification

Methods

decision_function	Apply a learned threshold to predicted probabilities to get predicted classes.
objective_function	Computes the relative value of the provided predictions compared to the actual labels, according a specified metric
optimize_threshold	Learn a binary classification threshold which optimizes the current objective.
score	Returns a numerical score indicating performance based on the differences between the predicted and actual values.

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*)

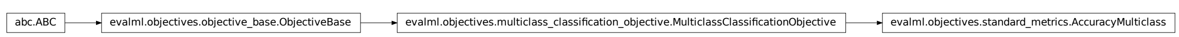
Returns a numerical score indicating performance based on the differences between the predicted and actual values.

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.AccuracyMulticlass



class evalml.objectives.**AccuracyMulticlass**

Accuracy score for multiclass classification

Methods

objective_function	Computes the relative value of the provided predictions compared to the actual labels, according a specified metric
score	Returns a numerical score indicating performance based on the differences between the predicted and actual values.

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

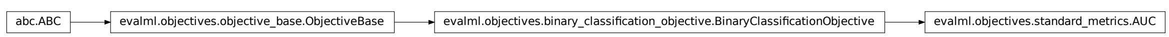
AccuracyMulticlass.**score**(*y_true*, *y_predicted*, *X=None*)
Returns a numerical score indicating performance based on the differences between the predicted and actual values.

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.AUC



class evalml.objectives.**AUC**
AUC score for binary classification

Methods

decision_function	Apply a learned threshold to predicted probabilities to get predicted classes.
objective_function	Computes the relative value of the provided predictions compared to the actual labels, according a specified metric
optimize_threshold	Learn a binary classification threshold which optimizes the current objective.
score	Returns a numerical score indicating performance based on the differences between the predicted and actual values.

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*)

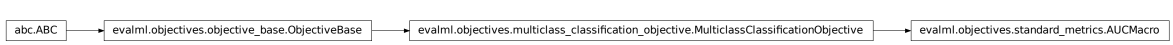
Returns a numerical score indicating performance based on the differences between the predicted and actual values.

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.AUCMacro



class evalml.objectives.**AUCMacro**

AUC score for multiclass classification using macro averaging

Methods

objective_function	Computes the relative value of the provided predictions compared to the actual labels, according a specified metric
score	Returns a numerical score indicating performance based on the differences between the predicted and actual values.

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*)

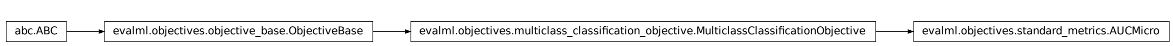
Returns a numerical score indicating performance based on the differences between the predicted and actual values.

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.AUCMicro



class evalml.objectives.**AUCMicro**

AUC score for multiclass classification using micro averaging

Methods

objective_function	Computes the relative value of the provided predictions compared to the actual labels, according a specified metric
score	Returns a numerical score indicating performance based on the differences between the predicted and actual values.

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*)

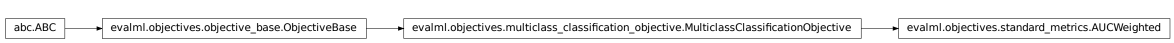
Returns a numerical score indicating performance based on the differences between the predicted and actual values.

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.AUCWeighted



class evalml.objectives.**AUCWeighted**

AUC Score for multiclass classification using weighted averaging

Methods

objective_function	Computes the relative value of the provided predictions compared to the actual labels, according a specified metric
score	Returns a numerical score indicating performance based on the differences between the predicted and actual values.

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*)

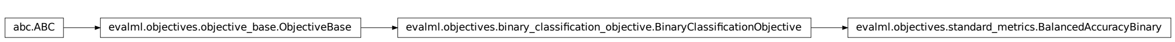
Returns a numerical score indicating performance based on the differences between the predicted and actual values.

