EvalML Documentation

Release 0.28.0

Alteryx, Inc.

Jul 02, 2021

CONTENTS

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 supervised machine learning solutions.

CHAPTER

ONE

INSTALL

EvalML is available for Python 3.7 and 3.8 with experimental support 3.9. It can be installed with pip or conda.

1.1 Pip with all dependencies

To install evalml with pip, run the following command:

```
pip install evalml
```

1.2 Pip with core dependencies

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

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

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

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

1.2.1 Add-ons

Update checker Receive automatic notifications of new EvalML releases

```
pip install evalml[update_checker]
```

1.3 Conda with all dependencies

To install evalml with conda run the following command:

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

1.4 Conda with core dependencies

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

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

1.5 Windows

Additionally, if you are using `pip` to install EvalML, it is recommended you first install the following packages using conda: * `numba` (needed for `shap` and prediction explanations) * `graphviz` if you're using EvalML's plotting utilities

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

1.6 Mac

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

```
brew install libomp
```

Additionally, `graphviz` can be installed by running

```
brew install graphviz
```

1.7 Python 3.9 support

Evalml can still be installed with pip in python 3.9 but note that `sktime`, one of our dependencies, will not be installed because that library does not yet support python 3.9. This means the `PolynomialDetrending` component will not be usable in python 3.9. You can try to install `sktime` from source in python 3.9 to use the `PolynomialDetrending` component but be warned that we only test it in python 3.7 and 3.8.

CHAPTER

TWO

START

In this guide, we'll show how you can use EvalML to automatically find the best pipeline for predicting whether or not a credit card transaction is fradulent. Along the way, we'll highlight EvalML's built-in tools and features for understanding and interacting with the search process.

[1]:
```
import evalml
from evalml import AutoMLSearch
from evalml.utils import infer_feature_types
```

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

[2]:
```
X, y = evalml.demos.load_fraud(n_rows=250)
```

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

Number of training examples: 250
Targets
False    88.40%
True     11.60%
Name: fraud, dtype: object
```

First, we will clean the data. Since EvalML accepts a pandas input, it can run type inference on this data directly. Since we'd like to change the types inferred by EvalML, we can use the `infer_feature_types` utility method. Here's what we're going to do with the following dataset:

- Reformat the `expiration_date` column so it reflects a more familiar date format.
- Cast the `lat` and `lng` columns from float to str.
- Use `infer_feature_types` to specify what types certain columns should be. For example, to avoid having the `provider` column be inferred as natural language text, we have specified it as a categorical column instead.

The `infer_feature_types` utility method takes a pandas or numpy input and converts it to a pandas dataframe with a Woodwork accessor, providing us with flexibility to cast the data as necessary.

[3]:
```
X.ww['expiration_date'] = X['expiration_date'].apply(lambda x: '20{}-01-{}'.format(x.
↪split("/")[1], x.split("/")[0]))
X = infer_feature_types(X, feature_types= {'store_id': 'categorical',
                                           'expiration_date': 'datetime',
                                           'lat': 'categorical',
                                           'lng': 'categorical',
                                           'provider': 'categorical'})
X.ww
```

[3]:
```
                  Physical Type Logical Type Semantic Tag(s)
Column
card_id                   int64      Integer     ['numeric']
store_id                  int64      Integer     ['numeric']
datetime          datetime64[ns]     Datetime              []
amount                    int64      Integer     ['numeric']
currency               category  Categorical    ['category']
customer_present           bool      Boolean              []
expiration_date        category  Categorical    ['category']
provider               category  Categorical    ['category']
lat                     float64       Double     ['numeric']
lng                     float64       Double     ['numeric']
region                 category  Categorical    ['category']
country                category  Categorical    ['category']
```

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,␣
↪problem_type='binary', test_size=.2)
```

Note: To provide data to EvalML, it is recommended that you initialize a woodwork accessor so that you control how EvalML will treat each feature, such as as a numeric feature, a categorical feature, a text feature or other type of feature. Consult the the Woodwork project for help on how to do this. Here, `split_data()` returns dataframes with woodwork accessors.

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

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

EvalML provides a number of parameters to control the search process. `max_batches` is one of the parameters which controls the stopping criterion for the AutoML search. It indicates the maximum number of rounds of AutoML to evaluate, where each round may train and score a variable number of pipelines. In this example, `max_batches` is set to 1.

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

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

[5]:
```
automl = AutoMLSearch(X_train=X_train, y_train=y_train,
                      problem_type='binary', objective='f1', max_batches=1)
```

```
Generating pipelines to search over...
8 pipelines ready for search.
```

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*.

[6]:
```
automl.search()
```

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

Optimizing for F1.
Greater score is better.

Using SequentialEngine to train and score pipelines.
Searching up to 1 batches for a total of 9 pipelines.
Allowed model families: catboost, random_forest, decision_tree, linear_model,␣
↪lightgbm, xgboost, extra_trees
```

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

```
Evaluating Baseline Pipeline: Mode Baseline Binary Classification Pipeline
Mode Baseline Binary Classification Pipeline:
        Starting cross validation
        Finished cross validation - mean F1: 0.000

*****************************
* Evaluating Batch Number 1 *
*****************************

Elastic Net Classifier w/ Imputer + DateTime Featurization Component + One Hot␣
↪Encoder + SMOTENC Oversampler + Standard Scaler:
        Starting cross validation
        Finished cross validation - mean F1: 0.185
        High coefficient of variation (cv >= 0.2) within cross validation scores.
        Elastic Net Classifier w/ Imputer + DateTime Featurization Component + One␣
↪Hot Encoder + SMOTENC Oversampler + Standard Scaler may not perform as estimated on␣
↪unseen data.
Decision Tree Classifier w/ Imputer + DateTime Featurization Component + One Hot␣
↪Encoder + SMOTENC Oversampler:
        Starting cross validation
        Finished cross validation - mean F1: 0.410
        High coefficient of variation (cv >= 0.2) within cross validation scores.
        Decision Tree Classifier w/ Imputer + DateTime Featurization Component + One␣
↪Hot Encoder + SMOTENC Oversampler may not perform as estimated on unseen data.
Random Forest Classifier w/ Imputer + DateTime Featurization Component + One Hot␣
↪Encoder + SMOTENC Oversampler:
        Starting cross validation
        Finished cross validation - mean F1: 0.706
        High coefficient of variation (cv >= 0.2) within cross validation scores.
        Random Forest Classifier w/ Imputer + DateTime Featurization Component + One␣
↪Hot Encoder + SMOTENC Oversampler may not perform as estimated on unseen data.
LightGBM Classifier w/ Imputer + DateTime Featurization Component + One Hot Encoder +␣
↪SMOTENC Oversampler:
        Starting cross validation
        Finished cross validation - mean F1: 0.589
Logistic Regression Classifier w/ Imputer + DateTime Featurization Component + One␣
↪Hot Encoder + SMOTENC Oversampler + Standard Scaler:
        Starting cross validation
        Finished cross validation - mean F1: 0.191
```

(continues on next page)

(continued from previous page)

```
        High coefficient of variation (cv >= 0.2) within cross validation scores.
        Logistic Regression Classifier w/ Imputer + DateTime Featurization Component␣
↪+ One Hot Encoder + SMOTENC Oversampler + Standard Scaler may not perform as␣
↪estimated on unseen data.
XGBoost Classifier w/ Imputer + DateTime Featurization Component + One Hot Encoder +␣
↪SMOTENC Oversampler:
        Starting cross validation
        Finished cross validation - mean F1: 0.599
        High coefficient of variation (cv >= 0.2) within cross validation scores.
        XGBoost Classifier w/ Imputer + DateTime Featurization Component + One Hot␣
↪Encoder + SMOTENC Oversampler may not perform as estimated on unseen data.
Extra Trees Classifier w/ Imputer + DateTime Featurization Component + One Hot␣
↪Encoder + SMOTENC Oversampler:
        Starting cross validation
        Finished cross validation - mean F1: 0.000
CatBoost Classifier w/ Imputer + DateTime Featurization Component + SMOTENC␣
↪Oversampler:
        Starting cross validation
        Finished cross validation - mean F1: 0.206

Search finished after 00:18
Best pipeline: Random Forest Classifier w/ Imputer + DateTime Featurization Component␣
↪+ One Hot Encoder + SMOTENC Oversampler
Best pipeline F1: 0.706349
```

We also provide *a standalone ``search`* method <../generated/evalml.automl.search.html>`__ which does all of the above in a single line, and returns the `AutoMLSearch` instance and data check results. If there were data check errors, AutoML will not be run and no `AutoMLSearch` instance will be returned.

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

[7]:
```
automl.rankings
```

[7]:

	id	pipeline_name	search_order	mean_cv_score	standard_deviation_cv_score	validation_score
0	3	Random Forest Classifier w/ Imputer + DateTime...	3	0.706349	0.240857	0.857143
1	6	XGBoost Classifier w/ Imputer + DateTime Featu...	6	0.599267	0.170331	0.769231
2	4	LightGBM Classifier w/ Imputer + DateTime Feat...	4	0.588889	0.083887	0.666667
3	2	Decision Tree Classifier w/ Imputer + DateTime...	2	0.410256	0.387171	0.769231
4	8	CatBoost Classifier w/ Imputer + DateTime Feat...	8	0.206149	0.012443	0.213333
5	5	Logistic Regression Classifier w/ Imputer + Da...	5	0.191453	0.167067	0.266667
6	1	Elastic Net Classifier w/ Imputer + DateTime F...	1	0.185185	0.169725	0.222222
7	0	Mode Baseline Binary Classification Pipeline	0	0.000000	0.000000	0.000000
8	7	Extra Trees Classifier w/ Imputer + DateTime F...	7	0.000000	0.000000	0.000000

(continues on next page)

(continued from previous page)

```
   percent_better_than_baseline  high_variance_cv  \
0                     70.634921              True
1                     59.926740              True
2                     58.888889             False
3                     41.025641              True
4                     20.614916             False
5                     19.145299              True
6                     18.518519              True
7                      0.000000             False
8                      0.000000             False

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

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

[8]:
```
automl.describe_pipeline(3)
```

```
*********************************************************************************************
* Random Forest Classifier w/ Imputer + DateTime Featurization Component + One Hot␣
↪Encoder + SMOTENC Oversampler *
*********************************************************************************************

Problem Type: binary
Model Family: Random Forest

Pipeline Steps
==============
1. Imputer
         * categorical_impute_strategy : most_frequent
         * numeric_impute_strategy : mean
         * categorical_fill_value : None
         * numeric_fill_value : None
2. DateTime Featurization Component
         * features_to_extract : ['year', 'month', 'day_of_week', 'hour']
         * encode_as_categories : False
         * date_index : None
3. One Hot Encoder
         * top_n : 10
         * features_to_encode : None
         * categories : None
         * drop : if_binary
         * handle_unknown : ignore
         * handle_missing : error
4. SMOTENC Oversampler
         * sampling_ratio : 0.25
```

(continues on next page)

(continued from previous page)

```
         * k_neighbors_default : 5
         * n_jobs : -1
         * sampling_ratio_dict : None
         * categorical_features : [3, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21,␣
↪22, 23, 24, 25, 26, 27, 28, 29, 30, 31, 32, 33, 34, 35, 36, 37, 38, 39, 40, 41, 42,␣
↪43, 44, 45, 46, 47, 48, 49, 50, 51, 52, 53, 54, 55, 56, 57, 58, 59]
         * k_neighbors : 5
5. Random Forest Classifier
         * n_estimators : 100
         * max_depth : 6
         * n_jobs : -1

Training
========
Training for binary problems.
Objective to optimize binary classification pipeline thresholds for: <evalml.
↪objectives.standard_metrics.F1 object at 0x7fc5e5b63520>
Total training time (including CV): 2.4 seconds

Cross Validation
----------------
               F1  MCC Binary  Log Loss Binary   AUC  Precision  Balanced Accuracy␣
↪Binary  Accuracy Binary # Training # Validation
0           0.857       0.852            0.264 0.824      1.000                    0.
↪875            0.970        133           67
1           0.429       0.368            0.333 0.712      0.500                    0.
↪662            0.881        133           67
2           0.833       0.831            0.260 0.879      1.000                    0.
↪857            0.970        134           66
mean        0.706       0.684                - -                                   0.
↪798            0.940          -           -
std         0.241       0.273            0.041 0.085      0.289                    0.
↪118            0.052          -           -
coef of var 0.341       0.400            0.144 0.106      0.346                    0.
↪148            0.055          -           -
```

We can also view the pipeline parameters directly:

[9]:
```
pipeline = automl.get_pipeline(3)
print(pipeline.parameters)
```

```
{'Imputer': {'categorical_impute_strategy': 'most_frequent', 'numeric_impute_strategy
↪': 'mean', 'categorical_fill_value': None, 'numeric_fill_value': None}, 'DateTime␣
↪Featurization Component': {'features_to_extract': ['year', 'month', 'day_of_week',
↪'hour'], 'encode_as_categories': False, 'date_index': None}, 'One Hot Encoder': {
↪'top_n': 10, 'features_to_encode': None, 'categories': None, 'drop': 'if_binary',
↪'handle_unknown': 'ignore', 'handle_missing': 'error'}, 'SMOTENC Oversampler': {
↪'sampling_ratio': 0.25, 'k_neighbors_default': 5, 'n_jobs': -1, 'sampling_ratio_dict
↪': None, 'categorical_features': [3, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21,
↪ 22, 23, 24, 25, 26, 27, 28, 29, 30, 31, 32, 33, 34, 35, 36, 37, 38, 39, 40, 41, 42,
↪ 43, 44, 45, 46, 47, 48, 49, 50, 51, 52, 53, 54, 55, 56, 57, 58, 59], 'k_neighbors':
↪ 5}, 'Random Forest Classifier': {'n_estimators': 100, 'max_depth': 6, 'n_jobs': -1}
↪}
```

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

[10]:
```
pipeline = automl.best_pipeline
```

(continues on next page)

(continued from previous page)

```
pipeline.score(X_holdout, y_holdout, ["f1"])
```

[10]:
```
OrderedDict([('F1', 0.8)])
```

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

[11]:
```
pipeline.graph()
```

[11]:

CHAPTER

THREE

TUTORIALS

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

3.1 Building a Fraud Prediction Model with EvalML

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

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

3.1.1 Configure "Cost of Fraud"

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

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

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

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

3.1.2 Search for best pipeline

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

[3]:
```
X, y = evalml.demos.load_fraud(n_rows=5000)
```

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

Number of training examples: 5000
Targets
False    86.20%
True     13.80%
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]:
```
cols_to_drop = ['datetime', 'expiration_date', 'country', 'region', 'provider']
for col in cols_to_drop:
    X.ww.pop(col)

X_train, X_holdout, y_train, y_holdout = evalml.preprocessing.split_data(X, y,␣
↪problem_type='binary', test_size=0.2, random_seed=0)

X.ww
```

[4]:

Column	Physical Type	Logical Type	Semantic Tag(s)
card_id	int64	Integer	['numeric']
store_id	int64	Integer	['numeric']
amount	int64	Integer	['numeric']
currency	category	Categorical	['category']
customer_present	bool	Boolean	[]
lat	float64	Double	['numeric']
lng	float64	Double	['numeric']

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

[5]:
```
automl = AutoMLSearch(X_train=X_train, y_train=y_train,
                      problem_type='binary',
                      objective=fraud_objective,
                      additional_objectives=['auc', 'f1', 'precision'],
                      allowed_model_families=["random_forest", "linear_model"],
                      max_batches=1,
                      optimize_thresholds=True)

automl.search()
```

```
Generating pipelines to search over...
3 pipelines ready for search.

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

(continues on next page)

(continued from previous page)

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

Using SequentialEngine to train and score pipelines.
Searching up to 1 batches for a total of 4 pipelines.
Allowed model families: linear_model, random_forest
```

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

```
Evaluating Baseline Pipeline: Mode Baseline Binary Classification Pipeline
Mode Baseline Binary Classification Pipeline:
        Starting cross validation
        Finished cross validation - mean Fraud Cost: 0.165

*****************************
* Evaluating Batch Number 1 *
*****************************

Elastic Net Classifier w/ Imputer + One Hot Encoder + SMOTENC Oversampler + Standard␣
↪Scaler:
        Starting cross validation
        Finished cross validation - mean Fraud Cost: 0.008
Random Forest Classifier w/ Imputer + One Hot Encoder + SMOTENC Oversampler:
        Starting cross validation
        Finished cross validation - mean Fraud Cost: 0.008
Logistic Regression Classifier w/ Imputer + One Hot Encoder + SMOTENC Oversampler +␣
↪Standard Scaler:
        Starting cross validation
        Finished cross validation - mean Fraud Cost: 0.008

Search finished after 00:07
Best pipeline: Elastic Net Classifier w/ Imputer + One Hot Encoder + SMOTENC␣
↪Oversampler + Standard Scaler
Best pipeline Fraud Cost: 0.007802
```

View rankings and select pipelines

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	search_order	\
0	1	Elastic Net Classifier w/ Imputer + One Hot En...	1	
1	3	Logistic Regression Classifier w/ Imputer + On...	3	
2	2	Random Forest Classifier w/ Imputer + One Hot ...	2	
3	0	Mode Baseline Binary Classification Pipeline	0	

	mean_cv_score	standard_deviation_cv_score	validation_score	\
0	0.007802	0.000039	0.007758	
1	0.007802	0.000039	0.007758	

(continues on next page)

(continued from previous page)

```
2       0.007803                    0.000040          0.007758
3       0.164789                    0.002964          0.168169

   percent_better_than_baseline  high_variance_cv  \
0                     15.698661             False
1                     15.698661             False
2                     15.698619             False
3                      0.000000             False

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

To select the best pipeline we can call `automl.best_pipeline`.

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

Describe pipelines

We can get more details about any pipeline created during the search process, including how it performed on other objective functions, by calling the `describe_pipeline` method and passing the `id` of the pipeline of interest.

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

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

Problem Type: binary
Model Family: Linear

Pipeline Steps
==============
1. Imputer
         * categorical_impute_strategy : most_frequent
         * numeric_impute_strategy : mean
         * categorical_fill_value : None
         * numeric_fill_value : None
2. One Hot Encoder
         * top_n : 10
         * features_to_encode : None
         * categories : None
         * drop : if_binary
         * handle_unknown : ignore
         * handle_missing : error
3. SMOTENC Oversampler
         * sampling_ratio : 0.25
         * k_neighbors_default : 5
         * n_jobs : -1
         * sampling_ratio_dict : None
         * categorical_features : [3, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15]
```

(continues on next page)

(continued from previous page)

```
         * k_neighbors : 5
4. Standard Scaler
5. Logistic Regression Classifier
         * penalty : l2
         * C : 1.0
         * n_jobs : -1
         * multi_class : auto
         * solver : lbfgs

Training
========
Training for binary problems.
Objective to optimize binary classification pipeline thresholds for: <evalml.
↪objectives.fraud_cost.FraudCost object at 0x7fe61be26f70>
Total training time (including CV): 2.9 seconds

Cross Validation
----------------
             Fraud Cost    AUC     F1  Precision # Training # Validation
0                 0.008  0.854  0.000      0.000      2,666        1,334
1                 0.008  0.799  0.000      0.000      2,667        1,333
2                 0.008  0.828  0.421      1.000      2,667        1,333
mean              0.008  0.827  0.140      0.333          -            -
std               0.000  0.027  0.243      0.577          -            -
coef of var       0.005  0.033  1.732      1.732          -            -
```

3.1.3 Evaluate on holdout data

Finally, since the best pipeline is already trained, we evaluate it on the holdout data.

Now, we can score the pipeline on the holdout data using both our fraud cost objective and the AUC (Area under the ROC Curve) objective.

[9]:
```
best_pipeline.score(X_holdout, y_holdout, objectives=["auc", fraud_objective])
```

[9]:
```
OrderedDict([('AUC', 0.8097111537038906),
             ('Fraud Cost', 0.007834043025742435)])
```

3.1.4 Why optimize for a problem-specific objective?

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

[10]:
```
automl_auc = AutoMLSearch(X_train=X_train, y_train=y_train,
                          problem_type='binary',
                          objective='auc',
                          additional_objectives=['f1', 'precision'],
                          max_batches=1,
                          allowed_model_families=["random_forest", "linear_model"],
                          optimize_thresholds=True)

automl_auc.search()
```

```
Generating pipelines to search over...
3 pipelines ready for search.

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

Optimizing for AUC.
Greater score is better.

Using SequentialEngine to train and score pipelines.
Searching up to 1 batches for a total of 4 pipelines.
Allowed model families: linear_model, random_forest
```

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

```
Evaluating Baseline Pipeline: Mode Baseline Binary Classification Pipeline
Mode Baseline Binary Classification Pipeline:
        Starting cross validation
        Finished cross validation - mean AUC: 0.500

*****************************
* Evaluating Batch Number 1 *
*****************************

Elastic Net Classifier w/ Imputer + One Hot Encoder + SMOTENC Oversampler + Standard␣
↪Scaler:
        Starting cross validation
        Finished cross validation - mean AUC: 0.827
Random Forest Classifier w/ Imputer + One Hot Encoder + SMOTENC Oversampler:
        Starting cross validation
        Finished cross validation - mean AUC: 0.853
Logistic Regression Classifier w/ Imputer + One Hot Encoder + SMOTENC Oversampler +␣
↪Standard Scaler:
        Starting cross validation
        Finished cross validation - mean AUC: 0.827

Search finished after 00:05
Best pipeline: Random Forest Classifier w/ Imputer + One Hot Encoder + SMOTENC␣
↪Oversampler
Best pipeline AUC: 0.853164
```

Like before, we can look at the rankings of all of the pipelines searched and pick the best pipeline.

[11]:
```
automl_auc.rankings
```

[11]:
```
   id                                   pipeline_name  search_order  \
0   2  Random Forest Classifier w/ Imputer + One Hot ...             2
1   1  Elastic Net Classifier w/ Imputer + One Hot En...             1
2   3  Logistic Regression Classifier w/ Imputer + On...             3
3   0       Mode Baseline Binary Classification Pipeline             0

   mean_cv_score  standard_deviation_cv_score  validation_score  \
0       0.853164                     0.010011          0.863894
1       0.827102                     0.027002          0.853667
```

(continues on next page)

(continued from previous page)

```
2       0.826675                    0.027444          0.853544
3       0.500000                    0.000000          0.500000

   percent_better_than_baseline  high_variance_cv  \
0                     35.316397             False
1                     32.710221             False
2                     32.667497             False
3                      0.000000             False

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

[12]:
```
best_pipeline_auc = automl_auc.best_pipeline
```

[13]:
```
# get the fraud score on holdout data
best_pipeline_auc.score(X_holdout, y_holdout,  objectives=["auc", fraud_objective])
```

[13]:
```
OrderedDict([('AUC', 0.8644456773933219), ('Fraud Cost', 0.00783500768341736)])
```

[14]:
```
# fraud score on fraud optimized again
best_pipeline.score(X_holdout, y_holdout, objectives=["auc", fraud_objective])
```

[14]:
```
OrderedDict([('AUC', 0.8097111537038906),
             ('Fraud Cost', 0.007834043025742435)])
```

When we optimize for AUC, we can see that the AUC score from this pipeline performs better compared to the AUC score from the pipeline optimized for fraud cost; however, the losses due to fraud are a much larger percentage of the total transaction amount when optimized for AUC and much smaller when optimized for fraud cost. As a result, we lose a noticable percentage of the total transaction amount by not optimizing for fraud cost specifically.

Optimizing for AUC does not take into account the user-specified `retry_percentage`, `interchange_fee`, `fraud_payout_percentage` values, which could explain the decrease in fraud performance. Thus, the best pipelines may produce the highest AUC but may not actually reduce the amount loss due to your specific type fraud.

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

3.2 Building a Lead Scoring Model with EvalML

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

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

3.2.1 Configure LeadScoring

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

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

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

[2]:
```
lead_scoring_objective = LeadScoring(
    true_positives=100,
    false_positives=-5
)
```

3.2.2 Dataset

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

[3]:
```
from urllib.request import urlopen
import pandas as pd
import woodwork as ww
customers_data = urlopen('https://featurelabs-static.s3.amazonaws.com/lead_scoring_ml_
↪apps/customers.csv')
interactions_data = urlopen('https://featurelabs-static.s3.amazonaws.com/lead_scoring_
↪ml_apps/interactions.csv')
leads_data = urlopen('https://featurelabs-static.s3.amazonaws.com/lead_scoring_ml_
↪apps/previous_leads.csv')
customers = pd.read_csv(customers_data)
interactions = pd.read_csv(interactions_data)
leads = pd.read_csv(leads_data)

X = customers.merge(interactions, on='customer_id').merge(leads, on='customer_id')
y = X['label']
X = X.drop(['customer_id', 'date_registered', 'birthday','phone', 'email',
            'owner', 'company', 'id', 'time_x',
            'session', 'referrer', 'time_y', 'label', 'country'], axis=1)
display(X.head())
```

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

We will convert our data into Woodwork data structures. Doing so enables us to have more control over the types passed to and inferred by AutoML.

[4]:
```
X.ww.init(semantic_tags={'job': 'category'}, logical_types={'job': 'Categorical'})
y = ww.init_series(y)
X.ww
```

[4]:
```
       Physical Type Logical Type Semantic Tag(s)
Column
job         category  Categorical    ['category']
```

(continues on next page)

(continued from previous page)

```
state       category  Categorical    ['category']
zip          float64       Double     ['numeric']
action      category  Categorical    ['category']
amount       float64       Double     ['numeric']
```

3.2.3 Search for the 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.

[5]:
```
X_train, X_holdout, y_train, y_holdout = evalml.preprocessing.split_data(X, y,␣
↪problem_type='binary', test_size=0.2, random_seed=0)

X.ww
```

[5]:
```
       Physical Type Logical Type Semantic Tag(s)
Column
job         category  Categorical    ['category']
state       category  Categorical    ['category']
zip          float64       Double     ['numeric']
action      category  Categorical    ['category']
amount       float64       Double     ['numeric']
```

Because the lead scoring labels are binary, we will use set the problem type to "binary". When we call `.search()`, the search for the best pipeline will begin.

[6]:
```
automl = AutoMLSearch(X_train=X_train, y_train=y_train,
                      problem_type='binary',
                      objective=lead_scoring_objective,
                      additional_objectives=['auc'],
                      allowed_model_families=["catboost", "random_forest", "linear_
↪model"],
                      max_batches=1)

automl.search()
```

```
Generating pipelines to search over...
4 pipelines ready for search.

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

Optimizing for Lead Scoring.
Greater score is better.

Using SequentialEngine to train and score pipelines.
Searching up to 1 batches for a total of 5 pipelines.
Allowed model families: random_forest, linear_model, catboost
```

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

(continues on next page)

(continued from previous page)

```
             'name': 'Best Score',
             'type'...
```

```
Evaluating Baseline Pipeline: Mode Baseline Binary Classification Pipeline
Mode Baseline Binary Classification Pipeline:
        Starting cross validation
        Finished cross validation - mean Lead Scoring: 0.000

*****************************
* Evaluating Batch Number 1 *
*****************************

Elastic Net Classifier w/ Imputer + One Hot Encoder + SMOTENC Oversampler + Standard␣
↪Scaler:
        Starting cross validation
        Finished cross validation - mean Lead Scoring: 0.018
        High coefficient of variation (cv >= 0.2) within cross validation scores.
        Elastic Net Classifier w/ Imputer + One Hot Encoder + SMOTENC Oversampler +␣
↪Standard Scaler may not perform as estimated on unseen data.
Random Forest Classifier w/ Imputer + One Hot Encoder + SMOTENC Oversampler:
        Starting cross validation
        Finished cross validation - mean Lead Scoring: 0.019
        High coefficient of variation (cv >= 0.2) within cross validation scores.
        Random Forest Classifier w/ Imputer + One Hot Encoder + SMOTENC Oversampler␣
↪may not perform as estimated on unseen data.
Logistic Regression Classifier w/ Imputer + One Hot Encoder + SMOTENC Oversampler +␣
↪Standard Scaler:
        Starting cross validation
        Finished cross validation - mean Lead Scoring: 0.018
        High coefficient of variation (cv >= 0.2) within cross validation scores.
        Logistic Regression Classifier w/ Imputer + One Hot Encoder + SMOTENC␣
↪Oversampler + Standard Scaler may not perform as estimated on unseen data.
CatBoost Classifier w/ Imputer + SMOTENC Oversampler:
        Starting cross validation
        Finished cross validation - mean Lead Scoring: 0.373
        High coefficient of variation (cv >= 0.2) within cross validation scores.
        CatBoost Classifier w/ Imputer + SMOTENC Oversampler may not perform as␣
↪estimated on unseen data.

Search finished after 00:11
Best pipeline: CatBoost Classifier w/ Imputer + SMOTENC Oversampler
Best pipeline Lead Scoring: 0.373253
```

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.

[7]:
```
automl.rankings
```

[7]:
```
   id                                     pipeline_name  search_order  \
0   4  CatBoost Classifier w/ Imputer + SMOTENC Overs...             4
1   2  Random Forest Classifier w/ Imputer + One Hot ...             2
2   1  Elastic Net Classifier w/ Imputer + One Hot En...             1
3   3  Logistic Regression Classifier w/ Imputer + On...             3
4   0       Mode Baseline Binary Classification Pipeline             0
```

(continues on next page)

(continued from previous page)

```
   mean_cv_score  standard_deviation_cv_score  validation_score  \
0       0.373253                     0.329982          0.000000
1       0.019368                     0.036376         -0.003226
2       0.018293                     0.037411         -0.006452
3       0.018293                     0.037411         -0.006452
4       0.000000                     0.000000          0.000000

   percent_better_than_baseline  high_variance_cv  \
0                           inf              True
1                           inf              True
2                           inf              True
3                           inf              True
4                           0.0             False

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

To select the best pipeline we can call `automl.best_pipeline`.

[8]:
```
best_pipeline = automl.best_pipeline
```

Describe pipeline

You can get more details about any pipeline, including how it performed on other objective functions by calling `.describe_pipeline()` and specifying the `id` of the pipeline.

[9]:
```
automl.describe_pipeline(automl.rankings.iloc[0]["id"])
```

```
*********************************************************
* CatBoost Classifier w/ Imputer + SMOTENC Oversampler *
*********************************************************

Problem Type: binary
Model Family: CatBoost

Pipeline Steps
==============
1. Imputer
         * categorical_impute_strategy : most_frequent
         * numeric_impute_strategy : mean
         * categorical_fill_value : None
         * numeric_fill_value : None
2. SMOTENC Oversampler
         * sampling_ratio : 0.25
         * k_neighbors_default : 5
         * n_jobs : -1
         * sampling_ratio_dict : None
         * categorical_features : [0, 1, 3]
         * k_neighbors : 5
```

(continues on next page)

(continued from previous page)

```
3. CatBoost Classifier
         * n_estimators : 10
         * eta : 0.03
         * max_depth : 6
         * bootstrap_type : None
         * silent : True
         * allow_writing_files : False
         * n_jobs : -1

Training
========
Training for binary problems.
Objective to optimize binary classification pipeline thresholds for: <evalml.
↪objectives.lead_scoring.LeadScoring object at 0x7f694efc2040>
Total training time (including CV): 1.0 seconds

Cross Validation
----------------
             Lead Scoring   AUC # Training # Validation
0                   0.000 0.869      3,099        1,550
1                   0.494 0.887      3,099        1,550
2                   0.626 0.889      3,100        1,549
mean                0.373 0.882          -            -
std                 0.330 0.011          -            -
coef of var         0.884 0.012          -            -
```

3.2.4 Evaluate on hold out

Finally, since the best pipeline was trained on all of the training data, we evaluate it on the holdout dataset.