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.BalancedAccuracyBinary



class `evalml.objectives.`**`BalancedAccuracyBinary`**

Balanced accuracy score for binary classification

Methods

decision_function	Apply a learned threshold to predicted probabilities to get predicted classes.

Continued on next page

Table 84 – continued from previous page

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.

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*)

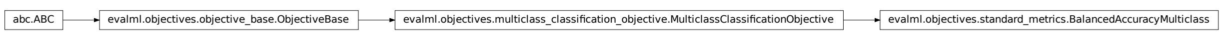
Returns a numerical score indicating performance based on the differences between the predicted and actual values.

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.BalancedAccuracyMulticlass



class evalml.objectives.**BalancedAccuracyMulticlass**

Balanced accuracy score for multiclass classification

Methods

objective_function	Computes the relative value of the provided predictions compared to the actual labels, according a specified metric
score	Returns a numerical score indicating performance based on the differences between the predicted and actual values.

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

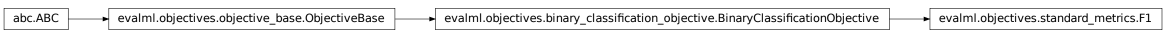
actual values.

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.F1



class evalml.objectives.**F1**
F1 score for binary classification

Methods

decision_function	Apply a learned threshold to predicted probabilities to get predicted classes.
objective_function	Computes the relative value of the provided predictions compared to the actual labels, according a specified metric
optimize_threshold	Learn a binary classification threshold which optimizes the current objective.
score	Returns a numerical score indicating performance based on the differences between the predicted and actual values.

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*)

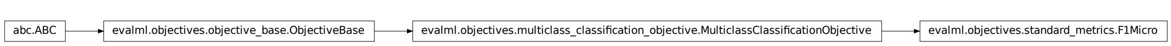
Returns a numerical score indicating performance based on the differences between the predicted and actual values.

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.F1Micro



class evalml.objectives.**F1Micro**

F1 score for multiclass classification using micro averaging

Methods

objective_function	Computes the relative value of the provided predictions compared to the actual labels, according a specified metric
score	Returns a numerical score indicating performance based on the differences between the predicted and actual values.

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*)

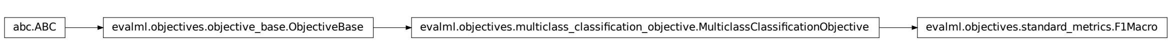
Returns a numerical score indicating performance based on the differences between the predicted and actual values.

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.F1Macro



class evalml.objectives.**F1Macro**

F1 score for multiclass classification using macro averaging

Methods

objective_function	Computes the relative value of the provided predictions compared to the actual labels, according a specified metric
score	Returns a numerical score indicating performance based on the differences between the predicted and actual values.

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*)

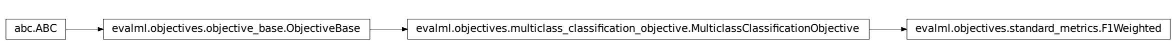
Returns a numerical score indicating performance based on the differences between the predicted and actual values.

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.F1Weighted



class evalml.objectives.**F1Weighted**

F1 score for multiclass classification using weighted averaging

Methods

objective_function	Computes the relative value of the provided predictions compared to the actual labels, according a specified metric
score	Returns a numerical score indicating performance based on the differences between the predicted and actual values.

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*)

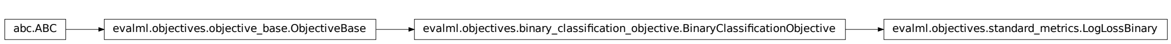
Returns a numerical score indicating performance based on the differences between the predicted and actual values.

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.LogLossBinary



class evalml.objectives.**LogLossBinary**

Log Loss for binary classification

Methods

decision_function	Apply a learned threshold to predicted probabilities to get predicted classes.

Continued on next page

Table 90 – continued from previous page

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.

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*)

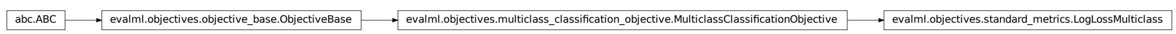
Returns a numerical score indicating performance based on the differences between the predicted and actual values.

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.LogLossMulticlass



class evalml.objectives.**LogLossMulticlass**

Log Loss for multiclass classification

Methods

objective_function	Computes the relative value of the provided predictions compared to the actual labels, according a specified metric
score	Returns a numerical score indicating performance based on the differences between the predicted and actual values.

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

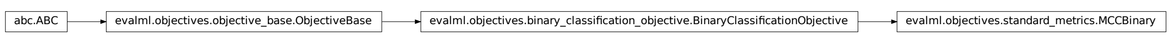
actual values.

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.MCCBinary



class evalml.objectives.**MCCBinary**

Matthews correlation coefficient for binary classification

Methods

decision_function	Apply a learned threshold to predicted probabilities to get predicted classes.
objective_function	Computes the relative value of the provided predictions compared to the actual labels, according a specified metric
optimize_threshold	Learn a binary classification threshold which optimizes the current objective.
score	Returns a numerical score indicating performance based on the differences between the predicted and actual values.

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*)

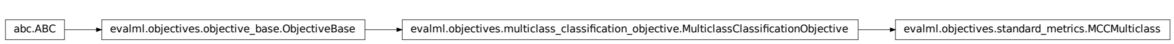
Returns a numerical score indicating performance based on the differences between the predicted and actual values.

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.MCCMulticlass



class evalml.objectives.**MCCMulticlass**

Matthews correlation coefficient for multiclass classification

Methods

objective_function	Computes the relative value of the provided predictions compared to the actual labels, according a specified metric
score	Returns a numerical score indicating performance based on the differences between the predicted and actual values.

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*)

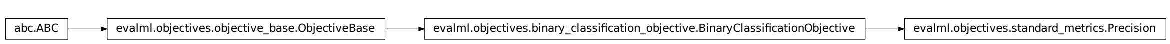
Returns a numerical score indicating performance based on the differences between the predicted and actual values.

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.Precision



class evalml.objectives.**Precision**

Precision score for binary classification

Methods

decision_function	Apply a learned threshold to predicted probabilities to get predicted classes.
objective_function	Computes the relative value of the provided predictions compared to the actual labels, according a specified metric
optimize_threshold	Learn a binary classification threshold which optimizes the current objective.
score	Returns a numerical score indicating performance based on the differences between the predicted and actual values.

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*)

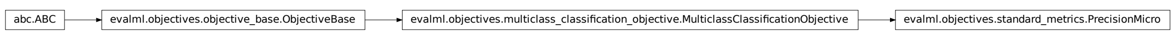
Returns a numerical score indicating performance based on the differences between the predicted and actual values.

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.PrecisionMicro



class evalml.objectives.**PrecisionMicro**

Precision score for multiclass classification using micro averaging

Methods

objective_function	Computes the relative value of the provided predictions compared to the actual labels, according a specified metric
score	Returns a numerical score indicating performance based on the differences between the predicted and actual values.

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

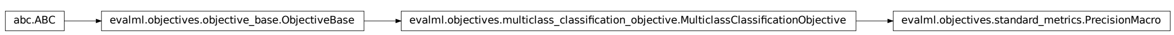
actual values.

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.PrecisionMacro



class evalml.objectives.**PrecisionMacro**

Precision score for multiclass classification using macro averaging

Methods

objective_function	Computes the relative value of the provided predictions compared to the actual labels, according a specified metric
score	Returns a numerical score indicating performance based on the differences between the predicted and actual values.

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.

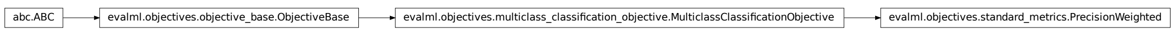
Parameters

- **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.PrecisionWeighted



class evalml.objectives.**PrecisionWeighted**

Precision score for multiclass classification using weighted averaging

Methods

objective_function	Computes the relative value of the provided predictions compared to the actual labels, according a specified metric
score	Returns a numerical score indicating performance based on the differences between the predicted and actual values.