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

[10]:
```
OrderedDict([('AUC', 0.8585599879117558),
             ('Lead Scoring', 0.8383490971625107)])
```

3.2.5 Why optimize for a problem-specific objective?

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

[11]:
```
automl_auc = evalml.AutoMLSearch(X_train=X_train, y_train=y_train,
                                 problem_type='binary',
                                 objective='auc',
                                 additional_objectives=[lead_scoring_objective],
                                 allowed_model_families=["catboost", "random_forest",
↪"linear_model"],
                                 max_batches=1)

automl_auc.search()
```

```
Generating pipelines to search over...
4 pipelines ready for search.

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

Optimizing for AUC.
Greater score is better.

Using SequentialEngine to train and score pipelines.
Searching up to 1 batches for a total of 5 pipelines.
Allowed model families: random_forest, linear_model, catboost
```

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

```
Evaluating Baseline Pipeline: Mode Baseline Binary Classification Pipeline
Mode Baseline Binary Classification Pipeline:
        Starting cross validation
        Finished cross validation - mean AUC: 0.500

*****************************
* Evaluating Batch Number 1 *
*****************************

Elastic Net Classifier w/ Imputer + One Hot Encoder + SMOTENC Oversampler + Standard␣
↪Scaler:
        Starting cross validation
        Finished cross validation - mean AUC: 0.653
Random Forest Classifier w/ Imputer + One Hot Encoder + SMOTENC Oversampler:
        Starting cross validation
        Finished cross validation - mean AUC: 0.652
Logistic Regression Classifier w/ Imputer + One Hot Encoder + SMOTENC Oversampler +␣
↪Standard Scaler:
        Starting cross validation
        Finished cross validation - mean AUC: 0.653
CatBoost Classifier w/ Imputer + SMOTENC Oversampler:
        Starting cross validation
        Finished cross validation - mean AUC: 0.882

Search finished after 00:09
Best pipeline: CatBoost Classifier w/ Imputer + SMOTENC Oversampler
Best pipeline AUC: 0.881709
```

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

[12]:

	id	pipeline_name	search_order	\
0	4	CatBoost Classifier w/ Imputer + SMOTENC Overs...	4	
1	3	Logistic Regression Classifier w/ Imputer + On...	3	
2	1	Elastic Net Classifier w/ Imputer + One Hot En...	1	
3	2	Random Forest Classifier w/ Imputer + One Hot ...	2	
4	0	Mode Baseline Binary Classification Pipeline	0	

	mean_cv_score	standard_deviation_cv_score	validation_score	\

(continues on next page)

(continued from previous page)

```
0       0.881709                      0.010861          0.869201
1       0.653038                      0.035883          0.691832
2       0.652855                      0.035875          0.691582
3       0.651591                      0.059622          0.717764
4       0.500000                      0.000000          0.500000

   percent_better_than_baseline  high_variance_cv  \
0                     38.170947             False
1                     15.303770             False
2                     15.285487             False
3                     15.159098             False
4                      0.000000             False

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

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

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

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

[14]:
```
OrderedDict([('AUC', 0.8585599879117558),
             ('Lead Scoring', 0.07738607050730868)])
```

[15]:
```
assert best_pipeline_score['Lead Scoring'] > best_pipeline_auc_score['Lead Scoring']
assert best_pipeline_auc_score['Lead Scoring'] >= 0
```

When we optimize for AUC, we can see that the AUC score from this pipeline is similar to the AUC score from the pipeline optimized for lead scoring. However, the revenue per lead is much smaller per lead when optimized for AUC and was much larger when optimized for lead scoring. As a result, we would have a huge gain on the amount of revenue if we optimized for lead scoring.

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

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

3.3 Using the Cost-Benefit Matrix Objective

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

3.3.1 Confusion Matrix

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

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

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

3.3.2 Cost-Benefit Matrix

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

3.3.3 Customer Churn Example

Data

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

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

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

[1]:
```
from evalml.demos.churn import load_churn
from evalml.preprocessing import split_data

X, y = load_churn()
X.ww.set_types({'PaymentMethod':'Categorical', 'Contract': 'Categorical'}) # Update␣
↪data types Woodwork did not correctly infer
X_train, X_holdout, y_train, y_holdout = split_data(X, y, problem_type='binary', test_
↪size=0.3, random_seed=0)
```

```
             Number of Features
Categorical                  16
Numeric                       3

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

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

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

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

AutoML Search with Log Loss

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

[3]:
```
from evalml import AutoMLSearch
automl = AutoMLSearch(X_train=X_train, y_train=y_train, problem_type='binary',␣
↪objective='log loss binary',
                      max_iterations=5)
automl.search()

ll_pipeline = automl.best_pipeline
ll_pipeline.score(X_holdout, y_holdout, ['log loss binary'])
```

```
Generating pipelines to search over...
8 pipelines ready for search.

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

Optimizing for Log Loss Binary.
Lower score is better.

Using SequentialEngine to train and score pipelines.
Searching up to 5 pipelines.
Allowed model families: decision_tree, xgboost, random_forest, linear_model, extra_
↪trees, lightgbm, catboost
```

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

```
Evaluating Baseline Pipeline: Mode Baseline Binary Classification Pipeline
Mode Baseline Binary Classification Pipeline:
        Starting cross validation
        Finished cross validation - mean Log Loss Binary: 9.164

*****************************
* Evaluating Batch Number 1 *
*****************************

Elastic Net Classifier w/ Imputer + One Hot Encoder + Standard Scaler:
        Starting cross validation
        Finished cross validation - mean Log Loss Binary: 0.424
Decision Tree Classifier w/ Imputer + One Hot Encoder:
        Starting cross validation
        Finished cross validation - mean Log Loss Binary: 0.628
Random Forest Classifier w/ Imputer + One Hot Encoder:
        Starting cross validation
        Finished cross validation - mean Log Loss Binary: 0.426
LightGBM Classifier w/ Imputer + One Hot Encoder:
        Starting cross validation
        Finished cross validation - mean Log Loss Binary: 0.471

Search finished after 00:10
Best pipeline: Elastic Net Classifier w/ Imputer + One Hot Encoder + Standard Scaler
Best pipeline Log Loss Binary: 0.424184
```

[3]:
```
OrderedDict([('Log Loss Binary', 0.4193730950794927)])
```

When we train our pipelines using log loss as our primary objective, we try to find pipelines that minimize log loss. However, our ultimate goal in training models is to find a model that gives us the most profit, so let's score our pipeline on the cost benefit matrix (using the costs outlined above) to determine the profit we would earn from the predictions made by this model:

[4]:
```
ll_pipeline_score = ll_pipeline.score(X_holdout, y_holdout, [cost_benefit_matrix])
print (ll_pipeline_score)
```

```
OrderedDict([('Cost Benefit Matrix', 53.715097018457165)])
```

[5]:
```
# Calculate total profit across all customers using pipeline optimized for Log Loss
total_profit_ll = ll_pipeline_score['Cost Benefit Matrix'] * len(X)
print (total_profit_ll)
```

```
378315.4283009938
```

AutoML Search with Cost-Benefit Matrix

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

[6]:
```
automl = AutoMLSearch(X_train=X_train, y_train=y_train, problem_type='binary',␣
↪objective=cost_benefit_matrix,
                      max_iterations=5)
automl.search()

cbm_pipeline = automl.best_pipeline
```

```
Generating pipelines to search over...
8 pipelines ready for search.

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

Optimizing for Cost Benefit Matrix.
Greater score is better.

Using SequentialEngine to train and score pipelines.
Searching up to 5 pipelines.
Allowed model families: decision_tree, xgboost, random_forest, linear_model, extra_
↪trees, lightgbm, catboost
```

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

```
Evaluating Baseline Pipeline: Mode Baseline Binary Classification Pipeline
Mode Baseline Binary Classification Pipeline:
        Starting cross validation
        Finished cross validation - mean Cost Benefit Matrix: -53.063

*****************************
* Evaluating Batch Number 1 *
*****************************

Elastic Net Classifier w/ Imputer + One Hot Encoder + Standard Scaler:
        Starting cross validation
        Finished cross validation - mean Cost Benefit Matrix: 59.574
Decision Tree Classifier w/ Imputer + One Hot Encoder:
        Starting cross validation
        Finished cross validation - mean Cost Benefit Matrix: 51.888
Random Forest Classifier w/ Imputer + One Hot Encoder:
        Starting cross validation
        Finished cross validation - mean Cost Benefit Matrix: 59.534
LightGBM Classifier w/ Imputer + One Hot Encoder:
        Starting cross validation
        Finished cross validation - mean Cost Benefit Matrix: 52.983

Search finished after 00:11
Best pipeline: Elastic Net Classifier w/ Imputer + One Hot Encoder + Standard Scaler
Best pipeline Cost Benefit Matrix: 59.574461
```

Now, if we calculate the cost-benefit matrix score on our best pipeline, we see that with this pipeline optimized for our cost-benefit matrix objective, we are able to generate more profit per customer. Across our 7043 customers, we

generate much more profit using this best pipeline! Custom objectives like `CostBenefitMatrix` are just one example of how using EvalML can help find pipelines that can perform better on real-world problems, rather than on arbitrary standard statistical metrics.

[7]:
```
cbm_pipeline_score = cbm_pipeline.score(X_holdout, y_holdout, [cost_benefit_matrix])
print (cbm_pipeline_score)
```

```
OrderedDict([('Cost Benefit Matrix', 61.76053005205868)])
```

[8]:
```
# Calculate total profit across all customers using pipeline optimized for␣
↪CostBenefitMatrix
total_profit_cbm = cbm_pipeline_score['Cost Benefit Matrix'] * len(X)
print (total_profit_cbm)
```

```
434979.4131566493
```

[9]:
```
# Calculate difference in profit made using both pipelines
profit_diff = total_profit_cbm - total_profit_ll
print (profit_diff)
```

```
56663.984855655464
```

Finally, we can graph the confusion matrices for both pipelines to better understand why the pipeline trained using the cost-benefit matrix is able to correctly classify more samples than the pipeline trained with log loss: we were able to correctly predict more cases where the customer would have churned (true positive), allowing us to intervene and prevent those customers from leaving.

[10]:
```
from evalml.model_understanding.graphs import graph_confusion_matrix

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

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

[11]:
```
# pipeline trained with cost-benefit matrix
y_pred = cbm_pipeline.predict(X_holdout)
graph_confusion_matrix(y_holdout, y_pred)
```

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

3.4 Using Text Data with EvalML

In this demo, we will show you how to use EvalML to build models which use text data.

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

3.4.1 Dataset

We will be utilizing a dataset of SMS text messages, some of which are categorized as spam, and others which are not ("ham"). This dataset is originally from Kaggle, but modified to produce a slightly more even distribution of spam to ham.

[2]:
```
from urllib.request import urlopen
import pandas as pd

input_data = urlopen('https://featurelabs-static.s3.amazonaws.com/spam_text_messages_
↪modified.csv')
data = pd.read_csv(input_data)[:750]

X = data.drop(['Category'], axis=1)
y = data['Category']

display(X.head())
```

```
                                             Message
0  Free entry in 2 a wkly comp to win FA Cup fina...
1  FreeMsg Hey there darling it's been 3 week's n...
2  WINNER!! As a valued network customer you have...
3  Had your mobile 11 months or more? U R entitle...
4  SIX chances to win CASH! From 100 to 20,000 po...
```

The ham vs spam distribution of the data is 3:1, so any machine learning model must get above 75% accuracy in order to perform better than a trivial baseline model which simply classifies everything as ham.

[3]:
```
y.value_counts(normalize=True)
```

[3]:
```
spam    0.593333
ham     0.406667
Name: Category, dtype: float64
```

3.4.2 Search for best pipeline

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

[4]:
```
X_train, X_holdout, y_train, y_holdout = evalml.preprocessing.split_data(X, y,␣
↪problem_type='binary', test_size=0.2, random_seed=0)
```

EvalML uses Woodwork to automatically detect which columns are text columns, so you can run search normally, as you would if there was no text data. We can print out the logical type of the `Message` column and assert that it is indeed inferred as a natural language column.

[5]:
```
X_train.ww
```

[5]:
```
        Physical Type     Logical Type Semantic Tag(s)
Column
Message        string  NaturalLanguage              []
```

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

[6]:
```
automl = AutoMLSearch(X_train=X_train, y_train=y_train,
                      problem_type='binary',
                      max_batches=1,
                      optimize_thresholds=True)

automl.search()
```

```
Generating pipelines to search over...
8 pipelines ready for search.

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

Optimizing for Log Loss Binary.
Lower score is better.

Using SequentialEngine to train and score pipelines.
Searching up to 1 batches for a total of 9 pipelines.
Allowed model families: catboost, lightgbm, random_forest, xgboost, decision_tree,␣
↪extra_trees, linear_model
```

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

```
Evaluating Baseline Pipeline: Mode Baseline Binary Classification Pipeline
Mode Baseline Binary Classification Pipeline:
        Starting cross validation
        Finished cross validation - mean Log Loss Binary: 14.046

*****************************
* Evaluating Batch Number 1 *
*****************************

Elastic Net Classifier w/ Text Featurization Component + Standard Scaler:
        Starting cross validation
        Finished cross validation - mean Log Loss Binary: 0.350
        High coefficient of variation (cv >= 0.2) within cross validation scores.
        Elastic Net Classifier w/ Text Featurization Component + Standard Scaler may␣
↪not perform as estimated on unseen data.
Decision Tree Classifier w/ Text Featurization Component:
        Starting cross validation
        Finished cross validation - mean Log Loss Binary: 3.386
Random Forest Classifier w/ Text Featurization Component:
        Starting cross validation
        Finished cross validation - mean Log Loss Binary: 0.221
LightGBM Classifier w/ Text Featurization Component:
        Starting cross validation
        Finished cross validation - mean Log Loss Binary: 0.292
        High coefficient of variation (cv >= 0.2) within cross validation scores.
        LightGBM Classifier w/ Text Featurization Component may not perform as␣
↪estimated on unseen data.
Logistic Regression Classifier w/ Text Featurization Component + Standard Scaler:
        Starting cross validation
        Finished cross validation - mean Log Loss Binary: 0.350
        High coefficient of variation (cv >= 0.2) within cross validation scores.
```

(continues on next page)

(continued from previous page)

```
        Logistic Regression Classifier w/ Text Featurization Component + Standard␣
↪Scaler may not perform as estimated on unseen data.
XGBoost Classifier w/ Text Featurization Component:
        Starting cross validation
        Finished cross validation - mean Log Loss Binary: 0.266
        High coefficient of variation (cv >= 0.2) within cross validation scores.
        XGBoost Classifier w/ Text Featurization Component may not perform as␣
↪estimated on unseen data.
Extra Trees Classifier w/ Text Featurization Component:
        Starting cross validation
        Finished cross validation - mean Log Loss Binary: 0.292
CatBoost Classifier w/ Text Featurization Component:
        Starting cross validation
        Finished cross validation - mean Log Loss Binary: 0.589

Search finished after 00:23
Best pipeline: Random Forest Classifier w/ Text Featurization Component
Best pipeline Log Loss Binary: 0.221422
```

View rankings and select pipeline

Once the fitting process is done, we can see all of the pipelines that were searched.

[7]:
```
automl.rankings
```

[7]:

	id	pipeline_name	search_order	mean_cv_score	standard_deviation_cv_score	validation_score	percent_better_than_baseline	high_variance_cv
0	3	Random Forest Classifier w/ Text Featurization...	3	0.221422	0.040958	0.221587	98.423568	False
1	6	XGBoost Classifier w/ Text Featurization Compo...	6	0.266164	0.106501	0.242896	98.105025	True
2	4	LightGBM Classifier w/ Text Featurization Comp...	4	0.291768	0.114862	0.291521	97.922737	True
3	7	Extra Trees Classifier w/ Text Featurization C...	7	0.292373	0.029893	0.325764	97.918427	False
4	5	Logistic Regression Classifier w/ Text Featuri...	5	0.350340	0.074833	0.349271	97.505728	True
5	1	Elastic Net Classifier w/ Text Featurization C...	1	0.350471	0.074886	0.349437	97.504795	True
6	8	CatBoost Classifier w/ Text Featurization Comp...	8	0.588944	0.004016	0.592259	95.806967	False
7	2	Decision Tree Classifier w/ Text Featurization...	2	3.385551	0.672118	3.708759	75.896294	False
8	0	Mode Baseline Binary Classification Pipeline	0	14.045769	0.099705	13.988204		

(continues on next page)

(continued from previous page)

```
8                     0.000000             False

                                           parameters
0  {'Random Forest Classifier': {'n_estimators': ...
1  {'XGBoost Classifier': {'eta': 0.1, 'max_depth...
2  {'LightGBM Classifier': {'boosting_type': 'gbd...
3  {'Extra Trees Classifier': {'n_estimators': 10...
4  {'Logistic Regression Classifier': {'penalty':...
5  {'Elastic Net Classifier': {'penalty': 'elasti...
6  {'CatBoost Classifier': {'n_estimators': 10, '...
7  {'Decision Tree Classifier': {'criterion': 'gi...
8      {'Baseline Classifier': {'strategy': 'mode'}}
```

To select the best pipeline we can call `automl.best_pipeline`.

[8]:
```
best_pipeline = automl.best_pipeline
```

Describe pipeline

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

[9]:
```
automl.describe_pipeline(automl.rankings.iloc[0]["id"])
```

```
*************************************************************
* Random Forest Classifier w/ Text Featurization Component *
*************************************************************

Problem Type: binary
Model Family: Random Forest

Pipeline Steps
==============
1. Text Featurization Component
2. Random Forest Classifier
         * n_estimators : 100
         * max_depth : 6
         * n_jobs : -1

Training
========
Training for binary problems.
Total training time (including CV): 2.9 seconds

Cross Validation
----------------
             Log Loss Binary  MCC Binary   AUC  Precision    F1  Balanced Accuracy␣
↪Binary  Accuracy Binary # Training # Validation
0                      0.222       0.813 0.975      0.889 0.889                     0.
↪907            0.910        400          200
1                      0.180       0.875 0.985      0.937 0.925                     0.
↪936            0.940        400          200
2                      0.262       0.772 0.963      0.875 0.864                     0.
↪884            0.890        400          200
mean                   0.221       0.820 0.974      0.900 0.893                     0.
↪909            0.913          -            -
```

(continues on next page)

(continued from previous page)

```
std                     0.041       0.052 0.011       0.032 0.031                     0.
↪026            0.025           -             -
coef of var             0.185       0.063 0.012       0.036 0.034                     0.
↪028            0.028           -             -
```

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

[10]:

Notice above that there is a `Text Featurization Component` as the first step in the pipeline. AutoMLSearch uses the woodwork accessor to recognize that `'Message'` is a text column, and converts this text into numerical values that can be handled by the estimator.

3.4.3 Evaluate on holdout

Now, we can score the pipeline on the holdout data using the core objectives for binary classification problems.

[11]:
```
scores = best_pipeline.score(X_holdout, y_holdout,  objectives=evalml.objectives.get_
↪core_objectives('binary'))
print(f'Accuracy Binary: {scores["Accuracy Binary"]}')
```

```
Accuracy Binary: 0.9466666666666667
```

As you can see, this model performs relatively well on this dataset, even on unseen data.

3.4.4 Why encode text this way?

To demonstrate the importance of text-specific modeling, let's train a model with the same dataset, without letting `AutoMLSearch` detect the text column. We can change this by explicitly setting the data type of the `'Message'` column in Woodwork to `Categorical` using the utility method `infer_feature_types`.

[12]:
```
from evalml.utils import infer_feature_types
X = infer_feature_types(X, {'Message': 'Categorical'})
X_train, X_holdout, y_train, y_holdout = evalml.preprocessing.split_data(X, y,␣
↪problem_type='binary', test_size=0.2, random_seed=0)
```

[13]:
```
automl_no_text = AutoMLSearch(X_train=X_train, y_train=y_train,
                              problem_type='binary',
                              max_batches=1,
                              optimize_thresholds=True)

automl_no_text.search()
```

```
Generating pipelines to search over...
8 pipelines ready for search.

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

Optimizing for Log Loss Binary.
Lower score is better.

Using SequentialEngine to train and score pipelines.
Searching up to 1 batches for a total of 9 pipelines.
```

(continues on next page)

(continued from previous page)

```
Allowed model families: catboost, lightgbm, random_forest, xgboost, decision_tree,␣
↪extra_trees, linear_model
```

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

```
Evaluating Baseline Pipeline: Mode Baseline Binary Classification Pipeline
Mode Baseline Binary Classification Pipeline:
        Starting cross validation
        Finished cross validation - mean Log Loss Binary: 14.046

*****************************
* Evaluating Batch Number 1 *
*****************************

Elastic Net Classifier w/ Imputer + One Hot Encoder + Standard Scaler:
        Starting cross validation
        Finished cross validation - mean Log Loss Binary: 0.674
Decision Tree Classifier w/ Imputer + One Hot Encoder:
        Starting cross validation
        Finished cross validation - mean Log Loss Binary: 0.673
Random Forest Classifier w/ Imputer + One Hot Encoder:
        Starting cross validation
        Finished cross validation - mean Log Loss Binary: 0.675
LightGBM Classifier w/ Imputer + One Hot Encoder:
        Starting cross validation
        Finished cross validation - mean Log Loss Binary: 0.676
Logistic Regression Classifier w/ Imputer + One Hot Encoder + Standard Scaler:
        Starting cross validation
        Finished cross validation - mean Log Loss Binary: 0.674
XGBoost Classifier w/ Imputer + One Hot Encoder:
        Starting cross validation
        Finished cross validation - mean Log Loss Binary: 0.676
Extra Trees Classifier w/ Imputer + One Hot Encoder:
        Starting cross validation
        Finished cross validation - mean Log Loss Binary: 0.674
CatBoost Classifier w/ Imputer:
        Starting cross validation
        Finished cross validation - mean Log Loss Binary: 0.684

Search finished after 00:05
Best pipeline: Decision Tree Classifier w/ Imputer + One Hot Encoder
Best pipeline Log Loss Binary: 0.673265
```

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

[14]:
```
automl_no_text.rankings
```

[14]:

	id	pipeline_name	search_order	\
0	2	Decision Tree Classifier w/ Imputer + One Hot ...	2	
1	5	Logistic Regression Classifier w/ Imputer + On...	5	
2	1	Elastic Net Classifier w/ Imputer + One Hot En...	1	
3	7	Extra Trees Classifier w/ Imputer + One Hot En...	7	
4	3	Random Forest Classifier w/ Imputer + One Hot ...	3	
5	6	XGBoost Classifier w/ Imputer + One Hot Encoder	6	

(continues on next page)

(continued from previous page)

```
6   4   LightGBM Classifier w/ Imputer + One Hot Encoder             4
7   8                    CatBoost Classifier w/ Imputer             8
8   0       Mode Baseline Binary Classification Pipeline            0

   mean_cv_score  standard_deviation_cv_score  validation_score  \
0       0.673265                     0.001010          0.672532
1       0.673990                     0.001041          0.673079
2       0.673990                     0.001038          0.673086
3       0.674350                     0.000880          0.673595
4       0.674897                     0.000910          0.674175
5       0.675623                     0.001095          0.674990
6       0.675623                     0.001095          0.674990
7       0.684270                     0.001001          0.683586
8      14.045769                     0.099705         13.988204

   percent_better_than_baseline  high_variance_cv  \
0                     95.206635             False
1                     95.201474             False
2                     95.201472             False
3                     95.198910             False
4                     95.195014             False
5                     95.189849             False
6                     95.189849             False
7                     95.128284             False
8                      0.000000             False

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

[15]:
```
best_pipeline_no_text = automl_no_text.best_pipeline
```

Here, changing the data type of the text column removed the `Text Featurization Component` from the pipeline.

[16]:
```
best_pipeline_no_text.graph()
```

[16]:

[17]:
```
automl_no_text.describe_pipeline(automl_no_text.rankings.iloc[0]["id"])
```

```
*********************************************************
* Decision Tree Classifier w/ Imputer + One Hot Encoder *
*********************************************************

Problem Type: binary
Model Family: Decision Tree

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

(continues on next page)

(continued from previous page)

```
1. Imputer
         * categorical_impute_strategy : most_frequent
         * numeric_impute_strategy : mean
         * categorical_fill_value : None
         * numeric_fill_value : None
2. One Hot Encoder
         * top_n : 10
         * features_to_encode : None
         * categories : None
         * drop : if_binary
         * handle_unknown : ignore
         * handle_missing : error
3. Decision Tree Classifier
         * criterion : gini
         * max_features : auto
         * max_depth : 6
         * min_samples_split : 2
         * min_weight_fraction_leaf : 0.0

Training
========
Training for binary problems.
Total training time (including CV): 0.4 seconds

Cross Validation
----------------
             Log Loss Binary  MCC Binary   AUC  Precision    F1  Balanced Accuracy␣
↪Binary  Accuracy Binary # Training # Validation
0                      0.673       0.058 0.504      0.407 0.579                    0.
↪504            0.410         400           200
1                      0.673       0.058 0.504      0.407 0.579                    0.
↪504            0.410         400           200
2                      0.674       0.059 0.504      0.412 0.584                    0.
↪504            0.415         400           200
mean                   0.673       0.059 0.504      0.409 0.580                    0.
↪504            0.412           -             -
std                    0.001       0.000 0.000      0.003 0.003                    0.
↪000            0.003           -             -
coef of var            0.001       0.006 0.000      0.007 0.005                    0.
↪000            0.007           -             -
```

[18]:
```
# get standard performance metrics on holdout data
scores = best_pipeline_no_text.score(X_holdout, y_holdout, objectives=evalml.
↪objectives.get_core_objectives('binary'))
print(f'Accuracy Binary: {scores["Accuracy Binary"]}')
```

```
Accuracy Binary: 0.5933333333333334
```

Without the `Text Featurization Component`, the `'Message'` column was treated as a categorical column, and therefore the conversion of this text to numerical features happened in the `One Hot Encoder`. The best pipeline encoded the top 10 most frequent "categories" of these texts, meaning 10 text messages were one-hot encoded and all the others were dropped. Clearly, this removed almost all of the information from the dataset, as we can see the `best_pipeline_no_text` performs very similarly to randomly guessing "ham" in every case.

CHAPTER

FOUR

USER GUIDE

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

4.1 Automated Machine Learning (AutoML) Search

4.1.1 Background

Machine Learning

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

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

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

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

EvalML is currently building support for supervised time series problems: time series regression, time series binary classification, and time series multiclass classification. While we've added some features to tackle these kinds of problems, our functionality is still being actively developed so please be mindful of that before using it.

AutoML and Search

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

An effective AutoML solution offers several advantages over constructing and tuning ML models by hand. AutoML can assist with many of the difficult aspects of ML, such as avoiding overfitting and underfitting, imbalanced data, detecting data leakage and other potential issues with the problem setup, and automatically applying best-practice data cleaning, feature engineering, feature selection and various modeling techniques. AutoML can also leverage

search algorithms to optimally sweep the hyperparameter search space, resulting in model performance which would be difficult to achieve by manual training.

4.1.2 AutoML in EvalML

EvalML supports all of the above and more.

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

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

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

[1]:
```
import evalml
from evalml.utils import infer_feature_types
X, y = evalml.demos.load_fraud(n_rows=250)
```

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

Number of training examples: 250
Targets
False    88.40%
True     11.60%
Name: fraud, dtype: object
```

To provide data to EvalML, it is recommended that you initialize a Woodwork accessor on your data. This allows you to easily control how EvalML will treat each of your features before training a model.

EvalML also accepts `pandas` input, and will run type inference on top of the input `pandas` data. If you'd like to change the types inferred by EvalML, you can use the `infer_feature_types` utility method, which takes pandas or numpy input and converts it to a Woodwork data structure. The `feature_types` parameter can be used to specify what types specific columns should be.

In the example below, we reformat a couple features to make them easily consumable by the model, and then specify that the provider, which would have otherwise been inferred as a column with natural language, is a categorical column.

[2]:
```
X.ww['expiration_date'] = X['expiration_date'].apply(lambda x: '20{}-01-{}'.format(x.
↪split("/")[1], x.split("/")[0]))
X = infer_feature_types(X, feature_types= {'store_id': 'categorical',
                                           'expiration_date': 'datetime',
                                           'lat': 'categorical',
                                           'lng': 'categorical',
                                           'provider': 'categorical'})
```

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_train, X_holdout, y_train, y_holdout = evalml.preprocessing.split_data(X, y,␣
↪problem_type='binary', test_size=.2)
```

Data Checks

Before calling `AutoMLSearch.search`, we should run some sanity checks on our data to ensure that the input data being passed will not run into some common issues before running a potentially time-consuming search. EvalML has various data checks that makes this easy. Each data check will return a collection of warnings and errors if it detects potential issues with the input data. This allows users to inspect their data to avoid confusing errors that may arise during the search process. You can learn about each of the data checks available through our *data checks guide*

Here, we will run the `DefaultDataChecks` class, which contains a series of data checks that are generally useful.

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

data_checks = DefaultDataChecks("binary", "log loss binary")
data_checks.validate(X_train, y_train)
```

[4]:
```
{'warnings': [], 'errors': [], 'actions': []}
```

Since there were no warnings or errors returned, we can safely continue with the search process.

[5]:
```
automl = evalml.automl.AutoMLSearch(X_train=X_train, y_train=y_train, problem_type=
↪'binary')
automl.search()
```

```
Using default limit of max_batches=1.

Generating pipelines to search over...
8 pipelines ready for search.

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

Optimizing for Log Loss Binary.
Lower score is better.

Using SequentialEngine to train and score pipelines.
Searching up to 1 batches for a total of 9 pipelines.
Allowed model families: xgboost, linear_model, random_forest, extra_trees, decision_
↪tree, catboost, lightgbm
```

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

```
Evaluating Baseline Pipeline: Mode Baseline Binary Classification Pipeline
Mode Baseline Binary Classification Pipeline:
        Starting cross validation
        Finished cross validation - mean Log Loss Binary: 3.970

*****************************
* Evaluating Batch Number 1 *
*****************************

Elastic Net Classifier w/ Imputer + DateTime Featurization Component + One Hot␣
↪Encoder + SMOTENC Oversampler + Standard Scaler:
        Starting cross validation
        Finished cross validation - mean Log Loss Binary: 0.512
Decision Tree Classifier w/ Imputer + DateTime Featurization Component + One Hot␣
↪Encoder + SMOTENC Oversampler:
```

(continues on next page)

(continued from previous page)

```
        Starting cross validation
        Finished cross validation - mean Log Loss Binary: 2.957
        High coefficient of variation (cv >= 0.2) within cross validation scores.
        Decision Tree Classifier w/ Imputer + DateTime Featurization Component + One␣
↪Hot Encoder + SMOTENC Oversampler may not perform as estimated on unseen data.
Random Forest Classifier w/ Imputer + DateTime Featurization Component + One Hot␣
↪Encoder + SMOTENC Oversampler:
        Starting cross validation
        Finished cross validation - mean Log Loss Binary: 0.286
LightGBM Classifier w/ Imputer + DateTime Featurization Component + One Hot Encoder +␣
↪SMOTENC Oversampler:
        Starting cross validation
        Finished cross validation - mean Log Loss Binary: 0.309
        High coefficient of variation (cv >= 0.2) within cross validation scores.
        LightGBM Classifier w/ Imputer + DateTime Featurization Component + One Hot␣
↪Encoder + SMOTENC Oversampler may not perform as estimated on unseen data.
Logistic Regression Classifier w/ Imputer + DateTime Featurization Component + One␣
↪Hot Encoder + SMOTENC Oversampler + Standard Scaler:
        Starting cross validation
        Finished cross validation - mean Log Loss Binary: 0.552
XGBoost Classifier w/ Imputer + DateTime Featurization Component + One Hot Encoder +␣
↪SMOTENC Oversampler:
        Starting cross validation
        Finished cross validation - mean Log Loss Binary: 0.279
        High coefficient of variation (cv >= 0.2) within cross validation scores.
        XGBoost Classifier w/ Imputer + DateTime Featurization Component + One Hot␣
↪Encoder + SMOTENC Oversampler may not perform as estimated on unseen data.
Extra Trees Classifier w/ Imputer + DateTime Featurization Component + One Hot␣
↪Encoder + SMOTENC Oversampler:
        Starting cross validation
        Finished cross validation - mean Log Loss Binary: 0.338
CatBoost Classifier w/ Imputer + DateTime Featurization Component + SMOTENC␣
↪Oversampler:
        Starting cross validation
        Finished cross validation - mean Log Loss Binary: 0.602

Search finished after 00:18
Best pipeline: XGBoost Classifier w/ Imputer + DateTime Featurization Component + One␣
↪Hot Encoder + SMOTENC Oversampler
Best pipeline Log Loss Binary: 0.278707
```

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

There are a number of mechanisms to control the AutoML search time. One way is to set the `max_batches` parameter which controls the maximum number of rounds of AutoML to evaluate, where each round may train and score a variable number of pipelines. Another way is to set the `max_iterations` parameter which controls the maximum number of candidate models to be evaluated during AutoML. By default, AutoML will search for a single batch. The first pipeline to be evaluated will always be a baseline model representing a trivial solution.

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

We also provide *a standalone ``search`* method <../generated/evalml.automl.search.html>`__ which does all of the above in a single line, and returns the `AutoMLSearch` instance and data check results. If there were data check errors, AutoML will not be run and no `AutoMLSearch` instance will be returned.

Detecting Problem Type

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

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

[6]:
```
import pandas as pd
from evalml.problem_types import detect_problem_type

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

[6]:
```
<ProblemTypes.BINARY: 'binary'>
```

Objective parameter

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

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

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

[7]:
```
from evalml.automl import get_default_primary_search_objective

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

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

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

Using custom pipelines

EvalML's AutoML algorithm generates a set of pipelines to search with. To provide a custom set instead, set allowed_component_graphs to a dictionary of custom component graphs. `AutoMLSearch` will use these to generate `Pipeline` instances. Note: this will prevent AutoML from generating other pipelines to search over.

[8]:
```
from evalml.pipelines import MulticlassClassificationPipeline


automl_custom = evalml.automl.AutoMLSearch(X_train=X_train,
                                           y_train=y_train,
                                           problem_type='multiclass',
```

(continues on next page)

(continued from previous page)

```
                                                allowed_component_graphs={"My_pipeline": [
↪'Simple Imputer', 'Random Forest Classifier'],
                                                                          "My_other_
↪pipeline": ['One Hot Encoder', 'Random Forest Classifier']})
```

```
Using default limit of max_batches=1.

2 pipelines ready for search.
```

Stopping the search early

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

Callback functions

`AutoMLSearch` supports several callback functions, which can be specified as parameters when initializing an `AutoMLSearch` object. They are:

- `start_iteration_callback`
- `add_result_callback`
- `error_callback`

Start Iteration Callback

Users can set `start_iteration_callback` to set what function is called before each pipeline training iteration. This callback function must take three positional parameters: the pipeline class, the pipeline parameters, and the `AutoMLSearch` object.