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*)

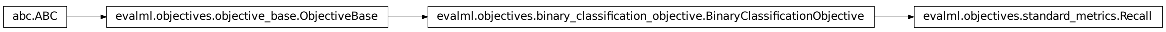
Returns a numerical score indicating performance based on the differences between the predicted and actual values.

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.Recall



class evalml.objectives.**Recall**
Recall score for binary classification

Methods

decision_function	Apply a learned threshold to predicted probabilities to get predicted classes.
objective_function	Computes the relative value of the provided predictions compared to the actual labels, according a specified metric
optimize_threshold	Learn a binary classification threshold which optimizes the current objective.
score	Returns a numerical score indicating performance based on the differences between the predicted and actual values.

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*)

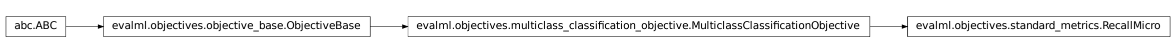
Returns a numerical score indicating performance based on the differences between the predicted and actual values.

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.RecallMicro



class `evalml.objectives.`**`RecallMicro`**

Recall score for multiclass classification using micro averaging

Methods

`objective_function`	Computes the relative value of the provided predictions compared to the actual labels, according a specified metric
`score`	Returns a numerical score indicating performance based on the differences between the predicted and actual values.

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*)

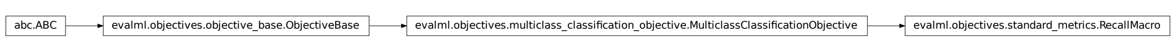
Returns a numerical score indicating performance based on the differences between the predicted and actual values.

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.RecallMacro



class `evalml.objectives.`**`RecallMacro`**

Recall score for multiclass classification using macro averaging

Methods

objective_function	Computes the relative value of the provided predictions compared to the actual labels, according a specified metric
score	Returns a numerical score indicating performance based on the differences between the predicted and actual values.

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*)

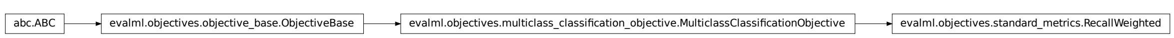
Returns a numerical score indicating performance based on the differences between the predicted and actual values.

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.RecallWeighted



class evalml.objectives.**RecallWeighted**

Recall score for multiclass classification using weighted averaging

Methods

objective_function	Computes the relative value of the provided predictions compared to the actual labels, according a specified metric
score	Returns a numerical score indicating performance based on the differences between the predicted and actual values.

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

Regression Objectives

R2	Coefficient of determination for regression
MAE	Mean absolute error for regression
MSE	Mean squared error for regression
MedianAE	Median absolute error for regression
MaxError	Maximum residual error for regression
ExpVariance	Explained variance score for regression

evalml.objectives.R2

class `evalml.objectives.R2`

Coefficient of determination for regression

Methods

objective_function	Computes the relative value of the provided predictions compared to the actual labels, according a specified metric
score	Returns a numerical score indicating performance based on the differences between the predicted and actual values.

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.MAE

class evalml.objectives.**MAE**

Mean absolute error for regression

Methods

objective_function	Computes the relative value of the provided predictions compared to the actual labels, according a specified metric
score	Returns a numerical score indicating performance based on the differences between the predicted and actual values.

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.MSE

class evalml.objectives.**MSE**

Mean squared error for regression

Methods

objective_function	Computes the relative value of the provided predictions compared to the actual labels, according a specified metric
score	Returns a numerical score indicating performance based on the differences between the predicted and actual values.

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.MedianAE

class evalml.objectives.**MedianAE**

Median absolute error for regression

Methods

objective_function	Computes the relative value of the provided predictions compared to the actual labels, according a specified metric
score	Returns a numerical score indicating performance based on the differences between the predicted and actual values.

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.MaxError

class evalml.objectives.**MaxError**

Maximum residual error for regression

Methods

objective_function	Computes the relative value of the provided predictions compared to the actual labels, according a specified metric
score	Returns a numerical score indicating performance based on the differences between the predicted and actual values.

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*)

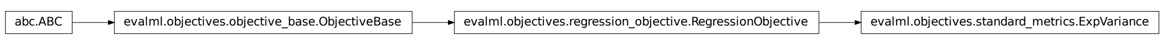
Returns a numerical score indicating performance based on the differences between the predicted and actual values.

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.ExpVariance



class evalml.objectives.**ExpVariance**

Explained variance score for regression

Methods

objective_function	Computes the relative value of the provided predictions compared to the actual labels, according a specified metric
score	Returns a numerical score indicating performance based on the differences between the predicted and actual values.

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

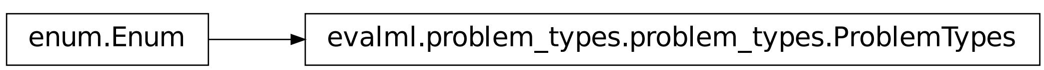
- **y_predicted** (*pd.Series*) – predicted values of length [n_samples]
- **y_true** (*pd.Series*) – actual class labels of length [n_samples]
- **X** (*pd.DataFrame or np.array*) – extra data of shape [n_samples, n_features] necessary to calculate score

Returns score

Problem Types

ProblemTypes	Enum for type of machine learning problem: BINARY, MULTICLASS, or REGRESSION

evalml.problem_types.ProblemTypes



class `evalml.problem_types.ProblemTypes`

Enum for type of machine learning problem: BINARY, MULTICLASS, or REGRESSION

handle_problem_types	Handles problem_type by either returning the ProblemTypes or converting from a str

evalml.problem_types.handle_problem_types

`evalml.problem_types.handle_problem_types(problem_type)`

Handles problem_type by either returning the ProblemTypes or converting from a str

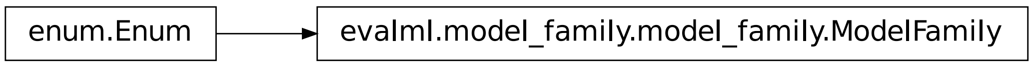
Parameters **problem_types** (*str or ProblemTypes*) – problem type that needs to be handled

Returns ProblemTypes

Model Family

ModelFamily	Enum for family of machine learning models.

evalml.model_family.ModelFamily



class evalml.model_family.**ModelFamily**
Enum for family of machine learning models.

Tuners

Tuner	Defines API for Tuners
SKOptTuner	Bayesian Optimizer
GridSearchTuner	Grid Search Optimizer
RandomSearchTuner	Random Search Optimizer

evalml.tuners.Tuner

class evalml.tuners.**Tuner**(*space*, *random_state=0*)
Defines API for Tuners

Tuners implement different strategies for sampling from a search space. They're used in EvalML to search the space of pipeline hyperparameters.

Methods

`__init__`	Init Tuner
`add`	Register a set of hyperparameters with the score obtained from training a pipeline with those hyperparameters.
`is_search_space_exhausted`	Optional.
`propose`	Returns a set of hyperparameters to train a pipeline with, based off the search space dimensions and prior samples

evalml.tuners.Tuner.__init__

Tuner.**__init__**(*space*, *random_state=0*)

Init Tuner

Parameters

- **space** (*dict*) – search space for hyperparameters
- **random_state** (*int, np.random.RandomState*) – The random state

Returns self

Return type *Tuner*

evalml.tuners.Tuner.add

Tuner.**add**(*parameters*, *score*)

Register a set of hyperparameters with the score obtained from training a pipeline with those hyperparameters.