[9]:
```
## start_iteration_callback example function
def start_iteration_callback_example(pipeline_class, pipeline_params, automl_obj):
    print ("Training pipeline with the following parameters:", pipeline_params)
```

Add Result Callback

Users can set `add_result_callback` to set what function is called after each pipeline training iteration. This callback function must take three positional parameters: a dictionary containing the training results for the new pipeline, an untrained_pipeline containing the parameters used during training, and the `AutoMLSearch` object.

[10]:
```
## add_result_callback example function
def add_result_callback_example(pipeline_results_dict, untrained_pipeline, automl_
↪obj):
    print ("Results for trained pipeline with the following parameters:", pipeline_
↪results_dict)
```

Error Callback

Users can set the `error_callback` to set what function called when `search()` errors and raises an `Exception`. This callback function takes three positional parameters: the `Exception raised`, the traceback, and the `AutoMLSearch object`. This callback function must also accept `kwargs`, so `AutoMLSearch` is able to pass along other parameters used by default.

Evalml defines several error callback functions, which can be found under `evalml.automl.callbacks`. They are:

- `silent_error_callback`
- `raise_error_callback`
- `log_and_save_error_callback`
- `raise_and_save_error_callback`
- `log_error_callback` (default used when `error_callback` is None)

[11]:
```
# error_callback example; this is implemented in the evalml library
def raise_error_callback(exception, traceback, automl, **kwargs):
    """Raises the exception thrown by the AutoMLSearch object. Also logs the␣
↪exception as an error."""
    logger.error(f'AutoMLSearch raised a fatal exception: {str(exception)}')
    logger.error("\n".join(traceback))
    raise exception
```

4.1.3 View Rankings

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

[12]:
```
automl.rankings
```

[12]:

	id	pipeline_name	search_order	mean_cv_score	standard_deviation_cv_score	validation_score
0	6	XGBoost Classifier w/ Imputer + DateTime Featu...	6	0.278707	0.190725	0.202638
1	3	Random Forest Classifier w/ Imputer + DateTime...	3	0.285613	0.041116	0.263695
2	4	LightGBM Classifier w/ Imputer + DateTime Feat...	4	0.308636	0.203878	0.234947
3	7	Extra Trees Classifier w/ Imputer + DateTime F...	7	0.338286	0.009381	0.329015
4	1	Elastic Net Classifier w/ Imputer + DateTime F...	1	0.511808	0.074992	0.590517
5	5	Logistic Regression Classifier w/ Imputer + Da...	5	0.551960	0.082695	0.628646
6	8	CatBoost Classifier w/ Imputer + DateTime Feat...	8	0.601819	0.007246	0.593693
7	2	Decision Tree Classifier w/ Imputer + DateTime...	2	2.956956	3.084439	0.651557
8	0	Mode Baseline Binary Classification Pipeline	0	3.970423	0.266060	4.124033

(continues on next page)

(continued from previous page)

```
   percent_better_than_baseline  high_variance_cv  \
0                     92.980409              True
1                     92.806475             False
2                     92.226623              True
3                     91.479857             False
4                     87.109474             False
5                     86.098206             False
6                     84.842444             False
7                     25.525413              True
8                      0.000000             False

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

4.1.4 Describe Pipeline

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

[13]:
```
automl.describe_pipeline(1)
```

```
*****************************************************************************************
* Elastic Net Classifier w/ Imputer + DateTime Featurization Component + One Hot␣
↪Encoder + SMOTENC Oversampler + Standard Scaler *
*****************************************************************************************

Problem Type: binary
Model Family: Linear

Pipeline Steps
==============
1. Imputer
         * categorical_impute_strategy : most_frequent
         * numeric_impute_strategy : mean
         * categorical_fill_value : None
         * numeric_fill_value : None
2. DateTime Featurization Component
         * features_to_extract : ['year', 'month', 'day_of_week', 'hour']
         * encode_as_categories : False
         * date_index : None
3. One Hot Encoder
         * top_n : 10
         * features_to_encode : None
         * categories : None
         * drop : if_binary
         * handle_unknown : ignore
```

(continues on next page)

(continued from previous page)

```
        * handle_missing : error
4. SMOTENC Oversampler
        * sampling_ratio : 0.25
        * k_neighbors_default : 5
        * n_jobs : -1
        * sampling_ratio_dict : None
        * categorical_features : [3, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21,␣
↪22, 23, 24, 25, 26, 27, 28, 29, 30, 31, 32, 33, 34, 35, 36, 37, 38, 39, 40, 41, 42,␣
↪43, 44, 45, 46, 47, 48, 49, 50, 51, 52, 53, 54, 55, 56, 57, 58, 59]
        * k_neighbors : 5
5. Standard Scaler
6. Elastic Net Classifier
        * penalty : elasticnet
        * C : 1.0
        * l1_ratio : 0.15
        * n_jobs : -1
        * multi_class : auto
        * solver : saga

Training
========
Training for binary problems.
Total training time (including CV): 2.3 seconds

Cross Validation
----------------
             Log Loss Binary  MCC Binary   AUC  Precision    F1  Balanced Accuracy␣
↪Binary  Accuracy Binary # Training # Validation
0                      0.591       0.104 0.597      0.200 0.222                    0.
↪557            0.791        133           67
1                      0.441       0.296 0.710      0.500 0.333                    0.
↪608            0.881        133           67
2                      0.504       0.000 0.630      0.000 0.000                    0.
↪500            0.894        134           66
mean                   0.512       0.133 0.646      0.233 0.185                    0.
↪555            0.855          -           -
std                    0.075       0.150 0.058      0.252 0.170                    0.
↪054            0.056          -           -
coef of var            0.147       1.126 0.090      1.079 0.917                    0.
↪097            0.065          -           -
```

4.1.5 Get Pipeline

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

[14]:
```
pipeline = automl.get_pipeline(1)
print(pipeline.name)
print(pipeline.parameters)
```

```
Elastic Net Classifier w/ Imputer + DateTime Featurization Component + One Hot␣
↪Encoder + SMOTENC Oversampler + Standard Scaler
{'Imputer': {'categorical_impute_strategy': 'most_frequent', 'numeric_impute_strategy
↪': 'mean', 'categorical_fill_value': None, 'numeric_fill_value': None}, 'DateTime␣
↪Featurization Component': {'features_to_extract': ['year', 'month', 'day_of_week',
↪'hour'], 'encode_as_categories': False, 'date_index': None}, 'One Hot Encoder': {
↪'top_n': 10, 'features_to_encode': None, 'categories': None, 'drop': 'if_binary',
↪'handle_unknown': 'ignore', 'handle_missing': 'error'}, 'SMOTENC Oversampler': {
↪'sampling_ratio': 0.25, 'k_neighbors_default': 5, 'n_jobs': -1, 'sampling_ratio_dict
↪': None, 'categorical_features': [3, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21,
↪ 22, 23, 24, 25, 26, 27, 28, 29, 30, 31, 32, 33, 34, 35, 36, 37, 38, 39, 40, 41, 42,
↪ 43, 44, 45, 46, 47, 48, 49, 50, 51, 52, 53, 54, 55, 56, 57, 58, 59], 'k_neighbors':
↪ 5}, 'Elastic Net Classifier': {'penalty': 'elasticnet', 'C': 1.0, 'l1_ratio': 0.15,
↪ 'n_jobs': -1, 'multi_class': 'auto', 'solver': 'saga'}}
```

(continues on next page)

(continued from previous page)

Get best pipeline

If you specifically want to get the best pipeline, there is a convenient accessor for that. The pipeline returned is already fitted on the input X, y data that we passed to AutoMLSearch. To turn off this default behavior, set `train_best_pipeline=False` when initializing AutoMLSearch.

[15]:
```
best_pipeline = automl.best_pipeline
print(best_pipeline.name)
print(best_pipeline.parameters)
best_pipeline.predict(X_train)
```

```
XGBoost Classifier w/ Imputer + DateTime Featurization Component + One Hot Encoder +␣
↪SMOTENC Oversampler
{'Imputer': {'categorical_impute_strategy': 'most_frequent', 'numeric_impute_strategy
↪': 'mean', 'categorical_fill_value': None, 'numeric_fill_value': None}, 'DateTime␣
↪Featurization Component': {'features_to_extract': ['year', 'month', 'day_of_week',
↪'hour'], 'encode_as_categories': False, 'date_index': None}, 'One Hot Encoder': {
↪'top_n': 10, 'features_to_encode': None, 'categories': None, 'drop': 'if_binary',
↪'handle_unknown': 'ignore', 'handle_missing': 'error'}, 'SMOTENC Oversampler': {
↪'sampling_ratio': 0.25, 'k_neighbors_default': 5, 'n_jobs': -1, 'sampling_ratio_dict
↪': None, 'categorical_features': [3, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21,
↪ 22, 23, 24, 25, 26, 27, 28, 29, 30, 31, 32, 33, 34, 35, 36, 37, 38, 39, 40, 41, 42,
↪ 43, 44, 45, 46, 47, 48, 49, 50, 51, 52, 53, 54, 55, 56, 57, 58, 59], 'k_neighbors':
↪ 5}, 'XGBoost Classifier': {'eta': 0.1, 'max_depth': 6, 'min_child_weight': 1, 'n_
↪estimators': 100, 'n_jobs': -1}}
```

[15]:
```
0      False
1      False
2      False
3       True
4      False
       ...
195    False
196    False
197    False
198    False
199    False
Name: fraud, Length: 200, dtype: bool
```

4.1.6 Training and Scoring Multiple Pipelines using AutoMLSearch

AutoMLSearch will automatically fit the best pipeline on the entire training data. It also provides an easy API for training and scoring other pipelines.

If you'd like to train one or more pipelines on the entire training data, you can use the `train_pipelines`method

Similarly, if you'd like to score one or more pipelines on a particular dataset, you can use the `train_pipelines`method

[16]:
```
trained_pipelines = automl.train_pipelines([automl.get_pipeline(i) for i in [0, 1,␣
↪2]])
trained_pipelines
```

[16]:
```
{'Mode Baseline Binary Classification Pipeline': pipeline =␣
↪BinaryClassificationPipeline(component_graph={'Baseline Classifier': ['Baseline␣
↪Classifier']}, parameters={'Baseline Classifier':{'strategy': 'mode'}}, custom_name=
↪'Mode Baseline Binary Classification Pipeline', random_seed=0),
 'Elastic Net Classifier w/ Imputer + DateTime Featurization Component + One Hot␣
↪Encoder + SMOTENC Oversampler + Standard Scaler': pipeline =␣
↪BinaryClassificationPipeline(component_graph={'Imputer': ['Imputer'], 'DateTime␣
↪Featurization Component': ['DateTime Featurization Component', 'Imputer.x'], 'One␣
↪Hot Encoder': ['One Hot Encoder', 'DateTime Featurization Component.x'], 'SMOTENC␣
↪Oversampler': ['SMOTENC Oversampler', 'One Hot Encoder.x'], 'Standard Scaler': [
↪'Standard Scaler', 'SMOTENC Oversampler.x', 'SMOTENC Oversampler.y'], 'Elastic Net␣
↪Classifier': ['Elastic Net Classifier', 'Standard Scaler.x']}, parameters={'Imputer
↪':{'categorical_impute_strategy': 'most_frequent', 'numeric_impute_strategy': 'mean
↪', 'categorical_fill_value': None, 'numeric_fill_value': None}, 'DateTime␣
↪Featurization Component':{'features_to_extract': ['year', 'month', 'day_of_week',
↪'hour'], 'encode_as_categories': False, 'date_index': None}, 'One Hot Encoder':{
↪'top_n': 10, 'features_to_encode': None, 'categories': None, 'drop': 'if_binary',
↪'handle_unknown': 'ignore', 'handle_missing': 'error'}, 'SMOTENC Oversampler':{
↪'sampling_ratio': 0.25, 'k_neighbors_default': 5, 'n_jobs': -1, 'sampling_ratio_dict
↪': None, 'categorical_features': [3, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21,
↪ 22, 23, 24, 25, 26, 27, 28, 29, 30, 31, 32, 33, 34, 35, 36, 37, 38, 39, 40, 41, 42,
↪ 43, 44, 45, 46, 47, 48, 49, 50, 51, 52, 53, 54, 55, 56, 57, 58, 59], 'k_neighbors':
↪ 5}, 'Elastic Net Classifier':{'penalty': 'elasticnet', 'C': 1.0, 'l1_ratio': 0.15,
↪'n_jobs': -1, 'multi_class': 'auto', 'solver': 'saga'}}, random_seed=0),
 'Decision Tree Classifier w/ Imputer + DateTime Featurization Component + One Hot␣
↪Encoder + SMOTENC Oversampler': pipeline = BinaryClassificationPipeline(component_
↪graph={'Imputer': ['Imputer'], 'DateTime Featurization Component': ['DateTime␣
↪Featurization Component', 'Imputer.x'], 'One Hot Encoder': ['One Hot Encoder',
↪'DateTime Featurization Component.x'], 'SMOTENC Oversampler': ['SMOTENC Oversampler
↪', 'One Hot Encoder.x'], 'Decision Tree Classifier': ['Decision Tree Classifier',
↪'SMOTENC Oversampler.x', 'SMOTENC Oversampler.y']}, parameters={'Imputer':{
↪'categorical_impute_strategy': 'most_frequent', 'numeric_impute_strategy': 'mean',
↪'categorical_fill_value': None, 'numeric_fill_value': None}, 'DateTime␣
↪Featurization Component':{'features_to_extract': ['year', 'month', 'day_of_week',
↪'hour'], 'encode_as_categories': False, 'date_index': None}, 'One Hot Encoder':{
↪'top_n': 10, 'features_to_encode': None, 'categories': None, 'drop': 'if_binary',
↪'handle_unknown': 'ignore', 'handle_missing': 'error'}, 'SMOTENC Oversampler':{
↪'sampling_ratio': 0.25, 'k_neighbors_default': 5, 'n_jobs': -1, 'sampling_ratio_dict
↪': None, 'categorical_features': [3, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21,
↪ 22, 23, 24, 25, 26, 27, 28, 29, 30, 31, 32, 33, 34, 35, 36, 37, 38, 39, 40, 41, 42,
↪ 43, 44, 45, 46, 47, 48, 49, 50, 51, 52, 53, 54, 55, 56, 57, 58, 59], 'k_neighbors':
↪ 5}, 'Decision Tree Classifier':{'criterion': 'gini', 'max_features': 'auto', 'max_
↪depth': 6, 'min_samples_split': 2, 'min_weight_fraction_leaf': 0.0}}, random_
↪seed=0)}
```

[17]:
```
pipeline_holdout_scores = automl.score_pipelines([trained_pipelines[name] for name in␣
↪trained_pipelines.keys()],
                                                 X_holdout,
                                                 y_holdout,
                                                 ['Accuracy Binary', 'F1', 'AUC'])
pipeline_holdout_scores
```

[17]:
```
{'Mode Baseline Binary Classification Pipeline': OrderedDict([('Accuracy Binary',
              0.88),
             ('F1', 0.0),
             ('AUC', 0.5)]),
```

(continues on next page)

(continued from previous page)

```
 'Elastic Net Classifier w/ Imputer + DateTime Featurization Component + One Hot␣
↪Encoder + SMOTENC Oversampler + Standard Scaler': OrderedDict([('Accuracy Binary',
               0.88),
              ('F1', 0.0),
              ('AUC', 0.5265151515151515)]),
 'Decision Tree Classifier w/ Imputer + DateTime Featurization Component + One Hot␣
↪Encoder + SMOTENC Oversampler': OrderedDict([('Accuracy Binary',
               0.92),
              ('F1', 0.6),
              ('AUC', 0.7234848484848486)])}
```

4.1.7 Saving AutoMLSearch and pipelines from AutoMLSearch

There are two ways to save results from AutoMLSearch.

- You can save the AutoMLSearch object itself, calling `.save(<filepath>)` to do so. This will allow you to save the AutoMLSearch state and reload all pipelines from this.
- If you want to save a pipeline from AutoMLSearch for future use, pipeline classes themselves have a `.save(<filepath>)` method.

[18]:
```
# saving the entire automl search
automl.save("automl.cloudpickle")
automl2 = evalml.automl.AutoMLSearch.load("automl.cloudpickle")
# saving the best pipeline using .save()
best_pipeline.save("pipeline.cloudpickle")
best_pipeline_copy = evalml.pipelines.PipelineBase.load("pipeline.cloudpickle")
```

4.1.8 Limiting the AutoML Search Space

The AutoML search algorithm first trains each component in the pipeline with their default values. After the first iteration, it then tweaks the parameters of these components using the pre-defined hyperparameter ranges that these components have. To limit the search over certain hyperparameter ranges, you can specify a `custom_hyperparameters` argument with your `AutoMLSearch` parameters. These parameters will limit the hyperparameter search space.

Hyperparameter ranges can be found through the API reference for each component. Parameter arguments must be specified as dictionaries, but the associated values can be single values or `skopt.space` Real, Integer, Categorical values.

If however you'd like to specify certain values for the initial batch of the AutoML search algorithm, you can use the `pipeline_parameters` argument. This will set the initial batch's component parameters to the values passed by this argument.

[19]:
```
from evalml import AutoMLSearch
from evalml.demos import load_fraud
from skopt.space import Categorical
from evalml.model_family import ModelFamily
import woodwork as ww

X, y = load_fraud(n_rows=1000)

# example of setting parameter to just one value
custom_hyperparameters = {'Imputer': {
    'numeric_impute_strategy': 'mean'
```

(continues on next page)

(continued from previous page)

```
}}


# limit the numeric impute strategy to include only `median` and `most_frequent`
# `mean` is the default value for this argument, but it doesn't need to be included␣
↪in the specified hyperparameter range for this to work
custom_hyperparameters = {'Imputer': {
    'numeric_impute_strategy': Categorical(['median', 'most_frequent'])
}}
# set the initial batch numeric impute strategy strategy to 'median'
pipeline_parameters = {'Imputer': {
    'numeric_impute_strategy': 'median'
}}

# using this custom hyperparameter means that our Imputer components in these␣
↪pipelines will only search through
# 'median' and 'most_frequent' strategies for 'numeric_impute_strategy', and the␣
↪initial batch parameter will be
# set to 'median'
automl_constrained = AutoMLSearch(X_train=X, y_train=y, problem_type='binary',␣
↪pipeline_parameters=pipeline_parameters,
                                  custom_hyperparameters=custom_hyperparameters)
```

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

Number of training examples: 1000
Targets
False    85.90%
True     14.10%
Name: fraud, dtype: object
Using default limit of max_batches=1.

Generating pipelines to search over...
8 pipelines ready for search.
```

4.1.9 Imbalanced Data

The AutoML search algorithm now has functionality to handle imbalanced data during classification! AutoMLSearch now provides two additional parameters, `sampler_method` and `sampler_balanced_ratio`, that allow you to let AutoMLSearch know whether to sample imbalanced data, and how to do so. `sampler_method` takes in either `Undersampler`, `Oversampler`, `auto`, or None as the sampler to use, and `sampler_balanced_ratio` specifies the `minority/majority` ratio that you want to sample to. Details on the Undersampler and Oversampler components can be found in the documentation.

This can be used for imbalanced datasets, like the fraud dataset, which has a 'minority:majority' ratio of < 0.2.

[20]:
```
automl_auto = AutoMLSearch(X_train=X, y_train=y, problem_type='binary')
automl_auto.allowed_pipelines[-1]
```

```
Using default limit of max_batches=1.

Generating pipelines to search over...
8 pipelines ready for search.
```

[20]:
```
pipeline = BinaryClassificationPipeline(component_graph={'Imputer': ['Imputer'],
↪'DateTime Featurization Component': ['DateTime Featurization Component', 'Imputer.x
↪'], 'One Hot Encoder': ['One Hot Encoder', 'DateTime Featurization Component.x'],
↪'SMOTENC Oversampler': ['SMOTENC Oversampler', 'One Hot Encoder.x'], 'Standard␣
↪Scaler': ['Standard Scaler', 'SMOTENC Oversampler.x', 'SMOTENC Oversampler.y'],
↪'Logistic Regression Classifier': ['Logistic Regression Classifier', 'Standard␣
↪Scaler.x']}, parameters={'Imputer':{'categorical_impute_strategy': 'most_frequent',
↪'numeric_impute_strategy': 'mean', 'categorical_fill_value': None, 'numeric_fill_
↪value': None}, 'DateTime Featurization Component':{'features_to_extract': ['year',
↪'month', 'day_of_week', 'hour'], 'encode_as_categories': False, 'date_index': None},
↪ 'One Hot Encoder':{'top_n': 10, 'features_to_encode': None, 'categories': None,
↪'drop': 'if_binary', 'handle_unknown': 'ignore', 'handle_missing': 'error'},
↪'SMOTENC Oversampler':{'sampling_ratio': 0.25, 'k_neighbors_default': 5, 'n_jobs': -
↪1, 'sampling_ratio_dict': None}, 'Logistic Regression Classifier':{'penalty': 'l2',
↪'C': 1.0, 'n_jobs': -1, 'multi_class': 'auto', 'solver': 'lbfgs'}}, random_seed=0)
```

The SMOTENC Oversampler is chosen as the default sampling component here, since the `sampler_balanced_ratio = 0.25`. If you specified a lower ratio, for instance `sampler_balanced_ratio = 0.1`, then there would be no sampling component added here. This is because if a ratio of 0.1 would be considered balanced, then a ratio of 0.2 would also be balanced.

[21]:
```
automl_auto_ratio = AutoMLSearch(X_train=X, y_train=y, problem_type='binary', sampler_
↪balanced_ratio=0.1)
automl_auto_ratio.allowed_pipelines[-1]
```

```
Using default limit of max_batches=1.

Generating pipelines to search over...
8 pipelines ready for search.
```

[21]:
```
pipeline = BinaryClassificationPipeline(component_graph={'Imputer': ['Imputer'],
↪'DateTime Featurization Component': ['DateTime Featurization Component', 'Imputer.x
↪'], 'One Hot Encoder': ['One Hot Encoder', 'DateTime Featurization Component.x'],
↪'Standard Scaler': ['Standard Scaler', 'One Hot Encoder.x'], 'Logistic Regression␣
↪Classifier': ['Logistic Regression Classifier', 'Standard Scaler.x']}, parameters={
↪'Imputer':{'categorical_impute_strategy': 'most_frequent', 'numeric_impute_strategy
↪': 'mean', 'categorical_fill_value': None, 'numeric_fill_value': None}, 'DateTime␣
↪Featurization Component':{'features_to_extract': ['year', 'month', 'day_of_week',
↪'hour'], 'encode_as_categories': False, 'date_index': None}, 'One Hot Encoder':{
↪'top_n': 10, 'features_to_encode': None, 'categories': None, 'drop': 'if_binary',
↪'handle_unknown': 'ignore', 'handle_missing': 'error'}, 'Logistic Regression␣
↪Classifier':{'penalty': 'l2', 'C': 1.0, 'n_jobs': -1, 'multi_class': 'auto', 'solver
↪': 'lbfgs'}}, random_seed=0)
```

Additionally, you can add more fine-grained sampling ratios by passing in a `sampling_ratio_dict` in pipeline parameters. For this dictionary, AutoMLSearch expects the keys to be int values from 0 to `n-1` for the classes, and the values would be the `sampler_balanced__ratio` associated with each target. This dictionary would override the AutoML argument `sampler_balanced_ratio`. Below, you can see the scenario for Oversampler component on this dataset. Note that the logic for Undersamplers is included in the commented section.

[22]:
```
# In this case, the majority class is the negative class
# for the oversampler, we don't want to oversample this class, so class 0 (majority)␣
↪will have a ratio of 1 to itself
# for the minority class 1, we want to oversample it to have a minority/majority␣
↪ratio of 0.5, which means we want minority to have 1/2 the samples as the minority
sampler_ratio_dict = {0: 1, 1: 0.5}
pipeline_parameters = {"SMOTENC Oversampler": {"sampler_balanced_ratio": sampler_
↪ratio_dict}}
```

(continues on next page)

(continued from previous page)

```
automl_auto_ratio_dict = AutoMLSearch(X_train=X, y_train=y, problem_type='binary',␣
↪pipeline_parameters=pipeline_parameters)
automl_auto_ratio_dict.allowed_pipelines[-1]

# Undersampler case
# we don't want to undersample this class, so class 1 (minority) will have a ratio of␣
↪1 to itself
# for the majority class 0, we want to undersample it to have a minority/majority␣
↪ratio of 0.5, which means we want majority to have 2x the samples as the minority
# sampler_ratio_dict = {0: 0.5, 1: 1}
# pipeline_parameters = {"SMOTENC Oversampler": {"sampler_balanced_ratio": sampler_
↪ratio_dict}}
# automl_auto_ratio_dict = AutoMLSearch(X_train=X, y_train=y, problem_type='binary',␣
↪pipeline_parameters=pipeline_parameters)
```

```
Using default limit of max_batches=1.

Generating pipelines to search over...
8 pipelines ready for search.
```

[22]:
```
pipeline = BinaryClassificationPipeline(component_graph={'Imputer': ['Imputer'],
↪'DateTime Featurization Component': ['DateTime Featurization Component', 'Imputer.x
↪'], 'One Hot Encoder': ['One Hot Encoder', 'DateTime Featurization Component.x'],
↪'SMOTENC Oversampler': ['SMOTENC Oversampler', 'One Hot Encoder.x'], 'Standard␣
↪Scaler': ['Standard Scaler', 'SMOTENC Oversampler.x', 'SMOTENC Oversampler.y'],
↪'Logistic Regression Classifier': ['Logistic Regression Classifier', 'Standard␣
↪Scaler.x']}, parameters={'Imputer':{'categorical_impute_strategy': 'most_frequent',
↪'numeric_impute_strategy': 'mean', 'categorical_fill_value': None, 'numeric_fill_
↪value': None}, 'DateTime Featurization Component':{'features_to_extract': ['year',
↪'month', 'day_of_week', 'hour'], 'encode_as_categories': False, 'date_index': None},
↪ 'One Hot Encoder':{'top_n': 10, 'features_to_encode': None, 'categories': None,
↪'drop': 'if_binary', 'handle_unknown': 'ignore', 'handle_missing': 'error'},
↪'SMOTENC Oversampler':{'sampling_ratio': 0.25, 'k_neighbors_default': 5, 'n_jobs': -
↪1, 'sampling_ratio_dict': None, 'sampler_balanced_ratio': {0: 1, 1: 0.5}},
↪'Logistic Regression Classifier':{'penalty': 'l2', 'C': 1.0, 'n_jobs': -1, 'multi_
↪class': 'auto', 'solver': 'lbfgs'}}, random_seed=0)
```

4.1.10 Adding ensemble methods to AutoML

Stacking

Stacking is an ensemble machine learning algorithm that involves training a model to best combine the predictions of several base learning algorithms. First, each base learning algorithms is trained using the given data. Then, the combining algorithm or meta-learner is trained on the predictions made by those base learning algorithms to make a final prediction.

AutoML enables stacking using the `ensembling` flag during initalization; this is set to `False` by default. The stacking ensemble pipeline runs in its own batch after a whole cycle of training has occurred (each allowed pipeline trains for one batch). Note that this means **a large number of iterations may need to run before the stacking ensemble runs**. It is also important to note that **only the first CV fold is calculated for stacking ensembles** because the model internally uses CV folds.

[23]:
```
X, y = evalml.demos.load_breast_cancer()
```

(continues on next page)

(continued from previous page)

```
automl_with_ensembling = AutoMLSearch(X_train=X, y_train=y,
                                      problem_type="binary",
                                      allowed_model_families=[ModelFamily.LINEAR_
↪MODEL],
                                      max_batches=4,
                                      ensembling=True)
automl_with_ensembling.search()
```

```
         Number of Features
Numeric                  30

Number of training examples: 569
Targets
benign       62.74%
malignant    37.26%
Name: target, dtype: object
Generating pipelines to search over...
2 pipelines ready for search.
Ensembling will run every 3 batches.

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

Optimizing for Log Loss Binary.
Lower score is better.

Using SequentialEngine to train and score pipelines.
Searching up to 4 batches for a total of 14 pipelines.
Allowed model families: linear_model
```

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

```
Evaluating Baseline Pipeline: Mode Baseline Binary Classification Pipeline
Mode Baseline Binary Classification Pipeline:
        Starting cross validation
        Finished cross validation - mean Log Loss Binary: 12.868

*****************************
* Evaluating Batch Number 1 *
*****************************

Elastic Net Classifier w/ Imputer + Standard Scaler:
        Starting cross validation
        Finished cross validation - mean Log Loss Binary: 0.077
Logistic Regression Classifier w/ Imputer + Standard Scaler:
        Starting cross validation
        Finished cross validation - mean Log Loss Binary: 0.077
        High coefficient of variation (cv >= 0.2) within cross validation scores.
        Logistic Regression Classifier w/ Imputer + Standard Scaler may not perform␣
↪as estimated on unseen data.

*****************************
* Evaluating Batch Number 2 *
```

(continues on next page)

(continued from previous page)

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

Logistic Regression Classifier w/ Imputer + Standard Scaler:
        Starting cross validation
        Finished cross validation - mean Log Loss Binary: 0.097
        High coefficient of variation (cv >= 0.2) within cross validation scores.
        Logistic Regression Classifier w/ Imputer + Standard Scaler may not perform␣
↪as estimated on unseen data.
Logistic Regression Classifier w/ Imputer + Standard Scaler:
        Starting cross validation
        Finished cross validation - mean Log Loss Binary: 0.085
        High coefficient of variation (cv >= 0.2) within cross validation scores.
        Logistic Regression Classifier w/ Imputer + Standard Scaler may not perform␣
↪as estimated on unseen data.
Logistic Regression Classifier w/ Imputer + Standard Scaler:
        Starting cross validation
        Finished cross validation - mean Log Loss Binary: 0.097
        High coefficient of variation (cv >= 0.2) within cross validation scores.
        Logistic Regression Classifier w/ Imputer + Standard Scaler may not perform␣
↪as estimated on unseen data.
Logistic Regression Classifier w/ Imputer + Standard Scaler:
        Starting cross validation
        Finished cross validation - mean Log Loss Binary: 0.091
        High coefficient of variation (cv >= 0.2) within cross validation scores.
        Logistic Regression Classifier w/ Imputer + Standard Scaler may not perform␣
↪as estimated on unseen data.
Logistic Regression Classifier w/ Imputer + Standard Scaler:
        Starting cross validation
        Finished cross validation - mean Log Loss Binary: 0.080

*****************************
* Evaluating Batch Number 3 *
*****************************

Elastic Net Classifier w/ Imputer + Standard Scaler:
        Starting cross validation
        Finished cross validation - mean Log Loss Binary: 0.075
        High coefficient of variation (cv >= 0.2) within cross validation scores.
        Elastic Net Classifier w/ Imputer + Standard Scaler may not perform as␣
↪estimated on unseen data.
Elastic Net Classifier w/ Imputer + Standard Scaler:
        Starting cross validation
        Finished cross validation - mean Log Loss Binary: 0.075
        High coefficient of variation (cv >= 0.2) within cross validation scores.
        Elastic Net Classifier w/ Imputer + Standard Scaler may not perform as␣
↪estimated on unseen data.
Elastic Net Classifier w/ Imputer + Standard Scaler:
        Starting cross validation
        Finished cross validation - mean Log Loss Binary: 0.079
Elastic Net Classifier w/ Imputer + Standard Scaler:
        Starting cross validation
        Finished cross validation - mean Log Loss Binary: 0.076
        High coefficient of variation (cv >= 0.2) within cross validation scores.
        Elastic Net Classifier w/ Imputer + Standard Scaler may not perform as␣
↪estimated on unseen data.
Elastic Net Classifier w/ Imputer + Standard Scaler:
        Starting cross validation
```

(continues on next page)

(continued from previous page)

```
        Finished cross validation - mean Log Loss Binary: 0.075
        High coefficient of variation (cv >= 0.2) within cross validation scores.
        Elastic Net Classifier w/ Imputer + Standard Scaler may not perform as␣
↪estimated on unseen data.

*****************************
* Evaluating Batch Number 4 *
*****************************

Stacked Ensemble Classification Pipeline:
        Starting cross validation
        Finished cross validation - mean Log Loss Binary: 0.123

Search finished after 00:20
Best pipeline: Elastic Net Classifier w/ Imputer + Standard Scaler
Best pipeline Log Loss Binary: 0.075387
```

We can view more information about the stacking ensemble pipeline (which was the best performing pipeline) by calling `.describe()`.

[24]:
```
automl_with_ensembling.best_pipeline.describe()
```

```


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

Problem Type: binary
Model Family: Linear
Number of features: 30

Pipeline Steps
==============
1. Imputer
         * categorical_impute_strategy : most_frequent
         * numeric_impute_strategy : median
         * categorical_fill_value : None
         * numeric_fill_value : None
2. Standard Scaler
3. Elastic Net Classifier
         * penalty : elasticnet
         * C : 8.123565600467177
         * l1_ratio : 0.47997717237505744
         * n_jobs : -1
         * multi_class : auto
         * solver : saga
```

4.1.11 Access raw results

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

[25]:
```
automl.results
```

[25]:
```
{'pipeline_results': {0: {'id': 0,
   'pipeline_name': 'Mode Baseline Binary Classification Pipeline',
   'pipeline_class': evalml.pipelines.binary_classification_pipeline.
↪BinaryClassificationPipeline,
   'pipeline_summary': 'Baseline Classifier',
   'parameters': {'Baseline Classifier': {'strategy': 'mode'}},
   'mean_cv_score': 3.970423187263591,
   'standard_deviation_cv_score': 0.26606000431837074,
   'high_variance_cv': False,
   'training_time': 0.18905210494995117,
   'cv_data': [{'all_objective_scores': OrderedDict([('Log Loss Binary',
                 4.124033002377396),
                ('MCC Binary', 0.0),
                ('AUC', 0.5),
                ('Precision', 0.0),
                ('F1', 0.0),
                ('Balanced Accuracy Binary', 0.5),
                ('Accuracy Binary', 0.8805970149253731),
                ('# Training', 133),
                ('# Validation', 67)]),
     'mean_cv_score': 4.124033002377396,
     'binary_classification_threshold': 0.9999940391390134},
    {'all_objective_scores': OrderedDict([('Log Loss Binary',
                 4.124033002377395),
                ('MCC Binary', 0.0),
                ('AUC', 0.5),
                ('Precision', 0.0),
                ('F1', 0.0),
                ('Balanced Accuracy Binary', 0.5),
                ('Accuracy Binary', 0.8805970149253731),
                ('# Training', 133),
                ('# Validation', 67)]),
     'mean_cv_score': 4.124033002377395,
     'binary_classification_threshold': 0.9999940391390134},
    {'all_objective_scores': OrderedDict([('Log Loss Binary',
                 3.6632035570359824),
                ('MCC Binary', 0.0),
                ('AUC', 0.5),
                ('Precision', 0.0),
                ('F1', 0.0),
                ('Balanced Accuracy Binary', 0.5),
                ('Accuracy Binary', 0.8939393939393939),
                ('# Training', 134),
                ('# Validation', 66)]),
     'mean_cv_score': 3.6632035570359824,
     'binary_classification_threshold': 0.9999940391390134}],
   'percent_better_than_baseline_all_objectives': {'Log Loss Binary': 0,
    'MCC Binary': 0,
    'AUC': 0,
    'Precision': 0,
    'F1': 0,
```

(continues on next page)

(continued from previous page)

```
    'Balanced Accuracy Binary': 0,
    'Accuracy Binary': 0},
   'percent_better_than_baseline': 0,
   'validation_score': 4.124033002377396},
  1: {'id': 1,
   'pipeline_name': 'Elastic Net Classifier w/ Imputer + DateTime Featurization␣
↪Component + One Hot Encoder + SMOTENC Oversampler + Standard Scaler',
   'pipeline_class': evalml.pipelines.binary_classification_pipeline.
↪BinaryClassificationPipeline,
   'pipeline_summary': 'Elastic Net Classifier w/ Imputer + DateTime Featurization␣
↪Component + One Hot Encoder + SMOTENC Oversampler + Standard Scaler',
   'parameters': {'Imputer': {'categorical_impute_strategy': 'most_frequent',
     'numeric_impute_strategy': 'mean',
     'categorical_fill_value': None,
     'numeric_fill_value': None},
    'DateTime Featurization Component': {'features_to_extract': ['year',
      'month',
      'day_of_week',
      'hour'],
     'encode_as_categories': False,
     'date_index': None},
    'One Hot Encoder': {'top_n': 10,
     'features_to_encode': None,
     'categories': None,
     'drop': 'if_binary',
     'handle_unknown': 'ignore',
     'handle_missing': 'error'},
    'SMOTENC Oversampler': {'sampling_ratio': 0.25,
     'k_neighbors_default': 5,
     'n_jobs': -1,
     'sampling_ratio_dict': None,
     'categorical_features': [3,
      10,
      11,
      12,
      13,
      14,
      15,
      16,
      17,
      18,
      19,
      20,
      21,
      22,
      23,
      24,
      25,
      26,
      27,
      28,
      29,
      30,
      31,
      32,
      33,
      34,
```

(continues on next page)

(continued from previous page)

```
     35,
     36,
     37,
     38,
     39,
     40,
     41,
     42,
     43,
     44,
     45,
     46,
     47,
     48,
     49,
     50,
     51,
     52,
     53,
     54,
     55,
     56,
     57,
     58,
     59],
    'k_neighbors': 5},
   'Elastic Net Classifier': {'penalty': 'elasticnet',
    'C': 1.0,
    'l1_ratio': 0.15,
    'n_jobs': -1,
    'multi_class': 'auto',
    'solver': 'saga'}},
  'mean_cv_score': 0.5118084226551022,
  'standard_deviation_cv_score': 0.07499222153032764,
  'high_variance_cv': False,
  'training_time': 2.3085663318634033,
  'cv_data': [{'all_objective_scores': OrderedDict([('Log Loss Binary',
                 0.590516914375052),
                ('MCC Binary', 0.10410829467682868),
                ('AUC', 0.597457627118644),
                ('Precision', 0.2),
                ('F1', 0.22222222222222224),
                ('Balanced Accuracy Binary', 0.5572033898305084),
                ('Accuracy Binary', 0.7910447761194029),
                ('# Training', 133),
                ('# Validation', 67)]),
    'mean_cv_score': 0.590516914375052,
    'binary_classification_threshold': 0.33812616662473555},
   {'all_objective_scores': OrderedDict([('Log Loss Binary',
                 0.44118816828058105),
                ('MCC Binary', 0.2957528252716689),
                ('AUC', 0.7097457627118644),
                ('Precision', 0.5),
                ('F1', 0.3333333333333333),
                ('Balanced Accuracy Binary', 0.6080508474576272),
                ('Accuracy Binary', 0.8805970149253731),
                ('# Training', 133),
```

(continues on next page)

(continued from previous page)

```
                  ('# Validation', 67)]),
    'mean_cv_score': 0.44118816828058105,
    'binary_classification_threshold': 0.6737665382921127},
   {'all_objective_scores': OrderedDict([('Log Loss Binary',
                  0.5037201853096736),
                 ('MCC Binary', 0.0),
                 ('AUC', 0.6295399515738498),
                 ('Precision', 0.0),
                 ('F1', 0.0),
                 ('Balanced Accuracy Binary', 0.5),
                 ('Accuracy Binary', 0.8939393939393939),
                 ('# Training', 134),
                 ('# Validation', 66)]),
    'mean_cv_score': 0.5037201853096736,
    'binary_classification_threshold': 0.9999940391390134}],
  'percent_better_than_baseline_all_objectives': {'Log Loss Binary': 87.
↪10947426720426,
   'MCC Binary': inf,
   'AUC': 14.558111380145277,
   'Precision': 23.333333333333332,
   'F1': 18.51851851851852,
   'Balanced Accuracy Binary': 5.508474576271183,
   'Accuracy Binary': -2.985074626865658},
  'percent_better_than_baseline': 87.10947426720426,
  'validation_score': 0.590516914375052},
 2: {'id': 2,
  'pipeline_name': 'Decision Tree Classifier w/ Imputer + DateTime Featurization␣
↪Component + One Hot Encoder + SMOTENC Oversampler',
  'pipeline_class': evalml.pipelines.binary_classification_pipeline.
↪BinaryClassificationPipeline,
  'pipeline_summary': 'Decision Tree Classifier w/ Imputer + DateTime Featurization␣
↪Component + One Hot Encoder + SMOTENC Oversampler',
  'parameters': {'Imputer': {'categorical_impute_strategy': 'most_frequent',
    'numeric_impute_strategy': 'mean',
    'categorical_fill_value': None,
    'numeric_fill_value': None},
   'DateTime Featurization Component': {'features_to_extract': ['year',
     'month',
     'day_of_week',
     'hour'],
    'encode_as_categories': False,
    'date_index': None},
   'One Hot Encoder': {'top_n': 10,
    'features_to_encode': None,
    'categories': None,
    'drop': 'if_binary',
    'handle_unknown': 'ignore',
    'handle_missing': 'error'},
   'SMOTENC Oversampler': {'sampling_ratio': 0.25,
    'k_neighbors_default': 5,
    'n_jobs': -1,
    'sampling_ratio_dict': None,
    'categorical_features': [3,
     10,
     11,
     12,
     13,
```

(continues on next page)

(continued from previous page)

```
      14,
      15,
      16,
      17,
      18,
      19,
      20,
      21,
      22,
      23,
      24,
      25,
      26,
      27,
      28,
      29,
      30,
      31,
      32,
      33,
      34,
      35,
      36,
      37,
      38,
      39,
      40,
      41,
      42,
      43,
      44,
      45,
      46,
      47,
      48,
      49,
      50,
      51,
      52,
      53,
      54,
      55,
      56,
      57,
      58,
      59],
     'k_neighbors': 5},
    'Decision Tree Classifier': {'criterion': 'gini',
     'max_features': 'auto',
     'max_depth': 6,
     'min_samples_split': 2,
     'min_weight_fraction_leaf': 0.0}},
   'mean_cv_score': 2.956956288246977,
   'standard_deviation_cv_score': 3.084438905133218,
   'high_variance_cv': True,
   'training_time': 1.824521780014038,
   'cv_data': [{'all_objective_scores': OrderedDict([('Log Loss Binary',
```

(continues on next page)

(continued from previous page)

```
                  0.651556943513563),
                 ('MCC Binary', 0.7712055916006633),
                 ('AUC', 0.8103813559322034),
                 ('Precision', 1.0),
                 ('F1', 0.7692307692307693),
                 ('Balanced Accuracy Binary', 0.8125),
                 ('Accuracy Binary', 0.9552238805970149),
                 ('# Training', 133),
                 ('# Validation', 67)]),
    'mean_cv_score': 0.651556943513563,
    'binary_classification_threshold': 0.9999940391390134},
   {'all_objective_scores': OrderedDict([('Log Loss Binary',
                  1.7585681792927779),
                 ('MCC Binary', 0.4208952550769721),
                 ('AUC', 0.6896186440677965),
                 ('Precision', 0.6),
                 ('F1', 0.4615384615384615),
                 ('Balanced Accuracy Binary', 0.6705508474576272),
                 ('Accuracy Binary', 0.8955223880597015),
                 ('# Training', 133),
                 ('# Validation', 67)]),
    'mean_cv_score': 1.7585681792927779,
    'binary_classification_threshold': 0.9999940391390134},
   {'all_objective_scores': OrderedDict([('Log Loss Binary',
                  6.460743741934591),
                 ('MCC Binary', -0.12792458327635464),
                 ('AUC', 0.3523002421307506),
                 ('Precision', 0.0),
                 ('F1', 0.0),
                 ('Balanced Accuracy Binary', 0.4322033898305085),
                 ('Accuracy Binary', 0.7727272727272727),
                 ('# Training', 134),
                 ('# Validation', 66)]),
    'mean_cv_score': 6.460743741934591,
    'binary_classification_threshold': 0.9999940391390134}],
  'percent_better_than_baseline_all_objectives': {'Log Loss Binary': 25.
↪52541256225873,
   'MCC Binary': inf,
   'AUC': 11.743341404358354,
   'Precision': 53.333333333333336,
   'F1': 41.02564102564103,
   'Balanced Accuracy Binary': 13.841807909604519,
   'Accuracy Binary': -1.055329413538364},
  'percent_better_than_baseline': 25.52541256225873,
  'validation_score': 0.651556943513563},
 3: {'id': 3,
  'pipeline_name': 'Random Forest Classifier w/ Imputer + DateTime Featurization␣
↪Component + One Hot Encoder + SMOTENC Oversampler',
  'pipeline_class': evalml.pipelines.binary_classification_pipeline.
↪BinaryClassificationPipeline,
  'pipeline_summary': 'Random Forest Classifier w/ Imputer + DateTime Featurization␣
↪Component + One Hot Encoder + SMOTENC Oversampler',
  'parameters': {'Imputer': {'categorical_impute_strategy': 'most_frequent',
    'numeric_impute_strategy': 'mean',
    'categorical_fill_value': None,
    'numeric_fill_value': None},
   'DateTime Featurization Component': {'features_to_extract': ['year',
```

(continues on next page)

(continued from previous page)

```
      'month',
      'day_of_week',
      'hour'],
     'encode_as_categories': False,
     'date_index': None},
    'One Hot Encoder': {'top_n': 10,
     'features_to_encode': None,
     'categories': None,
     'drop': 'if_binary',
     'handle_unknown': 'ignore',
     'handle_missing': 'error'},
    'SMOTENC Oversampler': {'sampling_ratio': 0.25,
     'k_neighbors_default': 5,
     'n_jobs': -1,
     'sampling_ratio_dict': None,
     'categorical_features': [3,
      10,
      11,
      12,
      13,
      14,
      15,
      16,
      17,
      18,
      19,
      20,
      21,
      22,
      23,
      24,
      25,
      26,
      27,
      28,
      29,
      30,
      31,
      32,
      33,
      34,
      35,
      36,
      37,
      38,
      39,
      40,
      41,
      42,
      43,
      44,
      45,
      46,
      47,
      48,
      49,
      50,
```

(continues on next page)

(continued from previous page)

```
      51,
      52,
      53,
      54,
      55,
      56,
      57,
      58,
      59],
     'k_neighbors': 5},
    'Random Forest Classifier': {'n_estimators': 100,
     'max_depth': 6,
     'n_jobs': -1}},
   'mean_cv_score': 0.2856133806139974,
   'standard_deviation_cv_score': 0.041115539258178,
   'high_variance_cv': False,
   'training_time': 2.4224915504455566,
   'cv_data': [{'all_objective_scores': OrderedDict([('Log Loss Binary',
                 0.26369520661129536),
                ('MCC Binary', 0.8517099483190118),
                ('AUC', 0.8241525423728814),
                ('Precision', 1.0),
                ('F1', 0.8571428571428571),
                ('Balanced Accuracy Binary', 0.875),
                ('Accuracy Binary', 0.9701492537313433),
                ('# Training', 133),
                ('# Validation', 67)]),
     'mean_cv_score': 0.26369520661129536,
     'binary_classification_threshold': 0.38197006323415905},
    {'all_objective_scores': OrderedDict([('Log Loss Binary',
                 0.33304413887923806),
                ('MCC Binary', 0.3681119268158441),
                ('AUC', 0.711864406779661),
                ('Precision', 0.5),
                ('F1', 0.42857142857142855),
                ('Balanced Accuracy Binary', 0.6620762711864407),
                ('Accuracy Binary', 0.8805970149253731),
                ('# Training', 133),
                ('# Validation', 67)]),
     'mean_cv_score': 0.33304413887923806,
     'binary_classification_threshold': 0.3693991777334231},
    {'all_objective_scores': OrderedDict([('Log Loss Binary',
                 0.2601007963514588),
                ('MCC Binary', 0.831183800695515),
                ('AUC', 0.8789346246973365),
                ('Precision', 1.0),
                ('F1', 0.8333333333333333),
                ('Balanced Accuracy Binary', 0.8571428571428572),
                ('Accuracy Binary', 0.9696969696969697),
                ('# Training', 134),
                ('# Validation', 66)]),
     'mean_cv_score': 0.2601007963514588,
     'binary_classification_threshold': 0.3811434637272718}],
   'percent_better_than_baseline_all_objectives': {'Log Loss Binary': 92.
↪80647509992905,
    'MCC Binary': inf,
    'AUC': 30.498385794995965,
```

(continues on next page)

(continued from previous page)

```
    'Precision': 83.33333333333334,
    'F1': 70.63492063492062,
    'Balanced Accuracy Binary': 29.807304277643265,
    'Accuracy Binary': 5.510327152118211},
   'percent_better_than_baseline': 92.80647509992905,
   'validation_score': 0.26369520661129536},
  4: {'id': 4,
   'pipeline_name': 'LightGBM Classifier w/ Imputer + DateTime Featurization␣
↪Component + One Hot Encoder + SMOTENC Oversampler',
   'pipeline_class': evalml.pipelines.binary_classification_pipeline.
↪BinaryClassificationPipeline,
   'pipeline_summary': 'LightGBM Classifier w/ Imputer + DateTime Featurization␣
↪Component + One Hot Encoder + SMOTENC Oversampler',
   'parameters': {'Imputer': {'categorical_impute_strategy': 'most_frequent',
     'numeric_impute_strategy': 'mean',
     'categorical_fill_value': None,
     'numeric_fill_value': None},
    'DateTime Featurization Component': {'features_to_extract': ['year',
      'month',
      'day_of_week',
      'hour'],
     'encode_as_categories': False,
     'date_index': None},
    'One Hot Encoder': {'top_n': 10,
     'features_to_encode': None,
     'categories': None,
     'drop': 'if_binary',
     'handle_unknown': 'ignore',
     'handle_missing': 'error'},
    'SMOTENC Oversampler': {'sampling_ratio': 0.25,
     'k_neighbors_default': 5,
     'n_jobs': -1,
     'sampling_ratio_dict': None,
     'categorical_features': [3,
      10,
      11,
      12,
      13,
      14,
      15,
      16,
      17,
      18,
      19,
      20,
      21,
      22,
      23,
      24,
      25,
      26,
      27,
      28,
      29,
      30,
      31,
      32,
```

(continues on next page)

(continued from previous page)

```
      33,
      34,
      35,
      36,
      37,
      38,
      39,
      40,
      41,
      42,
      43,
      44,
      45,
      46,
      47,
      48,
      49,
      50,
      51,
      52,
      53,
      54,
      55,
      56,
      57,
      58,
      59],
     'k_neighbors': 5},
    'LightGBM Classifier': {'boosting_type': 'gbdt',
     'learning_rate': 0.1,
     'n_estimators': 100,
     'max_depth': 0,
     'num_leaves': 31,
     'min_child_samples': 20,
     'n_jobs': -1,
     'bagging_freq': 0,
     'bagging_fraction': 0.9}},
   'mean_cv_score': 0.30863594948686357,
   'standard_deviation_cv_score': 0.20387814963979614,
   'high_variance_cv': True,
   'training_time': 2.020923614501953,
   'cv_data': [{'all_objective_scores': OrderedDict([('Log Loss Binary',
                  0.23494740261438743),
                 ('MCC Binary', 0.6266262136411754),
                 ('AUC', 0.8940677966101696),
                 ('Precision', 0.7142857142857143),
                 ('F1', 0.6666666666666666),
                 ('Balanced Accuracy Binary', 0.7955508474576272),
                 ('Accuracy Binary', 0.9253731343283582),
                 ('# Training', 133),
                 ('# Validation', 67)]),
     'mean_cv_score': 0.23494740261438743,
     'binary_classification_threshold': 0.381970475119517},
    {'all_objective_scores': OrderedDict([('Log Loss Binary',
                  0.5391133772263208),
                 ('MCC Binary', 0.4900218379501181),
                 ('AUC', 0.635593220338983),
```

(continues on next page)

(continued from previous page)

```
                 ('Precision', 0.75),
                 ('F1', 0.5),
                 ('Balanced Accuracy Binary', 0.6790254237288136),
                 ('Accuracy Binary', 0.9104477611940298),
                 ('# Training', 133),
                 ('# Validation', 67)]),
    'mean_cv_score': 0.5391133772263208,
    'binary_classification_threshold': 0.7082083915298905},
   {'all_objective_scores': OrderedDict([('Log Loss Binary',
                 0.15184706861988254),
                 ('MCC Binary', 0.6335302236023843),
                 ('AUC', 0.8934624697336562),
                 ('Precision', 1.0),
                 ('F1', 0.6),
                 ('Balanced Accuracy Binary', 0.7142857142857143),
                 ('Accuracy Binary', 0.9393939393939394),
                 ('# Training', 134),
                 ('# Validation', 66)]),
    'mean_cv_score': 0.15184706861988254,
    'binary_classification_threshold': 0.8541079271106711}],
  'percent_better_than_baseline_all_objectives': {'Log Loss Binary': 92.
↪22662333635083,
   'MCC Binary': inf,
   'AUC': 30.77078288942697,
   'Precision': 82.14285714285715,
   'F1': 58.88888888888889,
   'Balanced Accuracy Binary': 22.962066182405174,
   'Accuracy Binary': 4.00271370420624},
  'percent_better_than_baseline': 92.22662333635083,
  'validation_score': 0.23494740261438743},
 5: {'id': 5,
  'pipeline_name': 'Logistic Regression Classifier w/ Imputer + DateTime␣
↪Featurization Component + One Hot Encoder + SMOTENC Oversampler + Standard Scaler',
  'pipeline_class': evalml.pipelines.binary_classification_pipeline.
↪BinaryClassificationPipeline,
  'pipeline_summary': 'Logistic Regression Classifier w/ Imputer + DateTime␣
↪Featurization Component + One Hot Encoder + SMOTENC Oversampler + Standard Scaler',
  'parameters': {'Imputer': {'categorical_impute_strategy': 'most_frequent',
    'numeric_impute_strategy': 'mean',
    'categorical_fill_value': None,
    'numeric_fill_value': None},
   'DateTime Featurization Component': {'features_to_extract': ['year',
     'month',
     'day_of_week',
     'hour'],
    'encode_as_categories': False,
    'date_index': None},
   'One Hot Encoder': {'top_n': 10,
    'features_to_encode': None,
    'categories': None,
    'drop': 'if_binary',
    'handle_unknown': 'ignore',
    'handle_missing': 'error'},
   'SMOTENC Oversampler': {'sampling_ratio': 0.25,
    'k_neighbors_default': 5,
    'n_jobs': -1,
    'sampling_ratio_dict': None,
```

(continues on next page)

(continued from previous page)

```
    'categorical_features': [3,
     10,
     11,
     12,
     13,
     14,
     15,
     16,
     17,
     18,
     19,
     20,
     21,
     22,
     23,
     24,
     25,
     26,
     27,
     28,
     29,
     30,
     31,
     32,
     33,
     34,
     35,
     36,
     37,
     38,
     39,
     40,
     41,
     42,
     43,
     44,
     45,
     46,
     47,
     48,
     49,
     50,
     51,
     52,
     53,
     54,
     55,
     56,
     57,
     58,
     59],
    'k_neighbors': 5},
   'Logistic Regression Classifier': {'penalty': 'l2',
    'C': 1.0,
    'n_jobs': -1,
    'multi_class': 'auto',
    'solver': 'lbfgs'}},
```

(continues on next page)

(continued from previous page)

```
   'mean_cv_score': 0.5519600330274397,
   'standard_deviation_cv_score': 0.08269486722266915,
   'high_variance_cv': False,
   'training_time': 3.4933314323425293,
   'cv_data': [{'all_objective_scores': OrderedDict([('Log Loss Binary',
                 0.6286458411705291),
                ('MCC Binary', 0.17518582316850065),
                ('AUC', 0.5826271186440678),
                ('Precision', 0.2857142857142857),
                ('F1', 0.26666666666666666),
                ('Balanced Accuracy Binary', 0.5826271186440678),
                ('Accuracy Binary', 0.835820895522388),
                ('# Training', 133),
                ('# Validation', 67)]),
     'mean_cv_score': 0.6286458411705291,
     'binary_classification_threshold': 0.381970475119517},
    {'all_objective_scores': OrderedDict([('Log Loss Binary',
                 0.46434334092355634),
                ('MCC Binary', 0.2457400868151266),
                ('AUC', 0.7033898305084746),
                ('Precision', 0.4),
                ('F1', 0.3076923076923077),
                ('Balanced Accuracy Binary', 0.5995762711864407),
                ('Accuracy Binary', 0.8656716417910447),
                ('# Training', 133),
                ('# Validation', 67)]),
     'mean_cv_score': 0.46434334092355634,
     'binary_classification_threshold': 0.6180384491178528},
    {'all_objective_scores': OrderedDict([('Log Loss Binary',
                 0.5628909169882338),
                ('MCC Binary', 0.0),
                ('AUC', 0.6077481840193704),
                ('Precision', 0.0),
                ('F1', 0.0),
                ('Balanced Accuracy Binary', 0.5),
                ('Accuracy Binary', 0.8939393939393939),
                ('# Training', 134),
                ('# Validation', 66)]),
     'mean_cv_score': 0.5628909169882338,
     'binary_classification_threshold': 0.9999940391390134}],
   'percent_better_than_baseline_all_objectives': {'Log Loss Binary': 86.
↪09820648846629,
    'MCC Binary': inf,
    'AUC': 13.125504439063763,
    'Precision': 22.857142857142858,
    'F1': 19.145299145299145,
    'Balanced Accuracy Binary': 6.073446327683618,
    'Accuracy Binary': -1.990049751243772},
   'percent_better_than_baseline': 86.09820648846629,
   'validation_score': 0.6286458411705291},
  6: {'id': 6,
   'pipeline_name': 'XGBoost Classifier w/ Imputer + DateTime Featurization Component␣
↪+ One Hot Encoder + SMOTENC Oversampler',
   'pipeline_class': evalml.pipelines.binary_classification_pipeline.
↪BinaryClassificationPipeline,
   'pipeline_summary': 'XGBoost Classifier w/ Imputer + DateTime Featurization␣
↪Component + One Hot Encoder + SMOTENC Oversampler',
```

(continues on next page)

(continued from previous page)

```
 'parameters': {'Imputer': {'categorical_impute_strategy': 'most_frequent',
   'numeric_impute_strategy': 'mean',
   'categorical_fill_value': None,
   'numeric_fill_value': None},
  'DateTime Featurization Component': {'features_to_extract': ['year',
    'month',
    'day_of_week',
    'hour'],
   'encode_as_categories': False,
   'date_index': None},
  'One Hot Encoder': {'top_n': 10,
   'features_to_encode': None,
   'categories': None,
   'drop': 'if_binary',
   'handle_unknown': 'ignore',
   'handle_missing': 'error'},
  'SMOTENC Oversampler': {'sampling_ratio': 0.25,
   'k_neighbors_default': 5,
   'n_jobs': -1,
   'sampling_ratio_dict': None,
   'categorical_features': [3,
    10,
    11,
    12,
    13,
    14,
    15,
    16,
    17,
    18,
    19,
    20,
    21,
    22,
    23,
    24,
    25,
    26,
    27,
    28,
    29,
    30,
    31,
    32,
    33,
    34,
    35,
    36,
    37,
    38,
    39,
    40,
    41,
    42,
    43,
    44,
    45,
```

(continues on next page)

(continued from previous page)

```
     46,
     47,
     48,
     49,
     50,
     51,
     52,
     53,
     54,
     55,
     56,
     57,
     58,
     59],
    'k_neighbors': 5},
   'XGBoost Classifier': {'eta': 0.1,
    'max_depth': 6,
    'min_child_weight': 1,
    'n_estimators': 100,
    'n_jobs': -1}},
  'mean_cv_score': 0.2787074593034545,
  'standard_deviation_cv_score': 0.19072502137726743,
  'high_variance_cv': True,
  'training_time': 2.0280921459198,
  'cv_data': [{'all_objective_scores': OrderedDict([('Log Loss Binary',
                0.20263786688400182),
               ('MCC Binary', 0.7712055916006633),
               ('AUC', 0.8919491525423728),
               ('Precision', 1.0),
               ('F1', 0.7692307692307693),
               ('Balanced Accuracy Binary', 0.8125),
               ('Accuracy Binary', 0.9552238805970149),
               ('# Training', 133),
               ('# Validation', 67)]),
    'mean_cv_score': 0.20263786688400182,
    'binary_classification_threshold': 0.6180384491178528},
   {'all_objective_scores': OrderedDict([('Log Loss Binary',
                0.4957285583724021),
               ('MCC Binary', 0.3681119268158441),
               ('AUC', 0.6483050847457628),
               ('Precision', 0.5),
               ('F1', 0.42857142857142855),
               ('Balanced Accuracy Binary', 0.6620762711864407),
               ('Accuracy Binary', 0.8805970149253731),
               ('# Training', 133),
               ('# Validation', 67)]),
    'mean_cv_score': 0.4957285583724021,
    'binary_classification_threshold': 0.8541079271106711},
   {'all_objective_scores': OrderedDict([('Log Loss Binary',
                0.13775595265395954),
               ('MCC Binary', 0.6335302236023843),
               ('AUC', 0.8886198547215496),
               ('Precision', 1.0),
               ('F1', 0.6),
               ('Balanced Accuracy Binary', 0.7142857142857143),
               ('Accuracy Binary', 0.9393939393939394),
               ('# Training', 134),
```

(continues on next page)

(continued from previous page)

```
                  ('# Validation', 66)]),
    'mean_cv_score': 0.13775595265395954,
    'binary_classification_threshold': 0.9098360171115123}],
  'percent_better_than_baseline_all_objectives': {'Log Loss Binary': 92.
↪98040923704308,
   'MCC Binary': inf,
   'AUC': 30.96246973365617,
   'Precision': 83.33333333333334,
   'F1': 59.92673992673993,
   'Balanced Accuracy Binary': 22.962066182405174,
   'Accuracy Binary': 4.00271370420624},
  'percent_better_than_baseline': 92.98040923704308,
  'validation_score': 0.20263786688400182},
 7: {'id': 7,
  'pipeline_name': 'Extra Trees Classifier w/ Imputer + DateTime Featurization␣
↪Component + One Hot Encoder + SMOTENC Oversampler',
  'pipeline_class': evalml.pipelines.binary_classification_pipeline.
↪BinaryClassificationPipeline,
  'pipeline_summary': 'Extra Trees Classifier w/ Imputer + DateTime Featurization␣
↪Component + One Hot Encoder + SMOTENC Oversampler',
  'parameters': {'Imputer': {'categorical_impute_strategy': 'most_frequent',
    'numeric_impute_strategy': 'mean',
    'categorical_fill_value': None,
    'numeric_fill_value': None},
   'DateTime Featurization Component': {'features_to_extract': ['year',
     'month',
     'day_of_week',
     'hour'],
    'encode_as_categories': False,
    'date_index': None},
   'One Hot Encoder': {'top_n': 10,
    'features_to_encode': None,
    'categories': None,
    'drop': 'if_binary',
    'handle_unknown': 'ignore',
    'handle_missing': 'error'},
   'SMOTENC Oversampler': {'sampling_ratio': 0.25,
    'k_neighbors_default': 5,
    'n_jobs': -1,
    'sampling_ratio_dict': None,
    'categorical_features': [3,
     10,
     11,
     12,
     13,
     14,
     15,
     16,
     17,
     18,
     19,
     20,
     21,
     22,
     23,
     24,
     25,
```

(continues on next page)

(continued from previous page)

```
      26,
      27,
      28,
      29,
      30,
      31,
      32,
      33,
      34,
      35,
      36,
      37,
      38,
      39,
      40,
      41,
      42,
      43,
      44,
      45,
      46,
      47,
      48,
      49,
      50,
      51,
      52,
      53,
      54,
      55,
      56,
      57,
      58,
      59],
    'k_neighbors': 5},
   'Extra Trees Classifier': {'n_estimators': 100,
    'max_features': 'auto',
    'max_depth': 6,
    'min_samples_split': 2,
    'min_weight_fraction_leaf': 0.0,
    'n_jobs': -1}},
  'mean_cv_score': 0.33828572013370833,
  'standard_deviation_cv_score': 0.009380537483031018,
  'high_variance_cv': False,
  'training_time': 2.2405660152435303,
  'cv_data': [{'all_objective_scores': OrderedDict([('Log Loss Binary',
                 0.32901507195041363),
                ('MCC Binary', 0.0),
                ('AUC', 0.7838983050847458),
                ('Precision', 0.0),
                ('F1', 0.0),
                ('Balanced Accuracy Binary', 0.5),
                ('Accuracy Binary', 0.8805970149253731),
                ('# Training', 133),
                ('# Validation', 67)]),
    'mean_cv_score': 0.32901507195041363,
    'binary_classification_threshold': 0.9999940391390134},
```

(continues on next page)

(continued from previous page)