Parameters

- **parameters** (*dict*) – hyperparameters
- **score** (*float*) – associated score

Returns None

evalml.tuners.Tuner.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

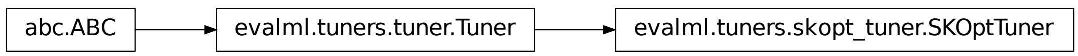
Tuner.**propose**()

Returns a set of hyperparameters to train a pipeline with, based off the search space dimensions and prior samples

Returns proposed hyperparameters

Return type dict

evalml.tuners.SKOptTuner



class `evalml.tuners.`**`SKOptTuner`**(*space*, *random_state=0*)

Bayesian Optimizer

Methods

`__init__`	Init SkOptTuner
`add`	Add score to sample
`is_search_space_exhausted`	Optional.
`propose`	Returns hyperparameters based off search space and samples

evalml.tuners.SKOptTuner.__init__

`SKOptTuner.`**`__init__`**(*space*, *random_state=0*)

Init SkOptTuner

Parameters

- **space** (*dict*) – search space for hyperparameters
- **random_state** (*int, np.random.RandomState*) – The random state

Returns self

Return type SKoptTuner

evalml.tuners.SKOptTuner.add

`SKOptTuner.`**`add`**(*parameters*, *score*)

Add score to sample

Parameters

- **parameters** (*dict*) – hyperparameters
- **score** (*float*) – associated score

Returns None

evalml.tuners.SKOptTuner.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

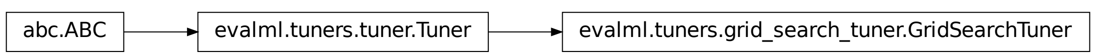
SKOptTuner.**propose**()

Returns hyperparameters based off search space and samples

Returns proposed hyperparameters

Return type dict

evalml.tuners.GridSearchTuner



class evalml.tuners.**GridSearchTuner**(*space*, *n_points=10*, *random_state=0*)

Grid Search Optimizer

Example

```
>>> tuner = GridSearchTuner([(1,10), ['A', 'B']], n_points=5)
>>> print(tuner.propose())
(1.0, 'A')
>>> print(tuner.propose())
(1.0, 'B')
>>> print(tuner.propose())
(3.25, '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.

Continued on next page

Table 115 – continued from previous page

is_search_space_exhausted	Checks if it is possible to generate a set of valid parameters.
propose	Returns hyperparameters from _grid_points iterations

evalml.tuners.GridSearchTuner.__init__

GridSearchTuner.**__init__**(*space*, *n_points=10*, *random_state=0*)

Generate all of the possible points to search for in the grid

Parameters

- **space** – A list of all dimensions available to tune
- **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**(*parameters*, *score*)

Not applicable to grid search tuner as generated parameters are not dependent on scores of previous parameters.

Parameters

- **parameters** – Hyperparameters used
- **score** – Associated score

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 hyperparameters from _grid_points iterations

If all possible combinations of parameters have been scored, then `NoParamsException` is raised.

Returns proposed hyperparameters

Return type dict

evalml.tuners.RandomSearchTuner

class evalml.tuners.**RandomSearchTuner**(*space*, *random_state=0*, *with_replacement=False*, *replacement_max_attempts=10*)

Random Search Optimizer

Example