```
    {'all_objective_scores': OrderedDict([('Log Loss Binary',
                  0.3380696737832299),
                 ('MCC Binary', 0.0),
                 ('AUC', 0.7139830508474576),
                 ('Precision', 0.0),
                 ('F1', 0.0),
                 ('Balanced Accuracy Binary', 0.5),
                 ('Accuracy Binary', 0.8805970149253731),
                 ('# Training', 133),
                 ('# Validation', 67)]),
     'mean_cv_score': 0.3380696737832299,
     'binary_classification_threshold': 0.9999940391390134},
    {'all_objective_scores': OrderedDict([('Log Loss Binary',
                  0.3477724146674814),
                 ('MCC Binary', 0.0),
                 ('AUC', 0.7191283292978208),
                 ('Precision', 0.0),
                 ('F1', 0.0),
                 ('Balanced Accuracy Binary', 0.5),
                 ('Accuracy Binary', 0.8939393939393939),
                 ('# Training', 134),
                 ('# Validation', 66)]),
     'mean_cv_score': 0.3477724146674814,
     'binary_classification_threshold': 0.9999940391390134}],
   'percent_better_than_baseline_all_objectives': {'Log Loss Binary': 91.
↪47985733060223,
    'MCC Binary': 0,
    'AUC': 23.900322841000797,
    'Precision': 0,
    'F1': 0,
    'Balanced Accuracy Binary': 0,
    'Accuracy Binary': 0},
   'percent_better_than_baseline': 91.47985733060223,
   'validation_score': 0.32901507195041363},
  8: {'id': 8,
   'pipeline_name': 'CatBoost Classifier w/ Imputer + DateTime Featurization␣
↪Component + SMOTENC Oversampler',
   'pipeline_class': evalml.pipelines.binary_classification_pipeline.
↪BinaryClassificationPipeline,
   'pipeline_summary': 'CatBoost Classifier w/ Imputer + DateTime Featurization␣
↪Component + SMOTENC Oversampler',
   'parameters': {'Imputer': {'categorical_impute_strategy': 'most_frequent',
     'numeric_impute_strategy': 'mean',
     'categorical_fill_value': None,
     'numeric_fill_value': None},
    'DateTime Featurization Component': {'features_to_extract': ['year',
      'month',
      'day_of_week',
      'hour'],
     'encode_as_categories': False,
     'date_index': None},
    'SMOTENC Oversampler': {'sampling_ratio': 0.25,
     'k_neighbors_default': 5,
     'n_jobs': -1,
     'sampling_ratio_dict': None,
     'categorical_features': [3, 4, 5, 6, 9, 10],
     'k_neighbors': 5},
```

(continues on next page)

(continued from previous page)

```
    'CatBoost Classifier': {'n_estimators': 10,
     'eta': 0.03,
     'max_depth': 6,
     'bootstrap_type': None,
     'silent': True,
     'allow_writing_files': False,
     'n_jobs': -1}},
   'mean_cv_score': 0.6018191346985708,
   'standard_deviation_cv_score': 0.007246095254215931,
   'high_variance_cv': False,
   'training_time': 0.9296927452087402,
   'cv_data': [{'all_objective_scores': OrderedDict([('Log Loss Binary',
                 0.5936927052183711),
                ('MCC Binary', 0.0),
                ('AUC', 0.8898305084745762),
                ('Precision', 0.11940298507462686),
                ('F1', 0.21333333333333335),
                ('Balanced Accuracy Binary', 0.5),
                ('Accuracy Binary', 0.11940298507462686),
                ('# Training', 133),
                ('# Validation', 67)]),
     'mean_cv_score': 0.5936927052183711,
     'binary_classification_threshold': 0.23607244353321918},
    {'all_objective_scores': OrderedDict([('Log Loss Binary',
                 0.6076076767585368),
                ('MCC Binary', 0.0),
                ('AUC', 0.5603813559322034),
                ('Precision', 0.11940298507462686),
                ('F1', 0.21333333333333335),
                ('Balanced Accuracy Binary', 0.5),
                ('Accuracy Binary', 0.11940298507462686),
                ('# Training', 133),
                ('# Validation', 67)]),
     'mean_cv_score': 0.6076076767585368,
     'binary_classification_threshold': 0.23607244353321918},
    {'all_objective_scores': OrderedDict([('Log Loss Binary',
                 0.6041570221188047),
                ('MCC Binary', 0.0),
                ('AUC', 0.9346246973365617),
                ('Precision', 0.10606060606060606),
                ('F1', 0.19178082191780824),
                ('Balanced Accuracy Binary', 0.5),
                ('Accuracy Binary', 0.10606060606060606),
                ('# Training', 134),
                ('# Validation', 66)]),
     'mean_cv_score': 0.6041570221188047,
     'binary_classification_threshold': 0.23607244353321918}],
   'percent_better_than_baseline_all_objectives': {'Log Loss Binary': 84.
↪84244358059615,
    'MCC Binary': 0,
    'AUC': 29.49455205811139,
    'Precision': 11.495552540328658,
    'F1': 20.614916286149164,
    'Balanced Accuracy Binary': 0,
    'Accuracy Binary': -77.00889491934268},
   'percent_better_than_baseline': 84.84244358059615,
   'validation_score': 0.5936927052183711}},
```

(continues on next page)

(continued from previous page)

```
 'search_order': [0, 1, 2, 3, 4, 5, 6, 7, 8]}
```

4.2 Objectives

4.2.1 Overview

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

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

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

4.2.2 Core Objectives

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

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

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

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

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

Binary Classification Objectives and Thresholds

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

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

X, y = load_fraud(n_rows=100)
X.ww.init(logical_types={"provider": "Categorical", "region": "Categorical"})
objective = F1()
pipeline = BinaryClassificationPipeline(component_graph=['Simple Imputer', 'DateTime␣
↪Featurization Component', 'One Hot Encoder', 'Random Forest Classifier'])
pipeline.fit(X, y)
print(pipeline.threshold)
print(pipeline.score(X, y, objectives=[objective]))

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

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

Number of training examples: 100
Targets
False    91.00%
True      9.00%
Name: fraud, dtype: object
None
OrderedDict([('F1', 1.0)])
0.23607244353321918
OrderedDict([('F1', 1.0)])
```

4.2.3 Custom Objectives

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

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

Defining a Custom Objective Function

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

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

Example: Fraud Detection

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

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


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

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

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

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

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

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

(continues on next page)

(continued from previous page)

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

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

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

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

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

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

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

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

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

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

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

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

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

(continues on next page)

(continued from previous page)

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

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

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

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

        loss_per_total_processed = loss / transaction_amount.sum()

        return loss_per_total_processed
```

4.3 Components

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

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

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

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

4.3.1 Transformers

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

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

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

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

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

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

[2]:
```
import woodwork as ww
imp = SimpleImputer(impute_strategy="mean")

X.ww.init()
X = imp.fit_transform(X)
display(X)
```

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

Below is a list of all transformers included with EvalML:

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

```
Transformer: DFS Transformer
Transformer: Delayed Feature Transformer
Transformer: Text Featurization Component
Transformer: LSA Transformer
Transformer: Drop Null Columns Transformer
Transformer: DateTime Featurization Component
Transformer: PCA Transformer
Transformer: Linear Discriminant Analysis Transformer
Transformer: Select Columns Transformer
Transformer: Drop Columns Transformer
Transformer: Undersampler
Transformer: SMOTEN Oversampler
Transformer: SMOTENC Oversampler
Transformer: SMOTE Oversampler
Transformer: Standard Scaler
Transformer: Target Imputer
Transformer: Imputer
Transformer: Per Column Imputer
Transformer: Simple Imputer
Transformer: RF Regressor Select From Model
Transformer: RF Classifier Select From Model
Transformer: Target Encoder
Transformer: One Hot Encoder
Transformer: Polynomial Detrender
```

4.3.2 Estimators

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

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

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

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

clf = LogisticRegressionClassifier()
```

(continues on next page)

(continued from previous page)

```
X = X
y = [1, 0]

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

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

Below is a list of all estimators included with EvalML:

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

```
Estimator: Stacked Ensemble Regressor
Estimator: Stacked Ensemble Classifier
Estimator: ARIMA Regressor
Estimator: SVM Regressor
Estimator: Time Series Baseline Estimator
Estimator: Decision Tree Regressor
Estimator: Baseline Regressor
Estimator: Extra Trees Regressor
Estimator: XGBoost Regressor
Estimator: CatBoost Regressor
Estimator: Random Forest Regressor
Estimator: LightGBM Regressor
Estimator: Linear Regressor
Estimator: Elastic Net Regressor
Estimator: SVM Classifier
Estimator: KNN Classifier
Estimator: Decision Tree Classifier
Estimator: LightGBM Classifier
Estimator: Baseline Classifier
Estimator: Extra Trees Classifier
Estimator: Elastic Net Classifier
Estimator: CatBoost Classifier
Estimator: XGBoost Classifier
Estimator: Random Forest Classifier
Estimator: Logistic Regression Classifier
```

4.3.3 Defining Custom Components

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

Custom Transformers

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

[6]:
```
from evalml.pipelines.components import Transformer
from evalml.utils import (
    infer_feature_types,
)

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

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

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

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

    def fit(self, X, y=None):
        """Fits DropNullColumns component to data

        Arguments:
            X (pd.DataFrame): The input training data of shape [n_samples, n_features]
            y (pd.Series, optional): The target training data of length [n_samples]

        Returns:
            self
        """
        pct_null_threshold = self.parameters["pct_null_threshold"]
        X_t = infer_feature_types(X)
        percent_null = X_t.isnull().mean()
        if pct_null_threshold == 0.0:
```

(continues on next page)

(continued from previous page)

```
            null_cols = percent_null[percent_null > 0]
        else:
            null_cols = percent_null[percent_null >= pct_null_threshold]
        self._cols_to_drop = list(null_cols.index)
        return self

    def transform(self, X, y=None):
        """Transforms data X by dropping columns that exceed the threshold of null␣
↪values.

        Arguments:
            X (pd.DataFrame): Data to transform
            y (pd.Series, optional): Ignored.

        Returns:
            pd.DataFrame: Transformed X
        """
        X_t = infer_feature_types(X)
        return X_t.drop(self._cols_to_drop)
```

Required fields

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

Required methods

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

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

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

Custom Estimators

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

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

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

(continues on next page)

(continued from previous page)

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

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

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

        Arguments:
            strategy (str): Method used to predict. Valid options are "mean", "median
↪". Defaults to "mean".
            random_seed (int): Seed for the random number generator. Defaults to 0.

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

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

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

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

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

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

(continues on next page)

(continued from previous page)

```
        Returns:
            np.ndarray (float): An array of zeroes

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

Required fields

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

Model families and problem types include:

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

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

```
Model Families:
 ['k_neighbors', 'random_forest', 'svm', 'xgboost', 'lightgbm', 'linear_model',
↪'catboost', 'extra_trees', 'ensemble', 'decision_tree', 'arima', 'baseline', 'none']
Problem Types:
 ['binary', 'multiclass', 'regression', 'time series regression', 'time series binary
↪', 'time series multiclass']
```

Required methods

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

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

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

Components Wrapping Third-Party Objects

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

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

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

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

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

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

Hyperparameter Ranges for AutoML

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

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

4.3.4 Generate Component Code

Once you have a component defined in EvalML, you can generate string Python code to recreate this component, which can then be saved and run elsewhere with EvalML. `generate_component_code` requires a component instance as the input. This method works for custom components as well, although it won't return the code required to define the custom component.

[10]:
```
from evalml.pipelines.components import LogisticRegressionClassifier
from evalml.pipelines.components.utils import generate_component_code

lr = LogisticRegressionClassifier(C=5)
code = generate_component_code(lr)
print(code)
```

```
from evalml.pipelines.components.estimators.classifiers.logistic_regression_
↪classifier import LogisticRegressionClassifier

logisticRegressionClassifier = LogisticRegressionClassifier(**{'penalty': 'l2', 'C':␣
↪5, 'n_jobs': -1, 'multi_class': 'auto', 'solver': 'lbfgs'})
```

[11]:
```
# this string can then be copy and pasted into a separate window and executed as␣
↪python code
exec(code)
```

[12]:
```
# We can also do this for custom components
from evalml.pipelines.components.utils import generate_component_code

myDropNull = DropNullColumns()
print(generate_component_code(myDropNull))
```

```
dropNullColumnsTransformer = DropNullColumns(**{'pct_null_threshold': 1.0})
```

Expectations for Custom Classification Components

EvalML expects the following from custom classification component implementations:

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

4.4 Pipelines

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

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

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

4.4.1 Defining a Pipeline Instance

Pipeline instances can be instantiated using any of the following classes:

- `RegressionPipeline`
- `BinaryClassificationPipeline`
- `MulticlassClassificationPipeline`
- `TimeSeriesRegressionPipeline`
- `TimeSeriesBinaryClassificationPipeline`
- `TimeSeriesMulticlassClassificationPipeline`

The class you want to use will depend on your problem type. The only required parameter input for instantiating a pipeline instance is `component_graph`, which is either a list or a dictionary containing a sequence of components to be fit and evaluated.

A `component_graph` list is the default representation, which represents a linear order of transforming components with an estimator as the final component. A `component_graph` dictionary is used to represent a non-linear graph of components, where the key is a unique name for each component and the value is a list with the component's class as the first element and any parents of the component as the following element(s). For either `component_graph` format, each component can be provided as a reference to the component class for custom components, and as either a string name or as a reference to the component class for components defined in EvalML.

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

component_graph_as_list = ['Imputer', 'Random Forest Classifier']
MulticlassClassificationPipeline(component_graph=component_graph_as_list)
```

[1]:
```
pipeline = MulticlassClassificationPipeline(component_graph={'Imputer': ['Imputer'],
↪'Random Forest Classifier': ['Random Forest Classifier', 'Imputer.x']}, parameters={
↪'Imputer':{'categorical_impute_strategy': 'most_frequent', 'numeric_impute_strategy
↪': 'mean', 'categorical_fill_value': None, 'numeric_fill_value': None}, 'Random␣
↪Forest Classifier':{'n_estimators': 100, 'max_depth': 6, 'n_jobs': -1}}, random_
↪seed=0)
```

[2]:
```
component_graph_as_dict = {
    'Imputer': ['Imputer'],
    'Encoder': ['One Hot Encoder', 'Imputer'],
    'Random Forest Clf': ['Random Forest Classifier', 'Encoder'],
    'Elastic Net Clf': ['Elastic Net Classifier', 'Encoder'],
    'Final Estimator': ['Logistic Regression Classifier', 'Random Forest Clf',
↪'Elastic Net Clf']
}

MulticlassClassificationPipeline(component_graph=component_graph_as_dict)
```

[2]:
```
pipeline = MulticlassClassificationPipeline(component_graph={'Imputer': ['Imputer'],
↪'Encoder': ['One Hot Encoder', 'Imputer'], 'Random Forest Clf': ['Random Forest␣
↪Classifier', 'Encoder'], 'Elastic Net Clf': ['Elastic Net Classifier', 'Encoder'],
↪'Final Estimator': ['Logistic Regression Classifier', 'Random Forest Clf', 'Elastic␣
↪Net Clf']}, parameters={'Imputer':{'categorical_impute_strategy': 'most_frequent',
↪'numeric_impute_strategy': 'mean', 'categorical_fill_value': None, 'numeric_fill_
↪value': None}, 'Encoder':{'top_n': 10, 'features_to_encode': None, 'categories':␣
↪None, 'drop': 'if_binary', 'handle_unknown': 'ignore', 'handle_missing': 'error'},
↪'Random Forest Clf':{'n_estimators': 100, 'max_depth': 6, 'n_jobs': -1}, 'Elastic␣
↪Net Clf':{'penalty': 'elasticnet', 'C': 1.0, 'l1_ratio': 0.15, 'n_jobs': -1, 'multi_
↪class': 'auto', 'solver': 'saga'}, 'Final Estimator':{'penalty': 'l2', 'C': 1.0, 'n_
↪jobs': -1, 'multi_class': 'auto', 'solver': 'lbfgs'}}, random_seed=0)
```

(continues on next page)

(continued from previous page)

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

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

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

    def __init__(self, parameter_1=1, random_seed=0):
        parameters = {"parameter_1": parameter_1}
        super().__init__(parameters=parameters,
                         random_seed=random_seed)

MulticlassClassificationPipeline([NewTransformer, 'Random Forest Classifier'])
```

[3]:
```
pipeline = MulticlassClassificationPipeline(component_graph={'New Transformer':␣
↪[NewTransformer], 'Random Forest Classifier': ['Random Forest Classifier', 'New␣
↪Transformer.x']}, parameters={'New Transformer':{'parameter_1': 1}, 'Random Forest␣
↪Classifier':{'n_estimators': 100, 'max_depth': 6, 'n_jobs': -1}}, random_seed=0)
```

4.4.2 Pipeline Usage

All pipelines define the following methods:

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

[4]:
```
from evalml.demos import load_wine
X, y = load_wine()

pipeline = MulticlassClassificationPipeline(['Imputer', 'Random Forest Classifier'])
pipeline.fit(X, y)
print(pipeline.predict(X))
print(pipeline.score(X, y, objectives=['log loss multiclass']))
```

```
         Number of Features
Numeric                  13

Number of training examples: 178
Targets
class_1    39.89%
class_0    33.15%
class_2    26.97%
Name: target, dtype: object
0      class_0
1      class_0
2      class_0
3      class_0
4      class_0
        ...
```

(continues on next page)

(continued from previous page)

```
173    class_2
174    class_2
175    class_2
176    class_2
177    class_2
Length: 178, dtype: category
Categories (3, object): ['class_0', 'class_1', 'class_2']
OrderedDict([('Log Loss Multiclass', 0.04132737017536148)])
```

4.4.3 Custom Name

By default, a pipeline's name is created using the component graph that makes up the pipeline. E.g. A pipeline with an imputer, one-hot encoder, and logistic regression classifier will have the name 'Logistic Regression Classifier w/ Imputer + One Hot Encoder'.

If you'd like to override the pipeline's name attribute, you can set the `custom_name` parameter when initalizing a pipeline, like so:

[5]:
```
component_graph = ['Imputer', 'One Hot Encoder', 'Logistic Regression Classifier']
pipeline = MulticlassClassificationPipeline(component_graph)
print("Pipeline with default name:", pipeline.name)


pipeline_with_name = MulticlassClassificationPipeline(component_graph, custom_name=
↪"My cool custom pipeline")
print("Pipeline with custom name:", pipeline_with_name.name)
```

```
Pipeline with default name: Logistic Regression Classifier w/ Imputer + One Hot␣
↪Encoder
Pipeline with custom name: My cool custom pipeline
```

4.4.4 Pipeline Parameters

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

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

[6]:
```
parameters = {
        'Imputer': {
            "categorical_impute_strategy": "most_frequent",
            "numeric_impute_strategy": "median"
        },
        'Logistic Regression Classifier': {
            'penalty': 'l2',
            'C': 1.0,
        }
    }
component_graph = ['Imputer', 'One Hot Encoder', 'Standard Scaler', 'Logistic␣
↪Regression Classifier']
MulticlassClassificationPipeline(component_graph=component_graph,␣
↪parameters=parameters)
```

[6]:
```
pipeline = MulticlassClassificationPipeline(component_graph={'Imputer': ['Imputer'],
↪'One Hot Encoder': ['One Hot Encoder', 'Imputer.x'], 'Standard Scaler': ['Standard␣
↪Scaler', 'One Hot Encoder.x'], 'Logistic Regression Classifier': ['Logistic␣
↪Regression Classifier', 'Standard Scaler.x']}, parameters={'Imputer':{'categorical_
↪impute_strategy': 'most_frequent', 'numeric_impute_strategy': 'median',
↪'categorical_fill_value': None, 'numeric_fill_value': None}, 'One Hot Encoder':{
↪'top_n': 10, 'features_to_encode': None, 'categories': None, 'drop': 'if_binary',
↪'handle_unknown': 'ignore', 'handle_missing': 'error'}, 'Logistic Regression␣
↪Classifier':{'penalty': 'l2', 'C': 1.0, 'n_jobs': -1, 'multi_class': 'auto', 'solver
↪': 'lbfgs'}}, random_seed=0)
```

4.4.5 Pipeline Description

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

[7]:
```
component_graph = ['Imputer', 'One Hot Encoder', 'Standard Scaler', 'Logistic␣
↪Regression Classifier']
pipeline = MulticlassClassificationPipeline(component_graph=component_graph,␣
↪parameters=parameters)
pipeline.graph()
```

[7]:

[8]:
```
component_graph_as_dict = {
    'Imputer': ['Imputer'],
    'Encoder': ['One Hot Encoder', 'Imputer'],
    'Random Forest Clf': ['Random Forest Classifier', 'Encoder'],
    'Elastic Net Clf': ['Elastic Net Classifier', 'Encoder'],
    'Final Estimator': ['Logistic Regression Classifier', 'Random Forest Clf',
↪'Elastic Net Clf']
}

nonlinear_pipeline = MulticlassClassificationPipeline(component_graph=component_graph_
↪as_dict)
nonlinear_pipeline.graph()
```

[8]:

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

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

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

Problem Type: multiclass
Model Family: Linear

Pipeline Steps
==============
1. Imputer
         * categorical_impute_strategy : most_frequent
         * numeric_impute_strategy : median
         * categorical_fill_value : None
         * numeric_fill_value : None
```

(continues on next page)

(continued from previous page)

```
2. One Hot Encoder
         * top_n : 10
         * features_to_encode : None
         * categories : None
         * drop : if_binary
         * handle_unknown : ignore
         * handle_missing : error
3. Standard Scaler
4. Logistic Regression Classifier
         * penalty : l2
         * C : 1.0
         * n_jobs : -1
         * multi_class : auto
         * solver : lbfgs
```

[10]:
```
nonlinear_pipeline.describe()
```

```
*****************************************************************************************************
* Logistic Regression Classifier w/ Imputer + One Hot Encoder + Random Forest␣
↪Classifier + Elastic Net Classifier *
*****************************************************************************************************

Problem Type: multiclass
Model Family: Linear

Pipeline Steps
==============
1. Imputer
         * categorical_impute_strategy : most_frequent
         * numeric_impute_strategy : mean
         * categorical_fill_value : None
         * numeric_fill_value : None
2. One Hot Encoder
         * top_n : 10
         * features_to_encode : None
         * categories : None
         * drop : if_binary
         * handle_unknown : ignore
         * handle_missing : error
3. Random Forest Classifier
         * n_estimators : 100
         * max_depth : 6
         * n_jobs : -1
4. Elastic Net Classifier
         * penalty : elasticnet
         * C : 1.0
         * l1_ratio : 0.15
         * n_jobs : -1
         * multi_class : auto
         * solver : saga
5. Logistic Regression Classifier
         * penalty : l2
         * C : 1.0
         * n_jobs : -1
         * multi_class : auto
         * solver : lbfgs
```

4.4.6 Component Graph

You can use `pipeline.get_component(name)` and provide the component name to access any component (API reference *here*):

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

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

```
[12]: nonlinear_pipeline.get_component('Elastic Net Clf')
```

```
[12]: ElasticNetClassifier(penalty='elasticnet', C=1.0, l1_ratio=0.15, n_jobs=-1, multi_
      ↪class='auto', solver='saga')
```

Alternatively, you can index directly into the pipeline to get a component

```
[13]: first_component = pipeline[0]
      print(first_component.name)
```

```
      Imputer
```

```
[14]: nonlinear_pipeline['Final Estimator']
```

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

4.4.7 Pipeline Estimator

EvalML enforces that the last component of a linear pipeline is an estimator. You can access this estimator directly by using `pipeline.estimator`.

```
[15]: pipeline.estimator
```

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

4.4.8 Input Feature Names

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

```
[16]: pipeline = MulticlassClassificationPipeline(['Imputer', 'Random Forest Classifier'])
      pipeline.fit(X, y)
      pipeline.input_feature_names
```

```
[16]: {'Imputer': ['alcohol',
        'malic_acid',
        'ash',
        'alcalinity_of_ash',
        'magnesium',
        'total_phenols',
        'flavanoids',
        'nonflavanoid_phenols',
        'proanthocyanins',
```

(continues on next page)

(continued from previous page)

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

4.4.9 Saving and Loading Pipelines

You can save and load trained or untrained pipeline instances using the Python pickle format, like so:

[17]:
```
import pickle

pipeline_to_pickle = MulticlassClassificationPipeline(['Imputer', 'Random Forest␣
↪Classifier'])

with open("pipeline.pkl", 'wb') as f:
    pickle.dump(pipeline_to_pickle, f)

pickled_pipeline = None
with open('pipeline.pkl', 'rb') as f:
    pickled_pipeline = pickle.load(f)

assert pickled_pipeline == pipeline_to_pickle
pickled_pipeline.fit(X, y)
```

[17]:
```
pipeline = MulticlassClassificationPipeline(component_graph={'Imputer': ['Imputer'],
↪'Random Forest Classifier': ['Random Forest Classifier', 'Imputer.x']}, parameters={
↪'Imputer':{'categorical_impute_strategy': 'most_frequent', 'numeric_impute_strategy
↪': 'mean', 'categorical_fill_value': None, 'numeric_fill_value': None}, 'Random␣
↪Forest Classifier':{'n_estimators': 100, 'max_depth': 6, 'n_jobs': -1}}, random_
↪seed=0)
```

4.4.10 Generate Code

Once you have instantiated a pipeline, you can generate string Python code to recreate this pipeline, which can then be saved and run elsewhere with EvalML. `generate_pipeline_code` requires a pipeline instance as the input. It can also handle custom components, but it won't return the code required to define the component. Note that any external libraries used in creating the pipeline instance will also need to be imported to execute the returned code.

Code generation is not yet supported for nonlinear pipelines.

[18]:
```
from evalml.pipelines.utils import generate_pipeline_code
from evalml.pipelines import MulticlassClassificationPipeline
import pandas as pd
from evalml.utils import infer_feature_types
from skopt.space import Integer

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

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

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

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

    def fit(self, X, y=None):
        pct_null_threshold = self.parameters["pct_null_threshold"]
        X = infer_feature_types(X)
        percent_null = X.isnull().mean()
        if pct_null_threshold == 0.0:
            null_cols = percent_null[percent_null > 0]
        else:
            null_cols = percent_null[percent_null >= pct_null_threshold]
        self._cols_to_drop = list(null_cols.index)
        return self

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

        X = infer_feature_types(X)
```

(continues on next page)

(continued from previous page)

```
        return X.drop(columns=self._cols_to_drop)


pipeline_instance = MulticlassClassificationPipeline(['Imputer', MyDropNullColumns,
                                                       'DateTime Featurization␣
↪Component',
                                                       'Text Featurization Component',
                                                       'One Hot Encoder', 'Random␣
↪Forest Classifier'],
                                                      custom_name="Pipeline with␣
↪Custom Component",
                                                      random_seed=20)

code = generate_pipeline_code(pipeline_instance)
print(code)

# This string can then be pasted into a separate window and run, although since the␣
↪pipeline has custom component `MyDropNullColumns`,
#      the code for that component must also be included
from evalml.demos import load_fraud
X, y = load_fraud(1000)
exec(code)
pipeline.fit(X, y)
```

```
from evalml.pipelines.multiclass_classification_pipeline import␣
↪MulticlassClassificationPipeline
pipeline = MulticlassClassificationPipeline(component_graph={'Imputer': ['Imputer'],
↪'My Drop Null Columns Transformer': [MyDropNullColumns, 'Imputer.x'], 'DateTime␣
↪Featurization Component': ['DateTime Featurization Component', 'My Drop Null␣
↪Columns Transformer.x'], 'Text Featurization Component': ['Text Featurization␣
↪Component', 'DateTime Featurization Component.x'], 'One Hot Encoder': ['One Hot␣
↪Encoder', 'Text Featurization Component.x'], 'Random Forest Classifier': ['Random␣
↪Forest Classifier', 'One Hot Encoder.x']}, parameters={'Imputer':{'categorical_
↪impute_strategy': 'most_frequent', 'numeric_impute_strategy': 'mean', 'categorical_
↪fill_value': None, 'numeric_fill_value': None}, 'My Drop Null Columns Transformer':{
↪'pct_null_threshold': 1.0}, 'DateTime Featurization Component':{'features_to_extract
↪': ['year', 'month', 'day_of_week', 'hour'], 'encode_as_categories': False, 'date_
↪index': None}, 'One Hot Encoder':{'top_n': 10, 'features_to_encode': None,
↪'categories': None, 'drop': 'if_binary', 'handle_unknown': 'ignore', 'handle_missing
↪': 'error'}, 'Random Forest Classifier':{'n_estimators': 100, 'max_depth': 6, 'n_
↪jobs': -1}}, custom_name='Pipeline with Custom Component', random_seed=20)
             Number of Features
Boolean                       1
Categorical                   6
Numeric                       5

Number of training examples: 1000
Targets
False    85.90%
True     14.10%
Name: fraud, dtype: object
```

[18]:
```
pipeline = MulticlassClassificationPipeline(component_graph={'Imputer': ['Imputer'],
↪'My Drop Null Columns Transformer': [MyDropNullColumns, 'Imputer.x'], 'DateTime␣
↪Featurization Component': ['DateTime Featurization Component', 'My Drop Null␣
↪Columns Transformer.x'], 'Text Featurization Component': ['Text Featurization␣
↪Component', 'DateTime Featurization Component.x'], 'One Hot Encoder': ['One Hot␣
↪Encoder', 'Text Featurization Component.x'], 'Random Forest Classifier': ['Random␣
↪Forest Classifier', 'One Hot Encoder.x']}, parameters={'Imputer':{'categorical_
↪impute_strategy': 'most_frequent', 'numeric_impute_strategy': 'mean', 'categorical_
↪fill_value': None, 'numeric_fill_value': None}, 'My Drop Null Columns Transformer':{
↪'pct_null_threshold': 1.0}, 'DateTime Featurization Component':{'features_to_extract
↪': ['year', 'month', 'day_of_week', 'hour'], 'encode_as_categories': False, 'date_
↪index': None}, 'One Hot Encoder':{'top_n': 10, 'features_to_encode': None,
↪'categories': None, 'drop': 'if_binary', 'handle_unknown': 'ignore', 'handle_missing
```

(continues on next page)

(continued from previous page)

4.5 Model Understanding

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

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

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

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

4.5.1 Graphing Utilities

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

[1]:
```
import evalml
from evalml.pipelines import BinaryClassificationPipeline
X, y = evalml.demos.load_breast_cancer()

X_train, X_holdout, y_train, y_holdout = evalml.preprocessing.split_data(X, y,␣
↪problem_type='binary',
                                                                         test_size=0.
↪2, random_seed=0)


pipeline_binary = BinaryClassificationPipeline(['Simple Imputer', 'Random Forest␣
↪Classifier'])
pipeline_binary.fit(X_train, y_train)
print(pipeline_binary.score(X_holdout, y_holdout, objectives=['log loss binary']))
```

```
         Number of Features
Numeric                  30

Number of training examples: 569
Targets
benign       62.74%
malignant    37.26%
Name: target, dtype: object
OrderedDict([('Log Loss Binary', 0.1686746297113362)])
```

Feature Importance

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

[2]:
```
pipeline_binary.feature_importance
```

[2]:

	feature	importance
0	mean concave points	0.138857
1	worst perimeter	0.137780
2	worst concave points	0.117782
3	worst radius	0.100584
4	mean concavity	0.086402
5	worst area	0.072027
6	mean perimeter	0.046500
7	worst concavity	0.043408
8	mean radius	0.037664
9	mean area	0.033683
10	radius error	0.025036
11	area error	0.019324
12	worst texture	0.014754
13	worst compactness	0.014462
14	mean texture	0.013856
15	worst smoothness	0.013710
16	worst symmetry	0.011395
17	perimeter error	0.010284
18	mean compactness	0.008162
19	mean smoothness	0.008154
20	worst fractal dimension	0.007034
21	fractal dimension error	0.005502
22	compactness error	0.004953
23	smoothness error	0.004728
24	texture error	0.004384
25	symmetry error	0.004250
26	mean fractal dimension	0.004164
27	concavity error	0.004089
28	mean symmetry	0.003997
29	concave points error	0.003076

We can also create a bar plot of the feature importances

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

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

Permutation Importance

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

[4]:
```
from evalml.model_understanding import calculate_permutation_importance
calculate_permutation_importance(pipeline_binary, X_holdout, y_holdout, 'log loss␣
↪binary')
```

[4]:

	feature	importance
0	worst perimeter	0.063657
1	worst area	0.045759

(continues on next page)

(continued from previous page)

```
2              worst radius    0.041926
3       mean concave points    0.029325
4      worst concave points    0.021045
5           worst concavity    0.010105
6             worst texture    0.010044
7              mean texture    0.006178
8             mean symmetry    0.005857
9                 mean area    0.004745
10        worst smoothness    0.003190
11              area error    0.003113
12          mean perimeter    0.002478
13   mean fractal dimension    0.001981
14       compactness error    0.001968
15         concavity error    0.001947
16           texture error    0.000291
17        smoothness error   -0.000206
18         mean smoothness   -0.000745
19  fractal dimension error   -0.000835
20       worst compactness   -0.002392
21          mean concavity   -0.003188
22        mean compactness   -0.005377
23            radius error   -0.006229
24             mean radius   -0.006870
25  worst fractal dimension   -0.007415
26          symmetry error   -0.008175
27         perimeter error   -0.008980
28    concave points error   -0.010415
29          worst symmetry   -0.018645
```

[5]:
```
from evalml.model_understanding import graph_permutation_importance
graph_permutation_importance(pipeline_binary, X_holdout, y_holdout, 'log loss binary')
```

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

Partial Dependence Plots

We can calculate the one-way partial dependence plots for a feature.

[6]:
```
from evalml.model_understanding.graphs import partial_dependence
partial_dependence(pipeline_binary, X_holdout, features='mean radius', grid_
↪resolution=5)
```

[6]:
```
   feature_values  partial_dependence class_label
0         9.69092            0.392453   malignant
1        12.40459            0.395962   malignant
2        15.11826            0.417396   malignant
3        17.83193            0.429542   malignant
4        20.54560            0.429717   malignant
```

[7]:
```
from evalml.model_understanding.graphs import graph_partial_dependence
graph_partial_dependence(pipeline_binary, X_holdout, features='mean radius', grid_
↪resolution=5)
```

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

You can also compute the partial dependence for a categorical feature. We will demonstrate this on the fraud dataset.

[8]:
```
X_fraud, y_fraud = evalml.demos.load_fraud(100, verbose=False)
X_fraud.ww.init(logical_types={"provider": "Categorical", 'region': "Categorical"})

fraud_pipeline = BinaryClassificationPipeline(["DateTime Featurization Component",
↪"One Hot Encoder", "Random Forest Classifier"])
fraud_pipeline.fit(X_fraud, y_fraud)

graph_partial_dependence(fraud_pipeline, X_fraud, features='provider')
```

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

Two-way partial dependence plots are also possible and invoke the same API.

[9]:
```
partial_dependence(pipeline_binary, X_holdout, features=('worst perimeter', 'worst␣
↪radius'), grid_resolution=5)
```

[9]:
```
              10.6876  14.404924999999999  18.12225  21.839575   25.5569  \
69.140700   0.279038            0.282898  0.435179   0.435355  0.435355
94.334275   0.304335            0.308194  0.458283   0.458458  0.458458
119.527850  0.464455            0.468314  0.612137   0.616932  0.616932
144.721425  0.483437            0.487297  0.631120   0.635915  0.635915
169.915000  0.483437            0.487297  0.631120   0.635915  0.635915

           class_label
69.140700    malignant
94.334275    malignant
119.527850   malignant
144.721425   malignant
169.915000   malignant
```

[10]:
```
graph_partial_dependence(pipeline_binary, X_holdout, features=('worst perimeter',
↪'worst radius'), grid_resolution=5)
```

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

Confusion Matrix

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

[11]:
```
from evalml.model_understanding.graphs import confusion_matrix
y_pred = pipeline_binary.predict(X_holdout)
confusion_matrix(y_holdout, y_pred)
```

[11]:
```
              benign  malignant
benign      0.930556   0.069444
malignant   0.023810   0.976190
```

[12]:
```
from evalml.model_understanding.graphs import graph_confusion_matrix
y_pred = pipeline_binary.predict(X_holdout)
graph_confusion_matrix(y_holdout, y_pred)
```

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

Precision-Recall Curve

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

[13]:
```
from evalml.model_understanding.graphs import graph_precision_recall_curve
# get the predicted probabilities associated with the "true" label
import woodwork as ww
y_encoded = y_holdout.ww.map({'benign': 0, 'malignant': 1})
y_pred_proba = pipeline_binary.predict_proba(X_holdout)["malignant"]
graph_precision_recall_curve(y_encoded, y_pred_proba)
```

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

ROC Curve

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

[14]:
```
from evalml.model_understanding.graphs import graph_roc_curve
# get the predicted probabilities associated with the "malignant" label
y_pred_proba = pipeline_binary.predict_proba(X_holdout)["malignant"]
graph_roc_curve(y_encoded, y_pred_proba)
```

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

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

[15]:
```
from evalml.pipelines import MulticlassClassificationPipeline
X_multi, y_multi = evalml.demos.load_wine()

pipeline_multi = MulticlassClassificationPipeline(['Simple Imputer', 'Random Forest␣
↪Classifier'])
pipeline_multi.fit(X_multi, y_multi)

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

```
         Number of Features
Numeric                  13

Number of training examples: 178
Targets
class_1    39.89%
class_0    33.15%
class_2    26.97%
Name: target, dtype: object
```

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

Binary Objective Score vs. Threshold Graph

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

[16]:
```
from evalml.model_understanding.graphs import binary_objective_vs_threshold
binary_objective_vs_threshold(pipeline_binary, X_holdout, y_holdout, 'f1', steps=10)
```

[16]:

	threshold	score
0	0.0	0.538462
1	0.1	0.811881
2	0.2	0.891304
3	0.3	0.901099
4	0.4	0.931818
5	0.5	0.931818
6	0.6	0.941176
7	0.7	0.951220
8	0.8	0.936709
9	0.9	0.923077
10	1.0	0.000000

[17]:
```
from evalml.model_understanding.graphs import graph_binary_objective_vs_threshold
graph_binary_objective_vs_threshold(pipeline_binary, X_holdout, y_holdout, 'f1',␣
↪steps=100)
```

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

Predicted Vs Actual Values Graph for Regression Problems

We can also create a scatterplot comparing predicted vs actual values for regression problems. We can specify an `outlier_threshold` to color values differently if the absolute difference between the actual and predicted values are outside of a given threshold.

[18]:
```
from evalml.model_understanding.graphs import graph_prediction_vs_actual
from evalml.pipelines import RegressionPipeline

X_regress, y_regress = evalml.demos.load_diabetes()
```

(continues on next page)

(continued from previous page)

```
X_train, X_test, y_train, y_test = evalml.preprocessing.split_data(X_regress, y_
↪regress, problem_type='regression')

pipeline_regress = RegressionPipeline(['One Hot Encoder', 'Linear Regressor'])
pipeline_regress.fit(X_train, y_train)

y_pred = pipeline_regress.predict(X_test)
graph_prediction_vs_actual(y_test, y_pred, outlier_threshold=50)
```

```
         Number of Features
Numeric                  10

Number of training examples: 442
Targets
200.0    1.36%
72.0     1.36%
90.0     1.13%
71.0     1.13%
178.0    1.13%
         ...
108.0    0.23%
40.0     0.23%
161.0    0.23%
45.0     0.23%
317.0    0.23%
Name: target, Length: 214, dtype: object
```

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

Now let's train a decision tree on some data.

[19]:
```
pipeline_dt = BinaryClassificationPipeline(['Simple Imputer', 'Decision Tree␣
↪Classifier'])
pipeline_dt.fit(X_train, y_train)
```

[19]:
```
pipeline = BinaryClassificationPipeline(component_graph={'Simple Imputer': ['Simple␣
↪Imputer'], 'Decision Tree Classifier': ['Decision Tree Classifier', 'Simple Imputer.
↪x']}, parameters={'Simple Imputer':{'impute_strategy': 'most_frequent', 'fill_value
↪': None}, 'Decision Tree Classifier':{'criterion': 'gini', 'max_features': 'auto',
↪'max_depth': 6, 'min_samples_split': 2, 'min_weight_fraction_leaf': 0.0}}, random_
↪seed=0)
```

Tree Visualization

We can visualize the structure of the Decision Tree that was fit to that data, and save it if necessary.

[20]:
```
from evalml.model_understanding.graphs import visualize_decision_tree

visualize_decision_tree(pipeline_dt.estimator, max_depth=2, rotate=False, filled=True,
↪ filepath=None)
```

[20]:

4.5.2 Explaining Predictions

We can explain why the model made certain predictions with the *explain_predictions* function. This will use the Shapley Additive Explanations (SHAP) algorithm to identify the top features that explain the predicted value.

This function can explain both classification and regression models - all you need to do is provide the pipeline, the input features, and a list of rows corresponding to the indices of the input features you want to explain. The function will return a table that you can print summarizing the top 3 most positive and negative contributing features to the predicted value.

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

[21]:
```
from evalml.model_understanding.prediction_explanations import explain_predictions

table = explain_predictions(pipeline=pipeline_binary, input_features=X_holdout,␣
↪y=None, indices_to_explain=[3],
                            top_k_features=6, include_shap_values=True)
print(table)
```

```
Random Forest Classifier w/ Simple Imputer

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

        1 of 1

                    Feature Name       Feature Value   Contribution to Prediction  ␣
↪SHAP Value
              ␣
↪=============================================================================
                  worst concavity         0.18                     -                 -
↪0.02
                  mean concavity          0.04                     -                 -
↪0.03
                    worst area           599.50                    -                 -
↪0.03
                   worst radius           14.04                    -                 -
↪0.05
                mean concave points       0.03                     -                 -
↪0.05
                  worst perimeter         92.80                    -                 -
↪0.06
```

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

Below is an example of how you would explain three predictions with *explain_predictions*.

[22]:
```
from evalml.model_understanding.prediction_explanations import explain_predictions

report = explain_predictions(pipeline=pipeline_binary,
                             input_features=X_holdout, y=y_holdout, indices_to_
↪explain=[0, 4, 9], include_shap_values=True,
```

(continues on next page)

(continued from previous page)

```
                              output_format='text')
print(report)
```

```
Random Forest Classifier w/ Simple Imputer

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

        1 of 3

                    Feature Name       Feature Value    Contribution to Prediction
↪SHAP Value
↪==========================================================================
                   worst perimeter         101.20                    -
↪-0.04
                 worst concave points       0.06                    -
↪-0.05
                 mean concave points        0.01                    -
↪-0.05


        2 of 3

                    Feature Name       Feature Value    Contribution to Prediction
↪SHAP Value
↪==========================================================================
                    worst radius           11.94                    -
↪0.05
                   worst perimeter         80.78                    -
↪0.06
                 mean concave points       0.02                    -
↪0.06


        3 of 3

                    Feature Name       Feature Value    Contribution to Prediction
↪SHAP Value
↪==========================================================================
                 worst concave points       0.10                    -
↪-0.05
                   worst perimeter          99.21                    -
↪-0.06
                 mean concave points        0.03                    -
↪-0.08
```

Explaining Best and Worst Predictions

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

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

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

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

[23]:
```
from evalml.model_understanding.prediction_explanations import explain_predictions_
↪best_worst

report = explain_predictions_best_worst(pipeline=pipeline_binary, input_features=X_
↪holdout, y_true=y_holdout,
                                        include_shap_values=True, top_k_features=6,␣
↪num_to_explain=2)

print(report)
```

```
Random Forest Classifier w/ Simple Imputer

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

        Best 1 of 2

                Predicted Probabilities: [benign: 1.0, malignant: 0.0]
                Predicted Value: benign
                Target Value: benign
                Cross Entropy: 0.0
                Index ID: 502

                    Feature Name       Feature Value   Contribution to Prediction␣
↪SHAP Value
↪==========================================================================
                   mean concavity          0.06                    -␣
↪-0.03
                     worst area          552.00                    -␣
↪-0.03
                worst concave points       0.08                    -␣
↪-0.05
                    worst radius          13.57                    -␣
↪-0.05
                mean concave points        0.03                    -␣
↪-0.05
                  worst perimeter          86.67                    -␣
↪-0.06


        Best 2 of 2

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

(continues on next page)

(continued from previous page)

```
                Predicted Value: benign
                Target Value: benign
                Cross Entropy: 0.0
                Index ID: 52

                    Feature Name       Feature Value    Contribution to Prediction
↪SHAP Value
↪=============
↪================================================================================
                   mean concavity          0.02                      -
↪-0.02
                    worst area           527.20                      -
↪-0.03
                    worst radius          13.10                      -
↪-0.04
                worst concave points       0.06                      -
↪-0.04
                mean concave points        0.01                      -
↪-0.05
                  worst perimeter          83.67                     -
↪-0.06


        Worst 1 of 2

                Predicted Probabilities: [benign: 0.266, malignant: 0.734]
                Predicted Value: malignant
                Target Value: benign
                Cross Entropy: 1.325
                Index ID: 363

                 Feature Name     Feature Value    Contribution to Prediction    SHAP
↪Value
↪=============
↪================================================================================
                worst perimeter      117.20                    +                    0.13
                 worst radius         18.13                    +                    0.12
                  worst area         1009.00                   +                    0.11
                   mean area         838.10                    +                    0.06
                  mean radius         16.50                    +                    0.05
                worst concavity       0.17                     -                   -0.05


        Worst 2 of 2

                Predicted Probabilities: [benign: 1.0, malignant: 0.0]
                Predicted Value: benign
                Target Value: malignant
                Cross Entropy: 7.987
                Index ID: 135

                    Feature Name       Feature Value    Contribution to Prediction
↪SHAP Value
↪=============
↪================================================================================
                   mean concavity          0.05                      -
↪-0.03
```

(continues on next page)

(continued from previous page)

```
                    worst area           653.60                  -
↪-0.04
                worst concave points       0.09                  -
↪-0.05
                    worst radius          14.49                  -
↪-0.05
                  worst perimeter         92.04                  -
↪-0.06
                mean concave points        0.03                  -
↪-0.06
```

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

```
import numpy as np

def hinge_loss(y_true, y_pred_proba):

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

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

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

print(report)
```

Changing Output Formats

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

Single prediction as a dictionary

[24]:
```
import json
single_prediction_report = explain_predictions(pipeline=pipeline_binary, input_
↪features=X_holdout, indices_to_explain=[3],
                                               y=y_holdout, top_k_features=6, include_
↪shap_values=True,
                                               output_format="dict")
print(json.dumps(single_prediction_report, indent=2))
```

```
{
  "explanations": [
    {
      "explanations": [
        {
          "feature_names": [
```

(continues on next page)

(continued from previous page)


```
            "worst concavity",
            "mean concavity",
            "worst area",
            "worst radius",
            "mean concave points",
            "worst perimeter"
          ],
          "feature_values": [
            0.1791,
            0.038,
            599.5,
            14.04,
            0.034,
            92.8
          ],
          "qualitative_explanation": [
            "-",
            "-",
            "-",
            "-",
            "-",
            "-"
          ],
          "quantitative_explanation": [
            -0.023008481104309524,
            -0.02621982146725469,
            -0.033821592020020774,
            -0.04666659740586632,
            -0.0541511910494414,
            -0.05523688273171911
          ],
          "drill_down": {},
          "class_name": "malignant",
          "expected_value": 0.3711208791208791
        }
      ]
    }
  ]
}
```


Single prediction as a dataframe

[25]:
```
single_prediction_report = explain_predictions(pipeline=pipeline_binary, input_
↪features=X_holdout,
                                               indices_to_explain=[3],
                                               y=y_holdout, top_k_features=6, include_
↪shap_values=True,
                                               output_format="dataframe")
single_prediction_report
```

[25]:
```
         feature_names  feature_values qualitative_explanation  \
0      worst concavity          0.1791                       -
1       mean concavity          0.0380                       -
2           worst area        599.5000                       -
3         worst radius         14.0400                       -
```

(continues on next page)

(continued from previous page)

```
4  mean concave points          0.0340                          -
5      worst perimeter         92.8000                          -

   quantitative_explanation class_name  prediction_number
0                 -0.023008  malignant                  0
1                 -0.026220  malignant                  0
2                 -0.033822  malignant                  0
3                 -0.046667  malignant                  0
4                 -0.054151  malignant                  0
5                 -0.055237  malignant                  0
```

Best and worst predictions as a dictionary

[26]:
```
report = explain_predictions_best_worst(pipeline=pipeline_binary, input_features=X, y_
↪true=y,
                                        num_to_explain=1, top_k_features=6,
                                        include_shap_values=True, output_format="dict
↪")
print(json.dumps(report, indent=2))
```

```
{
  "explanations": [
    {
      "rank": {
        "prefix": "best",
        "index": 1
      },
      "predicted_values": {
        "probabilities": {
          "benign": 1.0,
          "malignant": 0.0
        },
        "predicted_value": "benign",
        "target_value": "benign",
        "error_name": "Cross Entropy",
        "error_value": 0.0001970443507070075,
        "index_id": 475
      },
      "explanations": [
        {
          "feature_names": [
            "mean concavity",
            "worst area",
            "worst radius",
            "worst concave points",
            "worst perimeter",
            "mean concave points"
          ],
          "feature_values": [
            0.05835,
            605.8,
            14.09,
            0.09783,
            93.22,
            0.03078
```

(continues on next page)

(continued from previous page)


```
        ],
        "qualitative_explanation": [
          "-",
          "-",
          "-",
          "-",
          "-",
          "-"
        ],
        "quantitative_explanation": [
          -0.028481050954786636,
          -0.03050522196002462,
          -0.042922079201003216,
          -0.04429366151003684,
          -0.05486784013962313,
          -0.05639460900233733
        ],
        "drill_down": {},
        "class_name": "malignant",
        "expected_value": 0.3711208791208791
      }
    ]
  },
  {
    "rank": {
      "prefix": "worst",
      "index": 1
    },
    "predicted_values": {
      "probabilities": {
        "benign": 1.0,
        "malignant": 0.0
      },
      "predicted_value": "benign",
      "target_value": "malignant",
      "error_name": "Cross Entropy",
      "error_value": 7.986911819330411,
      "index_id": 135
    },
    "explanations": [
      {
        "feature_names": [
          "mean concavity",
          "worst area",
          "worst concave points",
          "worst radius",
          "worst perimeter",
          "mean concave points"
        ],
        "feature_values": [
          0.04711,
          653.6,
          0.09331,
          14.49,
          92.04,
          0.02704
        ],
```


(continues on next page)

(continued from previous page)


```
          "qualitative_explanation": [
            "-",
            "-",
            "-",
            "-",
            "-",
            "-"
          ],
          "quantitative_explanation": [
            -0.029936744551331215,
            -0.03748357654576422,
            -0.04553126236476177,
            -0.0483274199182721,
            -0.06039220265366764,
            -0.060441902449258976
          ],
          "drill_down": {},
          "class_name": "malignant",
          "expected_value": 0.3711208791208791
        }
      ]
    }
  ]
}
```


Best and worst predictions as a dataframe

[27]:
```
report = explain_predictions_best_worst(pipeline=pipeline_binary, input_features=X_
↪holdout, y_true=y_holdout,
                                        num_to_explain=1, top_k_features=6,
                                        include_shap_values=True, output_format=
↪"dataframe")
report
```

[27]:

	feature_names	feature_values	qualitative_explanation	\
0	mean concavity	0.05928	-	
1	worst area	552.00000	-	
2	worst concave points	0.08411	-	
3	worst radius	13.57000	-	
4	mean concave points	0.03279	-	
5	worst perimeter	86.67000	-	
6	mean concavity	0.04711	-	
7	worst area	653.60000	-	
8	worst concave points	0.09331	-	
9	worst radius	14.49000	-	
10	worst perimeter	92.04000	-	
11	mean concave points	0.02704	-	

	quantitative_explanation	class_name	label_benign_probability	\
0	-0.029022	malignant	1.0	
1	-0.034112	malignant	1.0	
2	-0.046896	malignant	1.0	
3	-0.046928	malignant	1.0	
4	-0.052902	malignant	1.0	
5	-0.064320	malignant	1.0	

(continues on next page)

(continued from previous page)

```
6                  -0.029937  malignant                       1.0
7                  -0.037484  malignant                       1.0
8                  -0.045531  malignant                       1.0
9                  -0.048327  malignant                       1.0
10                 -0.060392  malignant                       1.0
11                 -0.060442  malignant                       1.0

    label_malignant_probability predicted_value target_value     error_name  \
0                           0.0          benign       benign  Cross Entropy
1                           0.0          benign       benign  Cross Entropy
2                           0.0          benign       benign  Cross Entropy
3                           0.0          benign       benign  Cross Entropy
4                           0.0          benign       benign  Cross Entropy
5                           0.0          benign       benign  Cross Entropy
6                           0.0          benign    malignant  Cross Entropy
7                           0.0          benign    malignant  Cross Entropy
8                           0.0          benign    malignant  Cross Entropy
9                           0.0          benign    malignant  Cross Entropy
10                          0.0          benign    malignant  Cross Entropy
11                          0.0          benign    malignant  Cross Entropy

    error_value  index_id  rank prefix
0      0.000197       502     1   best
1      0.000197       502     1   best
2      0.000197       502     1   best
3      0.000197       502     1   best
4      0.000197       502     1   best
5      0.000197       502     1   best
6      7.986912       135     1  worst
7      7.986912       135     1  worst
8      7.986912       135     1  worst
9      7.986912       135     1  worst
10     7.986912       135     1  worst
11     7.986912       135     1  worst
```

Force Plots

Force plots can be generated to predict single or multiple rows for binary, multiclass and regression problem types. Here's an example of predicting a single row on a binary classification dataset. The force plots show the predictive power of each of the features in making the negative ("Class: 0") prediction and the positive ("Class: 1") prediction.

[28]:
```
import shap

from evalml.model_understanding.force_plots import graph_force_plot

rows_to_explain = [0] # Should be a list of integer indices of the rows to explain.

results = graph_force_plot(pipeline_binary, rows_to_explain=rows_to_explain,
                           training_data=X_holdout, y=y_holdout)

for result in results:
    for cls in result:
        print("Class:", cls)
        display(result[cls]["plot"])
```

```
<IPython.core.display.HTML object>
```

```
Class: malignant
```

```
<shap.plots._force.AdditiveForceVisualizer at 0x7f5b1473fd30>
```

Here's an example of a force plot explaining multiple predictions on a multiclass problem. These plots show the force plots for each row arranged as consecutive columns that can be ordered by the dropdown above. Clicking the column indicates which row explanation is underneath.

[29]:
```
rows_to_explain = [0,1,2,3,4] # Should be a list of integer indices of the rows to␣
↪explain.

results = graph_force_plot(pipeline_multi,
                           rows_to_explain=rows_to_explain,
                           training_data=X_multi, y=y_multi)

for idx, result in enumerate(results):
    print("Row:", idx)
    for cls in result:
        print("Class:", cls)
        display(result[cls]["plot"])
```

```
<IPython.core.display.HTML object>
```

```
Row: 0
Class: class_0
```

```
<shap.plots._force.AdditiveForceVisualizer at 0x7f5b3f5bdf70>
```

```
Class: class_1
```

```
<shap.plots._force.AdditiveForceVisualizer at 0x7f5b3f5ca0d0>
```

```
Class: class_2
```

```
<shap.plots._force.AdditiveForceVisualizer at 0x7f5b3f5cadc0>
```

```
Row: 1
Class: class_0
```

```
<shap.plots._force.AdditiveForceVisualizer at 0x7f5b14ca0460>
```

```
Class: class_1
```

```
<shap.plots._force.AdditiveForceVisualizer at 0x7f5b3f5ca4c0>
```

```
Class: class_2
```

```
<shap.plots._force.AdditiveForceVisualizer at 0x7f5b3f5c9220>
```

```
Row: 2
Class: class_0
```

```
<shap.plots._force.AdditiveForceVisualizer at 0x7f5b3f5c92e0>
```

```
Class: class_1
```

```
<shap.plots._force.AdditiveForceVisualizer at 0x7f5b3f5c9ac0>
```

```
Class: class_2
```

```
<shap.plots._force.AdditiveForceVisualizer at 0x7f5b3f5c9f10>
```

```
Row: 3
Class: class_0
```

```
<shap.plots._force.AdditiveForceVisualizer at 0x7f5b3f5c9df0>
Class: class_1
<shap.plots._force.AdditiveForceVisualizer at 0x7f5b3f5c9760>
Class: class_2
<shap.plots._force.AdditiveForceVisualizer at 0x7f5b3f5c95b0>
Row: 4
Class: class_0
<shap.plots._force.AdditiveForceVisualizer at 0x7f5b3f5c97f0>
Class: class_1
<shap.plots._force.AdditiveForceVisualizer at 0x7f5b3f5c9cd0>
Class: class_2
<shap.plots._force.AdditiveForceVisualizer at 0x7f5b3f5c9490>
```

4.6 Data Checks

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

4.6.1 Missing Data

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

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

from evalml.data_checks import HighlyNullDataCheck

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

null_check = HighlyNullDataCheck(pct_null_threshold=0.8)
results = null_check.validate(X)

for message in results['warnings']:
    print("Warning:", message['message'])
```

(continues on next page)

(continued from previous page)

```
for message in results['errors']:
    print("Error:", message['message'])
```

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

4.6.2 Abnormal Data

EvalML provides a few data checks to check for abnormal data:

- NoVarianceDataCheck
- ClassImbalanceDataCheck
- TargetLeakageDataCheck
- InvalidTargetDataCheck
- IDColumnsDataCheck
- OutliersDataCheck
- HighVarianceCVDataCheck
- MulticollinearityDataCheck
- UniquenessDataCheck

Zero Variance

Data with zero variance indicates that all values are identical. If a feature has zero variance, it is not likely to be a useful feature. Similarly, if the target has zero variance, there is likely something wrong. NoVarianceDataCheck checks if the target or any feature has only one unique value and alerts you to any such columns.

[2]:
```
from evalml.data_checks import NoVarianceDataCheck
X = pd.DataFrame({"no var col": [0, 0, 0],
                  "good col":[0, 4, 1]})
y = pd.Series([1, 0, 1])
no_variance_data_check = NoVarianceDataCheck()
results = no_variance_data_check.validate(X, y)

for message in results['warnings']:
    print("Warning:", message['message'])

for message in results['errors']:
    print("Error:", message['message'])
```

```
Error: no var col has 1 unique value.
```

Note that you can set NaN to count as an unique value, but NoVarianceDataCheck will still return a warning if there is only one unique non-NaN value in a given column.

[3]:
```
from evalml.data_checks import NoVarianceDataCheck

X = pd.DataFrame({"no var col": [0, 0, 0],
                  "no var col with nan": [1, np.nan, 1],
                  "good col":[0, 4, 1]})
y = pd.Series([1, 0, 1])
```

(continues on next page)

(continued from previous page)

```
no_variance_data_check = NoVarianceDataCheck(count_nan_as_value=True)
results = no_variance_data_check.validate(X, y)

for message in results['warnings']:
    print("Warning:", message['message'])

for message in results['errors']:
    print("Error:", message['message'])
```

```
Warning: no var col with nan has two unique values including nulls. Consider encoding␣
↪the nulls for this column to be useful for machine learning.
Error: no var col has 1 unique value.
```

Class Imbalance

For classification problems, the distribution of examples across each class can vary. For small variations, this is normal and expected. However, when the number of examples for each class label is disproportionately biased or skewed towards a particular class (or classes), it can be difficult for machine learning models to predict well. In addition, having a low number of examples for a given class could mean that one or more of the CV folds generated for the training data could only have few or no examples from that class. This may cause the model to only predict the majority class and ultimately resulting in a poor-performant model.

`ClassImbalanceDataCheck` checks if the target labels are imbalanced beyond a specified threshold for a certain number of CV folds. It returns `DataCheckError` messages for any classes that have less samples than double the number of CV folds specified (since that indicates the likelihood of having at little to no samples of that class in a given fold), and `DataCheckWarning` messages for any classes that fall below the set threshold percentage.

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

X = pd.DataFrame([[1, 2, 0, 1],
                  [4, 1, 9, 0],
                  [4, 4, 8, 3],
                  [9, 2, 7, 1]])
y = pd.Series([0, 1, 1, 1, 1])

class_imbalance_check = ClassImbalanceDataCheck(threshold=0.25, num_cv_folds=4)
results = class_imbalance_check.validate(X, y)

for message in results['warnings']:
    print("Warning:", message['message'])

for message in results['errors']:
    print("Error:", message['message'])
```

```
Warning: The following labels fall below 25% of the target: [0]
Warning: The following labels in the target have severe class imbalance because they␣
↪fall under 25% of the target and have less than 100 samples: [0]
Error: The number of instances of these targets is less than 2 * the number of cross␣
↪folds = 8 instances: [1, 0]
```

Target Leakage

Target leakage, also known as data leakage, can occur when you train your model on a dataset that includes information that should not be available at the time of prediction. This causes the model to score suspiciously well, but perform poorly in production. `TargetLeakageDataCheck` checks for features that could potentially be "leaking" information by calculating the Pearson correlation coefficient between each feature and the target to warn users if there are features are highly correlated with the target. Currently, only numerical features are considered.

[5]:
```
from evalml.data_checks import TargetLeakageDataCheck
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])

target_leakage_check = TargetLeakageDataCheck(pct_corr_threshold=0.8)
results = target_leakage_check.validate(X, y)

for message in results['warnings']:
    print("Warning:", message['message'])

for message in results['errors']:
    print("Error:", message['message'])
```

```
Warning: Column 'leak' is 80.0% or more correlated with the target
Warning: Column 'x' is 80.0% or more correlated with the target
Warning: Column 'y' is 80.0% or more correlated with the target
```

Invalid Target Data

The `InvalidTargetDataCheck` checks if the target data contains any missing or invalid values. Specifically:

- if any of the target values are missing, a `DataCheckError` message is returned
- if the specified problem type is a binary classification problem but there is more or less than two unique values in the target, a `DataCheckError` message is returned
- if binary classification target classes are numeric values not equal to {0, 1}, a `DataCheckError` message is returned because it can cause unpredictable behavior when passed to pipelines

[6]:
```
from evalml.data_checks import InvalidTargetDataCheck

X = pd.DataFrame({})
y = pd.Series([0, 1, None, None])

invalid_target_check = InvalidTargetDataCheck('binary', 'Log Loss Binary')
results = invalid_target_check.validate(X, y)

for message in results['warnings']:
    print("Warning:", message['message'])

for message in results['errors']:
    print("Error:", message['message'])
```

```
Warning: Input target and features have different lengths
Warning: Input target and features have mismatched indices
Error: 2 row(s) (50.0%) of target values are null
```

ID Columns

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

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

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

id_col_check = IDColumnsDataCheck(id_threshold=0.9)
results = id_col_check.validate(X, y)

for message in results['warnings']:
    print("Warning:", message['message'])

for message in results['errors']:
    print("Error:", message['message'])
```

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

Multicollinearity Data Check

The `MulticollinearityDataCheck` data check is used in to detect if are any set of features that are likely to be multicollinear. Multicollinear features affect the performance of a model, but more importantly, it may greatly impact model interpretation. EvalML uses mutual information to determine collinearity.

[8]:
```
from evalml.data_checks import MulticollinearityDataCheck

y = pd.Series([1, 0, 2, 3, 4])
X = pd.DataFrame({'col_1': y,
                  'col_2': y * 3,
                  'col_3': ~y,
                  'col_4': y / 2,
                  'col_5': y + 1,
                  'not_collinear': [0, 1, 0, 0, 0]})

multi_check = MulticollinearityDataCheck(threshold=0.95)
results = multi_check.validate(X, y)

for message in results['warnings']:
    print("Warning:", message['message'])

for message in results['errors']:
    print("Error:", message['message'])
```

```
Warning: Columns are likely to be correlated: [('col_1', 'col_2'), ('col_1', 'col_3'),
↪ ('col_1', 'col_4'), ('col_1', 'col_5'), ('col_2', 'col_3'), ('col_2', 'col_4'), (
↪'col_2', 'col_5'), ('col_3', 'col_4'), ('col_3', 'col_5'), ('col_4', 'col_5')]
```

Uniqueness Data Check

The `UniquenessDataCheck` is used to detect columns with either too unique or not unique enough values. For regression type problems, the data is checked for a lower limit of uniqueness. For multiclass type problems, the data is checked for an upper limit.

[9]:
```
import pandas as pd
from evalml.data_checks import UniquenessDataCheck

X = pd.DataFrame({'most_unique': [float(x) for x in range(10)],  # [0,1,2,3,4,5,6,7,8,
↪9]
                  'more_unique': [x % 5 for x in range(10)],  # [0,1,2,3,4,0,1,2,3,4]
                  'unique': [x % 3 for x in range(10)],  # [0,1,2,0,1,2,0,1,2,0]
                  'less_unique': [x % 2 for x in range(10)],  # [0,1,0,1,0,1,0,1,0,1]
                  'not_unique': [float(1) for x in range(10)]})  # [1,1,1,1,1,1,1,1,1,
↪1]

uniqueness_check = UniquenessDataCheck(problem_type="regression",
                                       threshold=.5)
results = uniqueness_check.validate(X, y=None)

for message in results['warnings']:
    print("Warning:", message['message'])

for message in results['errors']:
    print("Error:", message['message'])
```

```
Warning: Input columns (not_unique) for regression problem type are not unique enough.
```

Sparsity Data Check

The `SparsityDataCheck` is used to identify features that contain a sparsity of values.

[10]:
```
from evalml.data_checks import SparsityDataCheck

X = pd.DataFrame({'most_sparse': [float(x) for x in range(10)],  # [0,1,2,3,4,5,6,7,8,
↪9]
                  'more_sparse': [x % 5 for x in range(10)],     # [0,1,2,3,4,0,1,2,3,
↪4]
                  'sparse': [x % 3 for x in range(10)],          # [0,1,2,0,1,2,0,1,2,
↪0]
                  'less_sparse': [x % 2 for x in range(10)],     # [0,1,0,1,0,1,0,1,0,
↪1]
                  'not_sparse': [float(1) for x in range(10)]})  # [1,1,1,1,1,1,1,1,1,
↪1]


sparsity_check = SparsityDataCheck(problem_type="multiclass",
                                   threshold=.4,
                                   unique_count_threshold=3)
results = sparsity_check.validate(X,y=None)

for message in results['warnings']:
    print("Warning:", message['message'])

for message in results['errors']:
    print("Error:", message['message'])
```

```
Warning: Input columns (most_sparse) for multiclass problem type are too sparse.
Warning: Input columns (more_sparse) for multiclass problem type are too sparse.
Warning: Input columns (sparse) for multiclass problem type are too sparse.
```

4.6.3 Outliers

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

Below we generate a random dataset with some outliers.