```
>>> tuner = RandomSearchTuner([(1,10), ['A', 'B']], random_state=0)
>>> print(tuner.propose())
(6, 'B')
>>> print(tuner.propose())
(4, 'B')
>>> print(tuner.propose())
(5, 'A')
```

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__**(*space*, *random_state=0*, *with_replacement=False*, *replacement_max_attempts=10*)

Sets up check for duplication if needed.

Parameters

- **space** – A list of all dimensions available to tune
- **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**(*parameters*, *score*)
Not applicable to random search tuner as generated parameters are not dependent on scores of previous parameters.

Parameters

- **parameters** – Hyperparameters used
- **score** – Associated score

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 A list of unique parameters

Guardrails

detect_highly_null	Checks if there are any highly-null columns in a dataframe.
detect_label_leakage	Check if any of the features are highly correlated with the target.
detect_outliers	Checks if there are any outliers in a dataframe by using first Isolation Forest to obtain the anomaly score of each index and then using IQR to determine score anomalies.
detect_id_columns	Check if any of the features are ID columns.

evalml.guardrails.detect_highly_null

evalml.guardrails.**detect_highly_null**(*X*, *percent_threshold=0.95*)
Checks if there are any highly-null columns in a dataframe.

Parameters

- **X** (*pd.DataFrame*) – features
- **percent_threshold** (*float*) – Require that percentage of null values to be considered

"highly-null", defaults to .95

Returns A dictionary of features with column name or index and their percentage of null values

Example

```
>>> df = pd.DataFrame({
...     'lots_of_null': [None, None, None, None, 5],
...     'no_null': [1, 2, 3, 4, 5]
... })
>>> detect_highly_null(df, percent_threshold=0.8)
{'lots_of_null': 0.8}
```

evalml.guardrails.detect_label_leakage

`evalml.guardrails.`**`detect_label_leakage`**(*X*, *y*, *threshold=0.95*)

Check if any of the features are highly correlated with the target.

Currently only supports binary and numeric targets and features

Parameters

- **X** (*pd.DataFrame*) – The input features to check
- **y** (*pd.Series*) – the labels
- **threshold** (*float*) – the correlation threshold to be considered leakage. Defaults to .95

Returns leakage, dictionary of features with leakage and corresponding threshold

Example

```
>>> X = pd.DataFrame({
...     'leak': [10, 42, 31, 51, 61],
...     'x': [42, 54, 12, 64, 12],
...     'y': [12, 5, 13, 74, 24],
... })
>>> y = pd.Series([10, 42, 31, 51, 40])
>>> detect_label_leakage(X, y, threshold=0.8)
{'leak': 0.8827072320669518}
```

evalml.guardrails.detect_outliers

`evalml.guardrails.`**`detect_outliers`**(*X*, *random_state=0*)

Checks if there are any outliers in a dataframe by using first Isolation Forest to obtain the anomaly score of each index and then using IQR to determine score anomalies. Indices with score anomalies are considered outliers.

Parameters **X** (*pd.DataFrame*) – features

Returns A set of indices that may have outlier data.

Example

```
>>> df = pd.DataFrame({
...     'x': [1, 2, 3, 40, 5],
...     'y': [6, 7, 8, 990, 10],
...     'z': [-1, -2, -3, -1201, -4]
... })
>>> detect_outliers(df)
[3]
```

evalml.guardrails.detect_id_columns

evalml.guardrails.**detect_id_columns**(*X*, *threshold=1.0*)

Check if any of the features are ID columns. Currently performs these simple checks:

- column name is "id"
- column name ends in "_id"
- column contains all unique values (and is not float / boolean)

Parameters

- **X** (*pd.DataFrame*) – The input features to check
- **threshold** (*float*) – the probability threshold to be considered an ID column. Defaults to 1.0

Returns A dictionary of features with column name or index and their probability of being ID columns

Example