[11]:
```
data = np.tile(np.arange(10) * 0.01, (100, 10))
X = pd.DataFrame(data=data)

# generate some outliers in columns 3, 25, 55, and 72
X.iloc[0, 3] = -10000
X.iloc[3, 25] = 10000
X.iloc[5, 55] = 10000
X.iloc[10, 72] = -10000
```

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

[12]:
```
from evalml.data_checks import OutliersDataCheck

outliers_check = OutliersDataCheck()
results = outliers_check.validate(X, y)

for message in results['warnings']:
    print("Warning:", message['message'])

for message in results['errors']:
    print("Error:", message['message'])
```

```
Warning: Column(s) '3', '25', '55', '72' are likely to have outlier data.
```

4.6.4 Data Check Messages

Each data check's `validate` method returns a list of `DataCheckMessage` objects indicating warnings or errors found; warnings are stored as a `DataCheckWarning` object (*API reference*) and errors are stored as a `DataCheckError` object (*API reference*). You can filter the messages returned by a data check by checking for the type of message returned. Below, `NoVarianceDataCheck` returns a list containing a `DataCheckWarning` and a `DataCheckError` message. We can determine which is which by checking the type of each message.

[13]:
```
from evalml.data_checks import NoVarianceDataCheck, DataCheckError, DataCheckWarning

X = pd.DataFrame({"no var col": [0, 0, 0],
                  "no var col with nan": [1, np.nan, 1],
                  "good col":[0, 4, 1]})
y = pd.Series([1, 0, 1])

no_variance_data_check = NoVarianceDataCheck(count_nan_as_value=True)
results = no_variance_data_check.validate(X, y)
```

(continues on next page)

(continued from previous page)

```
for message in results['warnings']:
    print("Warning:", message['message'])

for message in results['errors']:
    print("Error:", message['message'])
```

```
Warning: no var col with nan has two unique values including nulls. Consider encoding␣
↪the nulls for this column to be useful for machine learning.
Error: no var col has 1 unique value.
```

4.6.5 Writing Your Own Data Check

If you would prefer to write your own data check, you can do so by extending the `DataCheck` class and implementing the `validate(self, X, y)` class method. Below, we've created a new `DataCheck`, `ZeroVarianceDataCheck`, which is similar to `NoVarianceDataCheck` defined in `EvalML`. The `validate(self, X, y)` method should return a dictionary with 'warnings' and 'errors' as keys mapping to list of warnings and errors, respectively.

[14]:
```
from evalml.data_checks import DataCheck

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

4.6.6 Defining Collections of Data Checks

For convenience, EvalML provides a `DataChecks` class to represent a collection of data checks. We will go over `DefaultDataChecks` (*API reference*), a collection defined to check for some of the most common data issues.

Default Data Checks

`DefaultDataChecks` is a collection of data checks defined to check for some of the most common data issues. They include:

- `HighlyNullDataCheck`
- `IDColumnsDataCheck`
- `TargetLeakageDataCheck`
- `InvalidTargetDataCheck`
- `ClassImbalanceDataCheck` (for classification problem types)
- `NoVarianceDataCheck`
- `DateTimeNaNDataCheck`
- `NaturalLanguageNaNDataCheck`

4.6.7 Writing Your Own Collection of Data Checks

If you would prefer to create your own collection of data checks, you could either write your own data checks class by extending the `DataChecks` class and setting the `self.data_checks` attribute to the list of `DataCheck` classes or objects, or you could pass that list of data checks to the constructor of the `DataChecks` class. Below, we create two identical collections of data checks using the two different methods.

[15]:
```
# Create a subclass of `DataChecks`
from evalml.data_checks import DataChecks, HighlyNullDataCheck,␣
↪InvalidTargetDataCheck, NoVarianceDataCheck, ClassImbalanceDataCheck,␣
↪TargetLeakageDataCheck
from evalml.problem_types import ProblemTypes, handle_problem_types

class MyCustomDataChecks(DataChecks):

    data_checks = [HighlyNullDataCheck, InvalidTargetDataCheck, NoVarianceDataCheck,␣
↪TargetLeakageDataCheck]

    def __init__(self, problem_type, objective):
        """
        A collection of basic data checks.
        Arguments:
            problem_type (str): The problem type that is being validated. Can be␣
↪regression, binary, or multiclass.
        """
        if handle_problem_types(problem_type) == ProblemTypes.REGRESSION:
            super().__init__(self.data_checks,
                             data_check_params={"InvalidTargetDataCheck": {"problem_
↪type": problem_type,
                                                                           "objective
↪": objective}})
        else:
            super().__init__(self.data_checks + [ClassImbalanceDataCheck],
                             data_check_params={"InvalidTargetDataCheck": {"problem_
↪type": problem_type,
                                                                           "objective
↪": objective}})


custom_data_checks = MyCustomDataChecks(problem_type=ProblemTypes.REGRESSION,␣
↪objective="R2")
for data_check in custom_data_checks.data_checks:
    print(data_check.name)
```

```
HighlyNullDataCheck
InvalidTargetDataCheck
NoVarianceDataCheck
TargetLeakageDataCheck
```

[16]:
```
# Pass list of data checks to the `data_checks` parameter of DataChecks
same_custom_data_checks = DataChecks(data_checks=[HighlyNullDataCheck,␣
↪InvalidTargetDataCheck, NoVarianceDataCheck, TargetLeakageDataCheck],
                                     data_check_params={"InvalidTargetDataCheck": {
↪"problem_type": ProblemTypes.REGRESSION,

↪"objective": "R2"}})
for data_check in custom_data_checks.data_checks:
    print(data_check.name)
```

```
HighlyNullDataCheck
InvalidTargetDataCheck
NoVarianceDataCheck
TargetLeakageDataCheck
```

4.7 Utilities

4.7.1 Configuring Logging

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

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

4.7.2 System Information

EvalML provides a command-line interface (CLI) tool prints the version of EvalML and core dependencies installed, as well as some basic system information. To use this tool, just run `evalml info` in your shell or terminal. This could be useful for debugging purposes or tracking down any version-related issues.

[1]:
```
!evalml info
```

```
EvalML version: 0.28.0
EvalML installation directory: /home/docs/checkouts/readthedocs.org/user_builds/
↪feature-labs-inc-evalml/envs/v0.28.0/lib/python3.8/site-packages/evalml

SYSTEM INFO
-----------
python: 3.8.6.final.0
python-bits: 64
OS: Linux
OS-release: 5.4.0-1035-aws
machine: x86_64
processor: x86_64
byteorder: little
LC_ALL: None
LANG: C.UTF-8
LOCALE: en_US.UTF-8
# of CPUS: 2
Available memory: 6.2G

INSTALLED VERSIONS
------------------
zict: 2.0.0
xgboost: 1.2.1
woodwork: 0.4.2
widgetsnbextension: 3.5.1
wheel: 0.36.2
webencodings: 0.5.1
wcwidth: 0.2.5
```

(continues on next page)

(continued from previous page)

```
urllib3: 1.26.6
traitlets: 5.0.5
tqdm: 4.61.1
tornado: 6.1
toolz: 0.11.1
threadpoolctl: 2.1.0
texttable: 1.6.3
testpath: 0.5.0
terminado: 0.10.1
tenacity: 7.0.0
tblib: 1.7.0
statsmodels: 0.12.2
sphinxcontrib-websupport: 1.2.4
sphinxcontrib-serializinghtml: 1.1.5
sphinxcontrib-qthelp: 1.0.3
sphinxcontrib-jsmath: 1.0.1
sphinxcontrib-htmlhelp: 2.0.0
sphinxcontrib-devhelp: 1.0.2
sphinxcontrib-applehelp: 1.0.2
sphinx: 3.5.4
sphinx-rtd-theme: 0.4.3
soupsieve: 2.2.1
sortedcontainers: 2.4.0
snowballstemmer: 2.1.0
slicer: 0.0.7
sktime: 0.6.1
six: 1.16.0
shap: 0.39.0
setuptools: 57.0.0
send2trash: 1.7.1
seaborn: 0.11.1
scipy: 1.7.0
scikit-optimize: 0.8.1
scikit-learn: 0.24.2
requirements-parser: 0.2.0
requests: 2.25.1
regex: 2021.7.1
recommonmark: 0.5.0
readthedocs-sphinx-ext: 2.1.4
pyzmq: 22.1.0
pyyaml: 5.4.1
pytz: 2021.1
python-dateutil: 2.8.1
pyrsistent: 0.18.0
pyparsing: 2.4.7
pygments: 2.9.0
pydata-sphinx-theme: 0.6.3
pycparser: 2.20
pyaml: 20.4.0
ptyprocess: 0.7.0
psutil: 5.8.0
prompt-toolkit: 3.0.19
prometheus-client: 0.11.0
pmdarima: 1.8.0
plotly: 5.1.0
pip: 21.1.3
pillow: 8.3.0
```

(continues on next page)

(continued from previous page)

```
pickleshare: 0.7.5
pexpect: 4.8.0
patsy: 0.5.1
partd: 1.2.0
parso: 0.8.2
pandocfilters: 1.4.3
pandas: 1.2.4
packaging: 20.9
numpy: 1.21.0
numba: 0.53.1
notebook: 6.4.0
nltk: 3.6.2
nlp-primitives: 1.1.0
networkx: 2.5.1
nest-asyncio: 1.5.1
nbsphinx: 0.8.6
nbformat: 5.1.3
nbconvert: 6.1.0
nbclient: 0.5.3
msgpack: 1.0.2
mock: 1.0.1
mistune: 0.8.4
matplotlib: 3.4.2
matplotlib-inline: 0.1.2
markupsafe: 2.0.1
locket: 0.2.1
llvmlite: 0.36.0
lightgbm: 3.2.1
kiwisolver: 1.3.1
kaleido: 0.2.1
jupyterlab-widgets: 1.0.0
jupyterlab-pygments: 0.1.2
jupyter-core: 4.7.1
jupyter-client: 6.1.12
jsonschema: 3.2.0
joblib: 1.0.1
jinja2: 3.0.1
jedi: 0.18.0
ipywidgets: 7.6.3
ipython: 7.25.0
ipython-genutils: 0.2.0
ipykernel: 6.0.1
imbalanced-learn: 0.8.0
imagesize: 1.2.0
idna: 2.10
heapdict: 1.0.1
graphviz: 0.16
future: 0.18.2
fsspec: 2021.6.1
featuretools: 0.25.0
evalml: 0.28.0
entrypoints: 0.3
docutils: 0.16
distributed: 2021.6.2
defusedxml: 0.7.1
decorator: 4.4.2
debugpy: 1.3.0
```

(continues on next page)

(continued from previous page)