```
>>> df = pd.DataFrame({
...     'df_id': [0, 1, 2, 3, 4],
...     'x': [10, 42, 31, 51, 61],
...     'y': [42, 54, 12, 64, 12]
... })
>>> detect_id_columns(df)
{'df_id': 1.0}
```

Utils

import_or_raise	Attempts to import the requested library by name.
convert_to_seconds	
get_random_state	Generates a numpy.random.RandomState instance using seed.
get_random_seed	Given a numpy.random.RandomState object, generate an int representing a seed value for another random number generator.

evalml.utils.import_or_raise

`evalml.utils.import_or_raise(library, error_msg=None)`

Attempts to import the requested library by name. If the import fails, raises an ImportError.

Parameters

- **library** (*str*) – the name of the library
- **error_msg** (*str*) – error message to return if the import fails

evalml.utils.convert_to_seconds

`evalml.utils.convert_to_seconds(input_str)`

evalml.utils.get_random_state

`evalml.utils.get_random_state(seed)`

Generates a numpy.random.RandomState instance using seed.

Parameters **seed** (*None, int, np.random.RandomState object*) – seed to use to generate numpy.random.RandomState. Must be between SEED_BOUNDS.min_bound and SEED_BOUNDS.max_bound, inclusive. Otherwise, an exception will be thrown.

evalml.utils.get_random_seed

`evalml.utils.get_random_seed(random_state, min_bound=0, max_bound=2147483647)`

Given a numpy.random.RandomState object, generate an int representing a seed value for another random number generator. Or, if given an int, return that int.

To protect against invalid input to a particular library's random number generator, if an int value is provided, and it is outside the bounds "[min_bound, max_bound)", the value will be projected into the range between the min_bound (inclusive) and max_bound (exclusive) using modular arithmetic.

Parameters

- **random_state** (*int, numpy.random.RandomState*) – random state
- **min_bound** (*None, int*) – if not default of None, will be min bound when generating seed (inclusive). Must be less than max_bound.
- **max_bound** (*None, int*) – if not default of None, will be max bound when generating seed (exclusive). Must be greater than min_bound.

Returns seed for random number generator

Return type int

1.6.16 FAQ

What is the difference between EvalML and other AutoML libraries?

EvalML optimizes machine learning pipelines on *custom practical objectives* instead of vague machine learning loss functions so that it will find the best pipelines for your specific needs. Furthermore, EvalML *pipelines* are able to take in all kinds of data (missing values, categorical, etc.) as long as the data are in a single table. EvalML also allows you to build your own pipelines with existing or custom components so you can have more control over the AutoML process. Moreover, EvalML also provides you with support in the form of *guardrails* to ensure that you are aware of potential issues your data may cause with machine learning algorithms".

How does EvalML handle missing values?

EvalML contains imputation components in its pipelines so that missing values are taken care of. EvalML optimizes over different types of imputation to search for the best possible pipeline. You can find more information about components *here* and in the API reference *here*.

How does EvalML handle categorical encoding?

EvalML provides a *one-hot-encoding component* in its pipelines for categorical variables. EvalML plans to support other encoders in the future.

How does EvalML handle feature selection?

EvalML currently utilizes scikit-learn's SelectFromModel with a Random Forest classifier/regressor to handle feature selection. EvalML plans on supporting more feature selectors in the future. You can find more information in the API reference *here*.

How are feature importances calculated?

Feature importance depends on the estimator used. Variable coefficients are used for regression-based estimators (Logistic Regression and Linear Regression) and Gini importance is used for tree-based estimators (Random Forest and XGBoost).

How does hyperparameter tuning work?

EvalML tunes hyperparameters for its pipelines through Bayesian optimization. In the future we plan to support more optimization techniques such as random search.

Can I create my own objective metric?

Yes you can! You can *create your own custom objective* so that EvalML optimizes the best model for your needs.

How does EvalML avoid overfitting?

EvalML provides *guardrails* to combat overfitting. Such guardrails include detecting label leakage, unstable pipelines, hold-out datasets and cross validation. EvalML defaults to using Stratified K-Fold cross-validation for classification problems and K-Fold cross-validation for regression problems but allows you to utilize your own cross-validation methods as well.

Can I create my own pipeline for EvalML?

Yes! EvalML allows you to create *custom pipelines* using modular components. This allows you to customize EvalML pipelines for your own needs or for AutoML.

Does EvalML work with X algorithm?

EvalML is constantly improving and adding new components and will allow your own algorithms to be used as components in our pipelines.

INDEX

Symbols

A

B

C

D

E

F

G

M

N

P

R

S

T

X