```
dask: 2021.6.2
cython: 0.29.17
cycler: 0.10.0
commonmark: 0.8.1
colorama: 0.4.4
cloudpickle: 1.6.0
click: 8.0.1
chardet: 4.0.0
cffi: 1.14.5
certifi: 2021.5.30
category-encoders: 2.2.2
catboost: 0.26
bleach: 3.3.0
beautifulsoup4: 4.9.3
backcall: 0.2.0
babel: 2.9.1
attrs: 21.2.0
async-generator: 1.10
argon2-cffi: 20.1.0
alabaster: 0.7.12
```

4.8 FAQ

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

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

4.8.2 Q: How does EvalML handle missing values?

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

4.8.3 Q: How does EvalML handle categorical encoding?

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

4.8.4 Q: How does EvalML handle feature selection?

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

4.8.5 Q: How is feature importance calculated?

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

4.8.6 Q: How does hyperparameter tuning work?

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

4.8.7 Q: Can I create my own objective metric?

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

4.8.8 Q: How does EvalML avoid overfitting?

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

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

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

4.8.10 Q: Does EvalML work with X algorithm?

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

CHAPTER

FIVE

API REFERENCE

5.1 Demo Datasets

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

5.1.1 evalml.demos.load_fraud

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

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

Parameters

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

Returns X and y

Return type (pd.Dataframe, pd.Series)

5.1.2 evalml.demos.load_wine

evalml.demos.**load_wine**()

Load wine dataset. Multiclass problem.

Returns X and y

Return type (pd.Dataframe, pd.Series)

5.1.3 evalml.demos.load_breast_cancer

evalml.demos.**load_breast_cancer**()

Load breast cancer dataset. Binary classification problem.

Returns X and y

Return type (pd.Dataframe, pd.Series)

5.1.4 evalml.demos.load_diabetes

evalml.demos.**load_diabetes**()

Load diabetes dataset. Regression problem

Returns X and y

Return type (pd.Dataframe, pd.Series)

5.1.5 evalml.demos.load_churn

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

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

Parameters

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

Returns X and y

Return type (pd.Dataframe, pd.Series)

5.2 Preprocessing

Utilities to preprocess data before using evalml.

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

5.2.1 evalml.preprocessing.load_data

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

Load features and target from file.

Parameters

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

Returns Features matrix and target

Return type pd.DataFrame, pd.Series

5.2.2 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, np.ndarray*) – Data to transform
- **y** (*pd.Series, np.ndarray*) – Target data

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

Return type pd.DataFrame, pd.DataFrame

5.2.3 evalml.preprocessing.target_distribution

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

Get the target distributions.

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

Returns Target data and their frequency distribution as percentages.

Return type pd.Series

5.2.4 evalml.preprocessing.number_of_features

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

Get the number of features of each specific dtype in a DataFrame.

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

Returns dtypes and the number of features for each input type

Return type pd.Series

5.2.5 evalml.preprocessing.split_data

evalml.preprocessing.**split_data**(*X*, *y*, *problem_type*, *problem_configuration=None*, *test_size=0.2*, *random_seed=0*)

Splits data into train and test sets.

Parameters

- **X** (*pd.DataFrame or np.ndarray*) – data of shape [n_samples, n_features]
- **y** (*pd.Series, or np.ndarray*) – target data of length [n_samples]
- **problem_type** (*str or ProblemTypes*) – type of supervised learning problem. see evalml.problem_types.problemtype.all_problem_types for a full list.
- **problem_configuration** (*dict*) – Additional parameters needed to configure the search. For example, in time series problems, values should be passed in for the date_index, gap, and max_delay variables.
- **test_size** (*float*) – What percentage of data points should be included in the test set. Defaults to 0.2 (20%).
- **random_seed** (*int*) – Seed for the random number generator. Defaults to 0.

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

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

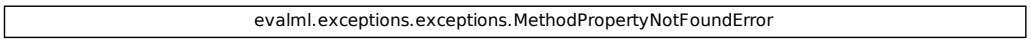
5.3 Exceptions

MethodPropertyNotFoundError	Exception to raise when a class is does not have an expected method or property.
PipelineNotFoundError	An exception raised when a particular pipeline is not found in automl search results
ObjectiveNotFoundError	Exception to raise when specified objective does not exist.
MissingComponentError	An exception raised when a component is not found in all_components()
ComponentNotYetFittedError	An exception to be raised when predict/predict_proba/transform is called on a component without fitting first.
PipelineNotYetFittedError	An exception to be raised when predict/predict_proba/transform is called on a pipeline without fitting first.
AutoMLSearchException	Exception raised when all pipelines in an automl batch return a score of NaN for the primary objective.
EnsembleMissingPipelinesError	An exception raised when an ensemble is missing *estimators* (list) as a parameter.
PipelineScoreError	An exception raised when a pipeline errors while scoring any objective in a list of objectives.
DataCheckInitError	Exception raised when a data check can't initialize with the parameters given.
NullsInColumnWarning	Warning thrown when there are null values in the column of interest

5.3.1 evalml.exceptions.MethodPropertyNotFoundError

class evalml.exceptions.**MethodPropertyNotFoundError**
Exception to raise when a class is does not have an expected method or property.

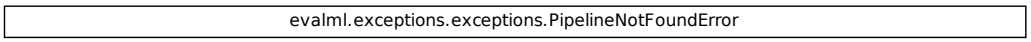
Class Inheritance



5.3.2 evalml.exceptions.PipelineNotFoundError

class evalml.exceptions.**PipelineNotFoundError**
An exception raised when a particular pipeline is not found in automl search results

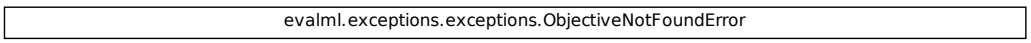
Class Inheritance



5.3.3 evalml.exceptions.ObjectiveNotFoundError

class evalml.exceptions.**ObjectiveNotFoundError**
Exception to raise when specified objective does not exist.

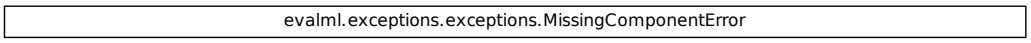
Class Inheritance



5.3.4 evalml.exceptions.MissingComponentError

class evalml.exceptions.**MissingComponentError**
An exception raised when a component is not found in all_components()

Class Inheritance



5.3.5 evalml.exceptions.ComponentNotYetFittedError

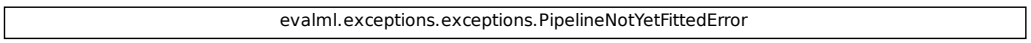
class evalml.exceptions.**ComponentNotYetFittedError**
An exception to be raised when predict/predict_proba/transform is called on a component without fitting first.

Class Inheritance

5.3.6 evalml.exceptions.PipelineNotYetFittedError

class evalml.exceptions.**PipelineNotYetFittedError**
An exception to be raised when predict/predict_proba/transform is called on a pipeline without fitting first.

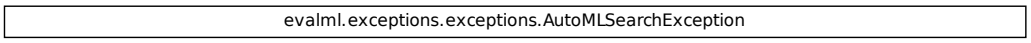
Class Inheritance



5.3.7 evalml.exceptions.AutoMLSearchException

class evalml.exceptions.**AutoMLSearchException**
Exception raised when all pipelines in an automl batch return a score of NaN for the primary objective.

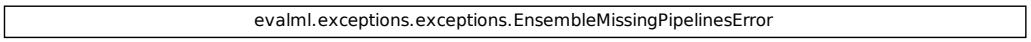
Class Inheritance



5.3.8 evalml.exceptions.EnsembleMissingPipelinesError

class evalml.exceptions.**EnsembleMissingPipelinesError**
An exception raised when an ensemble is missing *estimators* (list) as a parameter.

Class Inheritance



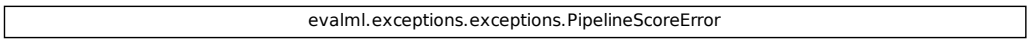
5.3.9 evalml.exceptions.PipelineScoreError

class evalml.exceptions.**PipelineScoreError**(*exceptions*, *scored_successfully*)
An exception raised when a pipeline errors while scoring any objective in a list of objectives.

Parameters

- **exceptions** (*dict*) – A dictionary mapping an objective name (str) to a tuple of the form (exception, traceback). All of the objectives that errored will be stored here.
- **scored_successfully** (*dict*) – A dictionary mapping an objective name (str) to a score value. All of the objectives that did not error will be stored here.

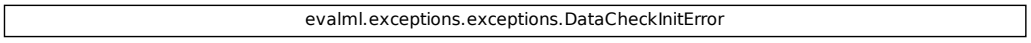
Class Inheritance



5.3.10 evalml.exceptions.DataCheckInitError

class evalml.exceptions.**DataCheckInitError**
Exception raised when a data check can't initialize with the parameters given.

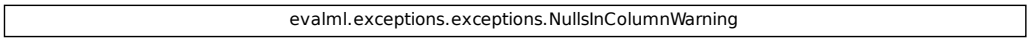
Class Inheritance



5.3.11 evalml.exceptions.NullsInColumnWarning

class evalml.exceptions.**NullsInColumnWarning**
Warning thrown when there are null values in the column of interest

Class Inheritance



5.4 AutoML

5.4.1 AutoML Search Interface

AutoMLSearch	Automated Pipeline search.

evalml.automl.AutoMLSearch

class evalml.automl.**AutoMLSearch**(*X_train=None*, *y_train=None*, *problem_type=None*, *objective='auto'*, *max_iterations=None*, *max_time=None*, *patience=None*, *tolerance=None*, *data_splitter=None*, *allowed_component_graphs=None*, *allowed_model_families=None*, *start_iteration_callback=None*, *add_result_callback=None*, *error_callback=None*, *additional_objectives=None*, *alternate_thresholding_objective='F1'*, *random_seed=0*, *n_jobs=- 1*, *tuner_class=None*, *optimize_thresholds=True*, *ensembling=False*, *max_batches=None*, *problem_configuration=None*, *train_best_pipeline=True*, *pipeline_parameters=None*, *custom_hyperparameters=None*, *sampler_method='auto'*, *sampler_balanced_ratio=0.25*, *_ensembling_split_size=0.2*, *_pipelines_per_batch=5*, *engine=None*)

Automated Pipeline search.

Methods

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

evalml.automl.AutoMLSearch.__init__

AutoMLSearch.**__init__**(*X_train=None*, *y_train=None*, *problem_type=None*, *objective='auto'*, *max_iterations=None*, *max_time=None*, *patience=None*, *tolerance=None*, *data_splitter=None*, *allowed_component_graphs=None*, *allowed_model_families=None*, *start_iteration_callback=None*, *add_result_callback=None*, *error_callback=None*, *additional_objectives=None*, *alternate_thresholding_objective='F1'*, *random_seed=0*, *n_jobs=- 1*, *tuner_class=None*, *optimize_thresholds=True*, *ensembling=False*, *max_batches=None*, *problem_configuration=None*, *train_best_pipeline=True*, *pipeline_parameters=None*, *custom_hyperparameters=None*, *sampler_method='auto'*, *sampler_balanced_ratio=0.25*, *_ensembling_split_size=0.2*, *_pipelines_per_batch=5*, *engine=None*)

Automated pipeline search

Parameters

- **X_train** (*pd.DataFrame*) – The input training data of shape [n_samples, n_features]. Required.
- **y_train** (*pd.Series*) – The target training data of length [n_samples]. Required for supervised learning tasks.
- **problem_type** (*str or ProblemTypes*) – type of supervised learning problem. See evalml.problem_types.ProblemType.all_problem_types for a full list.
- **objective** (*str, ObjectiveBase*) – The objective to optimize for. Used to propose and rank pipelines, but not for optimizing each pipeline during fit-time. When set to 'auto', chooses:
  - LogLossBinary for binary classification problems,
  - LogLossMulticlass for multiclass classification problems, and
  - R2 for regression problems.
- **max_iterations** (*int*) – Maximum number of iterations to search. If max_iterations and max_time is not set, then max_iterations will default to max_iterations of 5.
- **max_time** (*int, str*) – Maximum time to search for pipelines. This will not start a new pipeline search after the duration has elapsed. If it is an integer, then the time will be in seconds. For strings, time can be specified as seconds, minutes, or hours.
- **patience** (*int*) – Number of iterations without improvement to stop search early. Must be positive. If None, early stopping is disabled. Defaults to None.
- **tolerance** (*float*) – Minimum percentage difference to qualify as score improvement for early stopping. Only applicable if patience is not None. Defaults to None.
- **allowed_component_graphs** (*dict*) – A dictionary of lists or ComponentGraphs indicating the component graphs allowed in the search. The format should follow { "Name_0": [list_of_components], "Name_1": [ComponentGraph(...)] }

  The default of None indicates all pipeline component graphs for this problem type are allowed. Setting this field will cause allowed_model_families to be ignored.

  e.g. allowed_component_graphs = { "My_Graph": ["Imputer", "One Hot Encoder", "Random Forest Classifier"] }
- **allowed_model_families** (*list(str, ModelFamily)*) – The model families to search. The default of None searches over all model families.

Run evalml.pipelines.components.utils.allowed_model_families("binary") to see options. Change *binary* to *multiclass* or *regression* depending on the problem type. Note that if allowed_pipelines is provided, this parameter will be ignored.

- **data_splitter** (*sklearn.model_selection.BaseCrossValidator*) – Data splitting method to use. Defaults to StratifiedKFold.
- **tuner_class** – The tuner class to use. Defaults to SKOptTuner.
- **optimize_thresholds** (*bool*) – Whether or not to optimize the binary pipeline threshold. Defaults to True.
- **start_iteration_callback** (*callable*) – Function called before each pipeline training iteration. Callback function takes three positional parameters: The pipeline instance and the AutoMLSearch object.
- **add_result_callback** (*callable*) – Function called after each pipeline training iteration. Callback function takes three positional parameters: A dictionary containing the training results for the new pipeline, an untrained_pipeline containing the parameters used during training, and the AutoMLSearch object.
- **error_callback** (*callable*) – Function called when *search()* errors and raises an Exception. Callback function takes three positional parameters: the Exception raised, the traceback, and the AutoMLSearch object. Must also accepts kwargs, so AutoMLSearch is able to pass along other appropriate parameters by default. Defaults to None, which will call *log_error_callback*.
- **additional_objectives** (*list*) – Custom set of objectives to score on. Will override default objectives for problem type if not empty.
- **alternate_thresholding_objective** (*str*) – The objective to use for thresholding binary classification pipelines if the main objective provided isn't tuneable. Defaults to F1.
- **random_seed** (*int*) – Seed for the random number generator. 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.
- **ensembling** (*boolean*) – If True, runs ensembling in a separate batch after every allowed pipeline class has been iterated over. If the number of unique pipelines to search over per batch is one, ensembling will not run. Defaults to False.
- **max_batches** (*int*) – The maximum number of batches of pipelines to search. Parameters max_time, and max_iterations have precedence over stopping the search.
- **problem_configuration** (*dict, None*) – Additional parameters needed to configure the search. For example, in time series problems, values should be passed in for the date_index, gap, and max_delay variables.
- **train_best_pipeline** (*boolean*) – Whether or not to train the best pipeline before returning it. Defaults to True.
- **pipeline_parameters** (*dict*) – A dict of the parameters used to initialize a pipeline with. Keys should consist of the component names and values should specify parameter values

  e.g. pipeline_parameters = { 'Imputer' : { 'numeric_impute_strategy': 'most_frequent' } }

- **custom_hyperparameters** (*dict*) – A dict of the hyperparameter ranges used to iterate over during search. Keys should consist of the component names and values should specify a singular value or skopt.Space.

  e.g. custom_hyperparameters = { 'Imputer' : { 'numeric_impute_strategy': Categorical(['most_frequent', 'median']) } }

- **sampler_method** (*str*) – The data sampling component to use in the pipelines if the problem type is classification and the target balance is smaller than the sampler_balanced_ratio. Either 'auto', which will use our preferred sampler for the data, 'Undersampler', 'Oversampler', or None. Defaults to 'auto'.

- **sampler_balanced_ratio** (*float*) – The minority:majority class ratio that we consider balanced, so a 1:4 ratio would be equal to 0.25. If the class balance is larger than this provided value, then we will not add a sampler since the data is then considered balanced. Overrides the *sampler_ratio* of the samplers. Defaults to 0.25.

- **_ensembling_split_size** (*float*) – The amount of the training data we'll set aside for training ensemble metalearners. Only used when ensembling is True. Must be between 0 and 1, exclusive. Defaults to 0.2

- **_pipelines_per_batch** (*int*) – The number of pipelines to train for every batch after the first one. The first batch will train a baseline pipline + one of each pipeline family allowed in the search.

- **engine** (*EngineBase or None*) – The engine instance used to evaluate pipelines. If None, a SequentialEngine will be used.

evalml.automl.AutoMLSearch.add_to_rankings

AutoMLSearch.**add_to_rankings**(*pipeline*)

Fits and evaluates a given pipeline then adds the results to the automl rankings with the requirement that automl search has been run.

Parameters **pipeline** (PipelineBase) – pipeline to train and evaluate.

evalml.automl.AutoMLSearch.describe_pipeline

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

Describe a pipeline

Parameters

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

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

evalml.automl.AutoMLSearch.get_pipeline

AutoMLSearch.**get_pipeline**(*pipeline_id*)

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

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

Returns untrained pipeline instance associated with the provided ID

Return type *PipelineBase*

evalml.automl.AutoMLSearch.load

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

Loads AutoML object at file path

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

Returns AutoSearchBase object

evalml.automl.AutoMLSearch.save

AutoMLSearch.**save**(*file_path*, *pickle_protocol=5*)

Saves AutoML object at file path

Parameters

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

Returns None

evalml.automl.AutoMLSearch.score_pipelines

AutoMLSearch.**score_pipelines**(*pipelines*, *X_holdout*, *y_holdout*, *objectives*)

Score a list of pipelines on the given holdout data.

Parameters

- **pipelines** (*list(PipelineBase)*) – List of pipelines to train.
- **X_holdout** (*pd.DataFrame*) – Holdout features.
- **y_holdout** (*pd.Series*) – Holdout targets for scoring.
- **objectives** (*list(str), list(ObjectiveBase)*) – Objectives used for scoring.

Returns Dictionary keyed by pipeline name that maps to a dictionary of scores. Note that the any pipelines that error out during scoring will not be included in the dictionary but the exception and stacktrace will be displayed in the log.

Return type Dict[str, Dict[str, float]]

evalml.automl.AutoMLSearch.search

AutoMLSearch.**search**(*show_iteration_plot=True*)

Find the best pipeline for the data set.

Parameters

- **feature_types** (*list, optional*) – list of feature types, either numerical or categorical. Categorical features will automatically be encoded
- **show_iteration_plot** (*boolean, True*) – Shows an iteration vs. score plot in Jupyter notebook. Disabled by default in non-Jupyter enviroments.

evalml.automl.AutoMLSearch.train_pipelines

AutoMLSearch.**train_pipelines**(*pipelines*)

Train a list of pipelines on the training data.

This can be helpful for training pipelines once the search is complete.

Parameters **pipelines** (*list(PipelineBase)*) – List of pipelines to train.

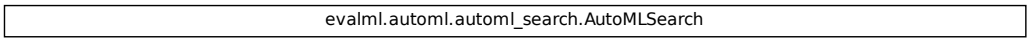
Returns Dictionary keyed by pipeline name that maps to the fitted pipeline. Note that the any pipelines that error out during training will not be included in the dictionary but the exception and stacktrace will be displayed in the log.

Return type Dict[str, *PipelineBase*]

Attributes

best_pipeline	Returns a trained instance of the best pipeline and parameters found during automl search.
full_rankings	Returns a pandas.DataFrame with scoring results from all pipelines searched
plot	
rankings	Returns a pandas.DataFrame with scoring results from the highest-scoring set of parameters used with each pipeline.
results	Class that allows access to a copy of the results from *automl_search*.

Class Inheritance



5.4.2 AutoML Utils

search	Given data and configuration, run an automl search.
get_default_primary_search_objective	Get the default primary search objective for a problem type.
make_data_splitter	Given the training data and ML problem parameters, compute a data splitting method to use during AutoML search.

evalml.automl.search

evalml.automl.**search**(*X_train=None*, *y_train=None*, *problem_type=None*, *objective='auto'*, ***kwargs*)

Given data and configuration, run an automl search.

This method will run EvalML's default suite of data checks. If the data checks produce errors, the data check results will be returned before running the automl search. In that case we recommend you alter your data to address these errors and try again.

This method is provided for convenience. If you'd like more control over when each of these steps is run, consider making calls directly to the various pieces like the data checks and AutoMLSearch, instead of using this method.

Parameters

- **X_train** (*pd.DataFrame*) – The input training data of shape [n_samples, n_features]. Required.
- **y_train** (*pd.Series*) – The target training data of length [n_samples]. Required for supervised learning tasks.
- **problem_type** (*str or ProblemTypes*) – type of supervised learning problem. See evalml.problem_types.ProblemType.all_problem_types for a full list.
- **objective** (*str, ObjectiveBase*) – The objective to optimize for. Used to propose and rank pipelines, but not for optimizing each pipeline during fit-time. When set to 'auto', chooses:
  - LogLossBinary for binary classification problems,
  - LogLossMulticlass for multiclass classification problems, and
  - R2 for regression problems.

Other keyword arguments which are provided will be passed to AutoMLSearch.

Returns the automl search object containing pipelines and rankings, and the results from running the data checks. If the data check results contain errors, automl search will not be run and an automl search object will not be returned.

Return type (*AutoMLSearch*, dict)

evalml.automl.get_default_primary_search_objective

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

Get the default primary search objective for a problem type.

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

Returns primary objective instance for the problem type.

Return type *ObjectiveBase*

evalml.automl.make_data_splitter

evalml.automl.**make_data_splitter**(*X*, *y*, *problem_type*, *problem_configuration=None*, *n_splits=3*, *shuffle=True*, *random_seed=0*)

Given the training data and ML problem parameters, compute a data splitting method to use during AutoML search.

Parameters

- **X** (*pd.DataFrame*) – The input training data of shape [n_samples, n_features].
- **y** (*pd.Series*) – The target training data of length [n_samples].
- **problem_type** (*ProblemType*) – The type of machine learning problem.
- **problem_configuration** (*dict, None*) – Additional parameters needed to configure the search. For example, in time series problems, values should be passed in for the date_index, gap, and max_delay variables. Defaults to None.
- **n_splits** (*int, None*) – The number of CV splits, if applicable. Defaults to 3.
- **shuffle** (*bool*) – Whether or not to shuffle the data before splitting, if applicable. Defaults to True.
- **random_seed** (*int*) – Seed for the random number generator. Defaults to 0.

Returns Data splitting method.

Return type sklearn.model_selection.BaseCrossValidator

5.4.3 AutoML Algorithm Classes

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

evalml.automl.automl_algorithm.AutoMLAlgorithm

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

Base class for the automl algorithms which power evalml.

Methods

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

evalml.automl.automl_algorithm.AutoMLAlgorithm.__init__

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

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

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

Parameters

- **allowed_pipelines** (*list (class)*) – A list of PipelineBase subclasses indicating the pipelines allowed in the search. The default of None indicates all pipelines for this problem type are allowed.
- **custom_hyperparameters** (*dict*) – Custom hyperparameter ranges specified for pipelines to iterate over.
- **max_iterations** (*int*) – The maximum number of iterations to be evaluated.
- **tuner_class** (*class*) – A subclass of Tuner, to be used to find parameters for each pipeline. The default of None indicates the SKOptTuner will be used.
- **random_seed** (*int*) – Seed for the random number generator. Defaults to 0.

evalml.automl.automl_algorithm.AutoMLAlgorithm.add_result

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

Register results from evaluating a pipeline

Parameters

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

evalml.automl.automl_algorithm.AutoMLAlgorithm.next_batch

abstract AutoMLAlgorithm.**next_batch**()

Get the next batch of pipelines to evaluate

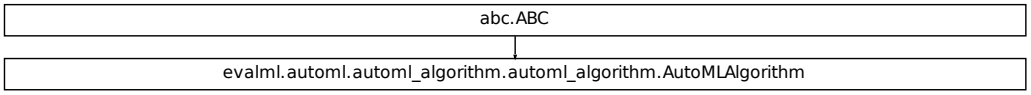
Returns a list of instances of PipelineBase subclasses, ready to be trained and evaluated.

Return type list(*PipelineBase*)

Attributes

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

Class Inheritance



evalml.automl.automl_algorithm.IterativeAlgorithm

class evalml.automl.automl_algorithm.**IterativeAlgorithm**(*allowed_pipelines=None*, *max_iterations=None*, *tuner_class=None*, *random_seed=0*, *pipelines_per_batch=5*, *n_jobs=-1*, *number_features=None*, *ensembling=False*, *text_in_ensembling=False*, *pipeline_params=None*, *custom_hyperparameters=None*, *_estimator_family_order=None*)

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

Methods

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

evalml.automl.automl_algorithm.IterativeAlgorithm.__init__

IterativeAlgorithm.**__init__**(*allowed_pipelines=None*, *max_iterations=None*, *tuner_class=None*, *random_seed=0*, *pipelines_per_batch=5*, *n_jobs=-1*, *number_features=None*, *ensembling=False*, *text_in_ensembling=False*, *pipeline_params=None*, *custom_hyperparameters=None*, *_estimator_family_order=None*)

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

Parameters

- **allowed_pipelines** (*list(class)*) – A list of PipelineBase instances indicating the pipelines allowed in the search. The default of None indicates all pipelines for this problem type are allowed.
- **max_iterations** (*int*) – The maximum number of iterations to be evaluated.
- **tuner_class** (*class*) – A subclass of Tuner, to be used to find parameters for each pipeline. The default of None indicates the SKOptTuner will be used.
- **random_seed** (*int*) – Seed for the random number generator. Defaults to 0.
- **pipelines_per_batch** (*int*) – The number of pipelines to be evaluated in each batch, after the first batch.

- **n_jobs** (*int or None*) – Non-negative integer describing level of parallelism used for pipelines.
- **number_features** (*int*) – The number of columns in the input features.
- **ensembling** (*boolean*) – If True, runs ensembling in a separate batch after every allowed pipeline class has been iterated over. Defaults to False.
- **text_in_ensembling** (*boolean*) – If True and ensembling is True, then n_jobs will be set to 1 to avoid downstream sklearn stacking issues related to nltk.
- **pipeline_params** (*dict or None*) – Pipeline-level parameters that should be passed to the proposed pipelines.
- **custom_hyperparameters** (*dict or None*) – Custom hyperparameter ranges specified for pipelines to iterate over.
- **_estimator_family_order** (*list(ModelFamily) or None*) – specify the sort order for the first batch. Defaults to _ESTIMATOR_FAMILY_ORDER.

evalml.automl.automl_algorithm.IterativeAlgorithm.add_result

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

Register results from evaluating a pipeline

Parameters

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

evalml.automl.automl_algorithm.IterativeAlgorithm.next_batch

IterativeAlgorithm.**next_batch**()

Get the next batch of pipelines to evaluate

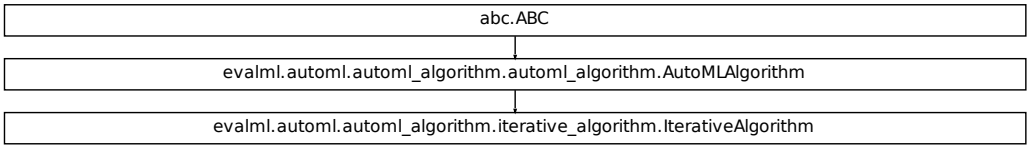
Returns a list of instances of PipelineBase subclasses, ready to be trained and evaluated.

Return type list(*PipelineBase*)

Attributes

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

Class Inheritance



5.4.4 AutoML Callbacks

silent_error_callback	No-op.
log_error_callback	Logs the exception thrown as an error.
raise_error_callback	Raises the exception thrown by the AutoMLSearch object.

evalml.automl.callbacks.silent_error_callback

evalml.automl.callbacks.**silent_error_callback**(*exception*, *traceback*, *automl*, ***kwargs*)
No-op.

evalml.automl.callbacks.log_error_callback

evalml.automl.callbacks.**log_error_callback**(*exception*, *traceback*, *automl*, ***kwargs*)
Logs the exception thrown as an error. Will not throw. This is the default behavior for AutoMLSearch.

evalml.automl.callbacks.raise_error_callback

evalml.automl.callbacks.**raise_error_callback**(*exception*, *traceback*, *automl*, ***kwargs*)
Raises the exception thrown by the AutoMLSearch object. Also logs the exception as an error.

5.5 Pipelines

5.5.1 Pipeline Base Classes

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

continues on next page

Table 14 – continued from previous page

RegressionPipeline	Pipeline subclass for all regression pipelines.
TimeSeriesClassificationPipeline	Pipeline base class for time series classification problems.
TimeSeriesBinaryClassificationPipeline	
TimeSeriesMulticlassClassificationPipeline	
TimeSeriesRegressionPipeline	Pipeline base class for time series regression problems.

evalml.pipelines.PipelineBase

class evalml.pipelines.**PipelineBase**(*component_graph*, *parameters=None*, *custom_name=None*, *random_seed=0*)

Base class for all pipelines.

Instance attributes

`custom_name`	Custom name of the pipeline.
`feature_importance`	Importance associated with each feature.
`linearized_component_graph`	this is not guaranteed to be in proper component computation order
`model_family`	Returns model family of this pipeline template
`name`	Name of the pipeline.
`parameters`	Parameter dictionary for this pipeline
`problem_type`	
`summary`	A short summary of the pipeline structure, describing the list of components used.

Methods:

__init__	Machine learning pipeline made out of transformers and a estimator.
can_tune_threshold_with_objective	Determine whether the threshold of a binary classification pipeline can be tuned.
clone	Constructs a new pipeline with the same components, parameters, and random state.
compute_estimator_features	Transforms the data by applying all pre-processing components.
create_objectives	
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

continues on next page

Table 16 – continued from previous page

graph_feature_importance	Generate a bar graph of the pipeline's feature importance
inverse_transform	Apply component inverse_transform methods to estimator predictions in reverse order.
load	Loads pipeline at file path
new	Constructs a new instance of the pipeline with the same component graph but with a different set of parameters.
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__**(*component_graph*, *parameters=None*, *custom_name=None*, *random_seed=0*)

Machine learning pipeline made out of transformers and a estimator.

Parameters

- **component_graph** (*list or dict*) – List of components in order. Accepts strings or ComponentBase subclasses in the list. Note that when duplicate components are specified in a list, the duplicate component names will be modified with the component's index in the list. For example, the component graph [Imputer, One Hot Encoder, Imputer, Logistic Regression Classifier] will have names ["Imputer", "One Hot Encoder", "Imputer_2", "Logistic Regression Classifier"]
- **parameters** (*dict*) – Dictionary with component names as keys and dictionary of that component's parameters as values. An empty dictionary or None implies using all default values for component parameters. Defaults to None.
- **custom_name** (*str*) – Custom name for the pipeline. Defaults to None.
- **random_seed** (*int*) – Seed for the random number generator. Defaults to 0.

evalml.pipelines.PipelineBase.can_tune_threshold_with_objective

PipelineBase.**can_tune_threshold_with_objective**(*objective*)

Determine whether the threshold of a binary classification pipeline can be tuned.

Parameters

- **pipeline** (PipelineBase) – Binary classification pipeline.
- **objective** – Primary AutoMLSearch objective.

evalml.pipelines.PipelineBase.clone

PipelineBase.**clone**()

Constructs a new pipeline with the same components, parameters, and random state.

Returns A new instance of this pipeline with identical components, parameters, and random state.

evalml.pipelines.PipelineBase.compute_estimator_features

PipelineBase.**compute_estimator_features**(*X*, *y=None*)

Transforms the data by applying all pre-processing components.

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

Returns New transformed features.

Return type pd.DataFrame

evalml.pipelines.PipelineBase.create_objectives

static PipelineBase.**create_objectives**(*objectives*)

evalml.pipelines.PipelineBase.describe

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

Outputs pipeline details including component parameters

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

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

Return type dict

evalml.pipelines.PipelineBase.fit

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

Build a model

Parameters

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

Returns self

evalml.pipelines.PipelineBase.get_component

PipelineBase.**get_component**(*name*)

Returns component by name

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

Returns Component to return

Return type Component

evalml.pipelines.PipelineBase.graph

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

Generate an image representing the pipeline graph

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

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

Return type graphviz.Digraph

evalml.pipelines.PipelineBase.graph_feature_importance

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

Generate a bar graph of the pipeline's feature importance

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

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

evalml.pipelines.PipelineBase.inverse_transform

PipelineBase.**inverse_transform**(*y*)

Apply component inverse_transform methods to estimator predictions in reverse order.

Components that implement inverse_transform are PolynomialDetrender, LabelEncoder (tbd).

Parameters **y** (*pd.Series*) – Final component features

evalml.pipelines.PipelineBase.load

static PipelineBase.**load**(*file_path*)

Loads pipeline at file path

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

Returns PipelineBase object

evalml.pipelines.PipelineBase.new

PipelineBase.**new**(*parameters*, *random_seed=0*)

Constructs a new instance of the pipeline with the same component graph but with a different set of parameters.
Not to be confused with python's __new__ method.

Parameters

- **parameters** (*dict*) – Dictionary with component names as keys and dictionary of that component's parameters as values. An empty dictionary or None implies using all default values for component parameters. Defaults to None.
- **random_seed** (*int*) – Seed for the random number generator. Defaults to 0.

Returns A new instance of this pipeline with identical components.

evalml.pipelines.PipelineBase.predict

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

Parameters

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

Returns Predicted values.

Return type pd.Series

evalml.pipelines.PipelineBase.save

PipelineBase.**save**(*file_path*, *pickle_protocol=5*)
Saves pipeline at file path

Parameters

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

Returns None

evalml.pipelines.PipelineBase.score

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

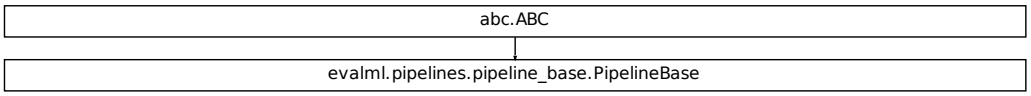
Parameters

- **X** (*pd.DataFrame or np.ndarray*) – Data of shape [n_samples, n_features]
- **y** (*pd.Series, np.ndarray*) – 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

Class Inheritance



evalml.pipelines.ClassificationPipeline

class evalml.pipelines.**ClassificationPipeline**(*component_graph*, *parameters=None*, *custom_name=None*, *random_seed=0*)

Pipeline subclass for all classification pipelines.

Instance attributes

classes_	Gets the class names for the problem.
custom_name	Custom name of the pipeline.
feature_importance	Importance associated with each feature.
linearized_component_graph	this is not guaranteed to be in proper component computation order
model_family	Returns model family of this pipeline template
name	Name of the pipeline.
parameters	Parameter dictionary for this pipeline
problem_type	
summary	A short summary of the pipeline structure, describing the list of components used.

Methods:

__init__	Machine learning pipeline made out of transformers and a estimator.
can_tune_threshold_with_objective	Determine whether the threshold of a binary classification pipeline can be tuned.
clone	Constructs a new pipeline with the same components, parameters, and random state.
compute_estimator_features	Transforms the data by applying all pre-processing components.
create_objectives	

continues on next page

Table 18 – continued from previous page

describe	Outputs pipeline details including component parameters
fit	Build a classification model. For string and categorical targets, classes are sorted
get_component	Returns component by name
graph	Generate an image representing the pipeline graph
graph_feature_importance	Generate a bar graph of the pipeline's feature importance
inverse_transform	Apply component inverse_transform methods to estimator predictions in reverse order.
load	Loads pipeline at file path
new	Constructs a new instance of the pipeline with the same component graph but with a different set of parameters.
predict	Make predictions using selected features.
predict_proba	Make probability estimates for labels.
save	Saves pipeline at file path
score	Evaluate model performance on objectives

evalml.pipelines.ClassificationPipeline.__init__

ClassificationPipeline.**__init__**(*component_graph*, *parameters=None*, *custom_name=None*, *random_seed=0*)

Machine learning pipeline made out of transformers and a estimator.

Parameters

- **component_graph** (*list or dict*) – List of components in order. Accepts strings or ComponentBase subclasses in the list. Note that when duplicate components are specified in a list, the duplicate component names will be modified with the component's index in the list. For example, the component graph [Imputer, One Hot Encoder, Imputer, Logistic Regression Classifier] will have names ["Imputer", "One Hot Encoder", "Imputer_2", "Logistic Regression Classifier"]
- **parameters** (*dict*) – Dictionary with component names as keys and dictionary of that component's parameters as values. An empty dictionary or None implies using all default values for component parameters. Defaults to None.
- **custom_name** (*str*) – Custom name for the pipeline. Defaults to None.
- **random_seed** (*int*) – Seed for the random number generator. Defaults to 0.

evalml.pipelines.ClassificationPipeline.can_tune_threshold_with_objective

ClassificationPipeline.**can_tune_threshold_with_objective**(*objective*)

Determine whether the threshold of a binary classification pipeline can be tuned.

Parameters

- **pipeline** (PipelineBase) – Binary classification pipeline.
- **objective** – Primary AutoMLSearch objective.

evalml.pipelines.ClassificationPipeline.clone

ClassificationPipeline.**clone**()

Constructs a new pipeline with the same components, parameters, and random state.

Returns A new instance of this pipeline with identical components, parameters, and random state.

evalml.pipelines.ClassificationPipeline.compute_estimator_features

ClassificationPipeline.**compute_estimator_features**(*X*, *y=None*)

Transforms the data by applying all pre-processing components.

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

Returns New transformed features.

Return type pd.DataFrame

evalml.pipelines.ClassificationPipeline.create_objectives

static ClassificationPipeline.**create_objectives**(*objectives*)

evalml.pipelines.ClassificationPipeline.describe

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

Outputs pipeline details including component parameters

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

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

Return type dict

evalml.pipelines.ClassificationPipeline.fit

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

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

Parameters

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

Returns self

evalml.pipelines.ClassificationPipeline.get_component

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

Returns component by name

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

Returns Component to return

Return type Component

evalml.pipelines.ClassificationPipeline.graph

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

Generate an image representing the pipeline graph

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

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

Return type graphviz.Digraph

evalml.pipelines.ClassificationPipeline.graph_feature_importance

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

Generate a bar graph of the pipeline's feature importance

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

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

evalml.pipelines.ClassificationPipeline.inverse_transform

ClassificationPipeline.**inverse_transform**(*y*)

Apply component inverse_transform methods to estimator predictions in reverse order.

Components that implement inverse_transform are PolynomialDetrender, LabelEncoder (tbd).

Parameters **y** (*pd.Series*) – Final component features

evalml.pipelines.ClassificationPipeline.load

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

Loads pipeline at file path

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

Returns PipelineBase object

evalml.pipelines.ClassificationPipeline.new

ClassificationPipeline.**new**(*parameters*, *random_seed=0*)

Constructs a new instance of the pipeline with the same component graph but with a different set of parameters.
Not to be confused with python's __new__ method.

Parameters

- **parameters** (*dict*) – Dictionary with component names as keys and dictionary of that component's parameters as values. An empty dictionary or None implies using all default values for component parameters. Defaults to None.
- **random_seed** (*int*) – Seed for the random number generator. Defaults to 0.

Returns A new instance of this pipeline with identical components.

evalml.pipelines.ClassificationPipeline.predict

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

Parameters

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

Returns Probability estimates

Return type pd.DataFrame

evalml.pipelines.ClassificationPipeline.save

ClassificationPipeline.**save**(*file_path*, *pickle_protocol=5*)
Saves pipeline at file path

Parameters

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

Returns None

evalml.pipelines.ClassificationPipeline.score

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

Evaluate model performance on objectives

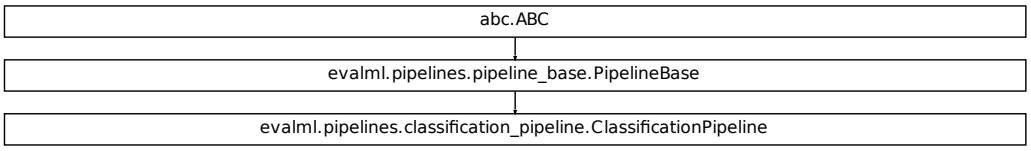
Parameters

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

Returns Ordered dictionary of objective scores

Return type dict

Class Inheritance



evalml.pipelines.BinaryClassificationPipeline

class evalml.pipelines.**BinaryClassificationPipeline**(*component_graph*, *parameters=None*, *custom_name=None*, *random_seed=0*)

Pipeline subclass for all binary classification pipelines.

Instance attributes

classes_	Gets the class names for the problem.
custom_name	Custom name of the pipeline.
feature_importance	Importance associated with each feature.
linearized_component_graph	this is not guaranteed to be in proper component computation order
model_family	Returns model family of this pipeline template
name	Name of the pipeline.
parameters	Parameter dictionary for this pipeline
problem_type	
summary	A short summary of the pipeline structure, describing the list of components used.

continues on next page

Table 19 – continued from previous page

threshold	Threshold used to make a prediction.

Methods:

__init__	Machine learning pipeline made out of transformers and a estimator.
can_tune_threshold_with_objective	Determine whether the threshold of a binary classification pipeline can be tuned.
clone	Constructs a new pipeline with the same components, parameters, and random state.
compute_estimator_features	Transforms the data by applying all pre-processing components.
create_objectives	
describe	Outputs pipeline details including component parameters
fit	Build a classification model. For string and categorical targets, classes are sorted
get_component	Returns component by name
graph	Generate an image representing the pipeline graph
graph_feature_importance	Generate a bar graph of the pipeline's feature importance
inverse_transform	Apply component inverse_transform methods to estimator predictions in reverse order.
load	Loads pipeline at file path
new	Constructs a new instance of the pipeline with the same component graph but with a different set of parameters.
optimize_threshold	Optimize the pipeline threshold given the objective to use.
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__**(*component_graph*, *parameters=None*, *custom_name=None*, *random_seed=0*)

Machine learning pipeline made out of transformers and a estimator.

Parameters

- **component_graph** (*list or dict*) – List of components in order. Accepts strings or ComponentBase subclasses in the list. Note that when duplicate components are specified in a list, the duplicate component names will be modified with the component's index in the list. For example, the component graph [Imputer, One Hot Encoder, Imputer, Logistic Regression Classifier] will have names ["Imputer", "One Hot Encoder", "Imputer_2", "Logistic Regression Classifier"]
- **parameters** (*dict*) – Dictionary with component names as keys and dictionary of that

component's parameters as values. An empty dictionary or None implies using all default values for component parameters. Defaults to None.

- **custom_name** (*str*) – Custom name for the pipeline. Defaults to None.
- **random_seed** (*int*) – Seed for the random number generator. Defaults to 0.

evalml.pipelines.BinaryClassificationPipeline.can_tune_threshold_with_objective

BinaryClassificationPipeline.**can_tune_threshold_with_objective**(*objective*)

Determine whether the threshold of a binary classification pipeline can be tuned.

Parameters

- **pipeline** (PipelineBase) – Binary classification pipeline.
- **objective** – Primary AutoMLSearch objective.

evalml.pipelines.BinaryClassificationPipeline.clone

BinaryClassificationPipeline.**clone**()

Constructs a new pipeline with the same components, parameters, and random state.

Returns A new instance of this pipeline with identical components, parameters, and random state.

evalml.pipelines.BinaryClassificationPipeline.compute_estimator_features

BinaryClassificationPipeline.**compute_estimator_features**(*X*, *y=None*)

Transforms the data by applying all pre-processing components.

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

Returns New transformed features.

Return type pd.DataFrame

evalml.pipelines.BinaryClassificationPipeline.create_objectives

static BinaryClassificationPipeline.**create_objectives**(*objectives*)

evalml.pipelines.BinaryClassificationPipeline.describe

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

Outputs pipeline details including component parameters

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

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

Return type dict

evalml.pipelines.BinaryClassificationPipeline.fit

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

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

Parameters

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

Returns self

evalml.pipelines.BinaryClassificationPipeline.get_component

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

Returns component by name

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

Returns Component to return

Return type Component

evalml.pipelines.BinaryClassificationPipeline.graph

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

Generate an image representing the pipeline graph

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

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

Return type graphviz.Digraph

evalml.pipelines.BinaryClassificationPipeline.graph_feature_importance

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

Generate a bar graph of the pipeline's feature importance

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

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

evalml.pipelines.BinaryClassificationPipeline.inverse_transform

BinaryClassificationPipeline.**inverse_transform**(*y*)

Apply component inverse_transform methods to estimator predictions in reverse order.

Components that implement inverse_transform are PolynomialDetrender, LabelEncoder (tbd).

Parameters **y** (*pd.Series*) – Final component features

evalml.pipelines.BinaryClassificationPipeline.load

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

Loads pipeline at file path

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

Returns PipelineBase object

evalml.pipelines.BinaryClassificationPipeline.new

BinaryClassificationPipeline.**new**(*parameters*, *random_seed=0*)

Constructs a new instance of the pipeline with the same component graph but with a different set of parameters.
Not to be confused with python's __new__ method.

Parameters

- **parameters** (*dict*) – Dictionary with component names as keys and dictionary of that component's parameters as values. An empty dictionary or None implies using all default values for component parameters. Defaults to None.
- **random_seed** (*int*) – Seed for the random number generator. Defaults to 0.

Returns A new instance of this pipeline with identical components.

evalml.pipelines.BinaryClassificationPipeline.optimize_threshold

BinaryClassificationPipeline.**optimize_threshold**(*X*, *y*, *y_pred_proba*, *objective*)

Optimize the pipeline threshold given the objective to use. Only used for binary problems with objectives whose thresholds can be tuned.

Parameters

- **X** (*pd.DataFrame*) – Input features
- **y** (*pd.Series*) – Input target values
- **y_pred_proba** (*pd.Series*) – The predicted probabilities of the target outputted by the pipeline
- **objective** (ObjectiveBase) – The objective to threshold with. Must have a tunable threshold.

evalml.pipelines.BinaryClassificationPipeline.predict

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

Make predictions using selected features.

Parameters

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

Returns Estimated labels

Return type pd.Series

evalml.pipelines.BinaryClassificationPipeline.predict_proba

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

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

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

Returns Probability estimates

Return type pd.Series

evalml.pipelines.BinaryClassificationPipeline.save

BinaryClassificationPipeline.**save**(*file_path*, *pickle_protocol=5*)

Saves pipeline at file path

Parameters

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

Returns None

evalml.pipelines.BinaryClassificationPipeline.score

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

Evaluate model performance on objectives

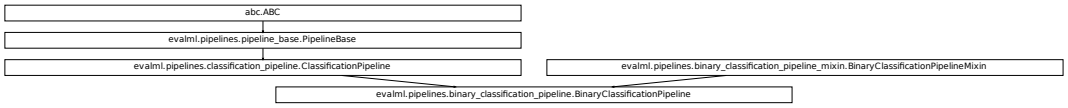
Parameters

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

Returns Ordered dictionary of objective scores

Return type dict

Class Inheritance



evalml.pipelines.MulticlassClassificationPipeline

class evalml.pipelines.**MulticlassClassificationPipeline**(*component_graph*, *parameters=None*, *custom_name=None*, *random_seed=0*)

Pipeline subclass for all multiclass classification pipelines.

Instance attributes

classes_	Gets the class names for the problem.
custom_name	Custom name of the pipeline.
feature_importance	Importance associated with each feature.
linearized_component_graph	this is not guaranteed to be in proper component computation order
model_family	Returns model family of this pipeline template
name	Name of the pipeline.
parameters	Parameter dictionary for this pipeline
problem_type	
summary	A short summary of the pipeline structure, describing the list of components used.

Methods:

__init__	Machine learning pipeline made out of transformers and a estimator.
can_tune_threshold_with_objective	Determine whether the threshold of a binary classification pipeline can be tuned.
clone	Constructs a new pipeline with the same components, parameters, and random state.
compute_estimator_features	Transforms the data by applying all pre-processing components.
create_objectives	

continues on next page

Table 22 – continued from previous page

describe	Outputs pipeline details including component parameters
fit	Build a classification model. For string and categorical targets, classes are sorted
get_component	Returns component by name
graph	Generate an image representing the pipeline graph
graph_feature_importance	Generate a bar graph of the pipeline's feature importance
inverse_transform	Apply component inverse_transform methods to estimator predictions in reverse order.
load	Loads pipeline at file path
new	Constructs a new instance of the pipeline with the same component graph but with a different set of parameters.
predict	Make predictions using selected features.
predict_proba	Make probability estimates for labels.
save	Saves pipeline at file path
score	Evaluate model performance on objectives

evalml.pipelines.MulticlassClassificationPipeline.__init__

MulticlassClassificationPipeline.**__init__**(*component_graph*, *parameters=None*, *custom_name=None*, *random_seed=0*)

Machine learning pipeline made out of transformers and a estimator.

Parameters

- **component_graph** (*list or dict*) – List of components in order. Accepts strings or ComponentBase subclasses in the list. Note that when duplicate components are specified in a list, the duplicate component names will be modified with the component's index in the list. For example, the component graph [Imputer, One Hot Encoder, Imputer, Logistic Regression Classifier] will have names ["Imputer", "One Hot Encoder", "Imputer_2", "Logistic Regression Classifier"]
- **parameters** (*dict*) – Dictionary with component names as keys and dictionary of that component's parameters as values. An empty dictionary or None implies using all default values for component parameters. Defaults to None.
- **custom_name** (*str*) – Custom name for the pipeline. Defaults to None.
- **random_seed** (*int*) – Seed for the random number generator. Defaults to 0.

evalml.pipelines.MulticlassClassificationPipeline.can_tune_threshold_with_objective

MulticlassClassificationPipeline.**can_tune_threshold_with_objective**(*objective*)

Determine whether the threshold of a binary classification pipeline can be tuned.

Parameters

- **pipeline** (PipelineBase) – Binary classification pipeline.
- **objective** – Primary AutoMLSearch objective.

evalml.pipelines.MulticlassClassificationPipeline.clone

MulticlassClassificationPipeline.**clone**()

Constructs a new pipeline with the same components, parameters, and random state.

Returns A new instance of this pipeline with identical components, parameters, and random state.

evalml.pipelines.MulticlassClassificationPipeline.compute_estimator_features

MulticlassClassificationPipeline.**compute_estimator_features**(*X*, *y=None*)

Transforms the data by applying all pre-processing components.

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

Returns New transformed features.

Return type pd.DataFrame

evalml.pipelines.MulticlassClassificationPipeline.create_objectives

static MulticlassClassificationPipeline.**create_objectives**(*objectives*)

evalml.pipelines.MulticlassClassificationPipeline.describe

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

Outputs pipeline details including component parameters

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

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

Return type dict

evalml.pipelines.MulticlassClassificationPipeline.fit

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

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

Parameters

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

Returns self

evalml.pipelines.MulticlassClassificationPipeline.get_component

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

Returns component by name

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

Returns Component to return

Return type Component

evalml.pipelines.MulticlassClassificationPipeline.graph

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

Generate an image representing the pipeline graph

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

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

Return type graphviz.Digraph

evalml.pipelines.MulticlassClassificationPipeline.graph_feature_importance

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

Generate a bar graph of the pipeline's feature importance

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

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

evalml.pipelines.MulticlassClassificationPipeline.inverse_transform

MulticlassClassificationPipeline.**inverse_transform**(*y*)

Apply component inverse_transform methods to estimator predictions in reverse order.

Components that implement inverse_transform are PolynomialDetrender, LabelEncoder (tbd).

Parameters **y** (*pd.Series*) – Final component features

evalml.pipelines.MulticlassClassificationPipeline.load

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

Loads pipeline at file path

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

Returns PipelineBase object

evalml.pipelines.MulticlassClassificationPipeline.new

MulticlassClassificationPipeline.**new**(*parameters*, *random_seed=0*)

Constructs a new instance of the pipeline with the same component graph but with a different set of parameters.
Not to be confused with python's __new__ method.

Parameters

- **parameters** (*dict*) – Dictionary with component names as keys and dictionary of that component's parameters as values. An empty dictionary or None implies using all default values for component parameters. Defaults to None.
- **random_seed** (*int*) – Seed for the random number generator. Defaults to 0.

Returns A new instance of this pipeline with identical components.

evalml.pipelines.MulticlassClassificationPipeline.predict

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

Parameters

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

Returns Probability estimates

Return type pd.DataFrame

evalml.pipelines.MulticlassClassificationPipeline.save

MulticlassClassificationPipeline.**save**(*file_path*, *pickle_protocol=5*)
Saves pipeline at file path

Parameters

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

Returns None

evalml.pipelines.MulticlassClassificationPipeline.score

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

Evaluate model performance on objectives

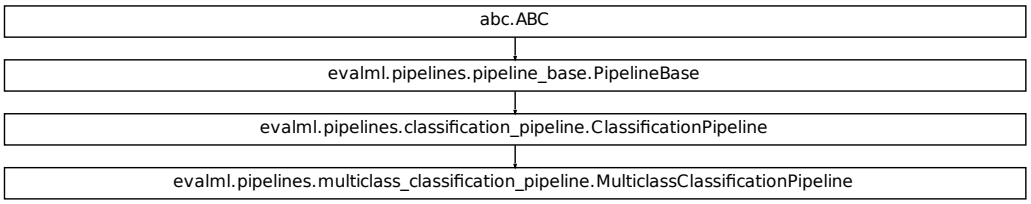
Parameters

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

Returns Ordered dictionary of objective scores

Return type dict

Class Inheritance



evalml.pipelines.RegressionPipeline

class evalml.pipelines.**RegressionPipeline**(*component_graph*, *parameters=None*, *custom_name=None*, *random_seed=0*)

Pipeline subclass for all regression pipelines.

Instance attributes

custom_name	Custom name of the pipeline.
feature_importance	Importance associated with each feature.
linearized_component_graph	this is not guaranteed to be in proper component computation order
model_family	Returns model family of this pipeline template
name	Name of the pipeline.
parameters	Parameter dictionary for this pipeline
problem_type	
summary	A short summary of the pipeline structure, describing the list of components used.

Methods:

__init__	Machine learning pipeline made out of transformers and a estimator.
can_tune_threshold_with_objective	Determine whether the threshold of a binary classification pipeline can be tuned.
clone	Constructs a new pipeline with the same components, parameters, and random state.
compute_estimator_features	Transforms the data by applying all pre-processing components.
create_objectives	
describe	Outputs pipeline details including component parameters
fit	Build a regression model.
get_component	Returns component by name
graph	Generate an image representing the pipeline graph
graph_feature_importance	Generate a bar graph of the pipeline's feature importance
inverse_transform	Apply component inverse_transform methods to estimator predictions in reverse order.
load	Loads pipeline at file path
new	Constructs a new instance of the pipeline with the same component graph but with a different set of parameters.
predict	Make predictions using selected features.
save	Saves pipeline at file path
score	Evaluate model performance on current and additional objectives

evalml.pipelines.RegressionPipeline.__init__

RegressionPipeline.**__init__**(*component_graph*, *parameters=None*, *custom_name=None*, *random_seed=0*)

Machine learning pipeline made out of transformers and a estimator.

Parameters

- **component_graph** (*list or dict*) – List of components in order. Accepts strings or ComponentBase subclasses in the list. Note that when duplicate components are specified in a list, the duplicate component names will be modified with the component's index in the list. For example, the component graph [Imputer, One Hot Encoder, Imputer, Logistic Regression Classifier] will have names ["Imputer", "One Hot Encoder", "Imputer_2", "Logistic Regression Classifier"]
- **parameters** (*dict*) – Dictionary with component names as keys and dictionary of that component's parameters as values. An empty dictionary or None implies using all default values for component parameters. Defaults to None.
- **custom_name** (*str*) – Custom name for the pipeline. Defaults to None.
- **random_seed** (*int*) – Seed for the random number generator. Defaults to 0.

evalml.pipelines.RegressionPipeline.can_tune_threshold_with_objective

RegressionPipeline.**can_tune_threshold_with_objective**(*objective*)

Determine whether the threshold of a binary classification pipeline can be tuned.

Parameters

- **pipeline** (PipelineBase) – Binary classification pipeline.
- **objective** – Primary AutoMLSearch objective.

evalml.pipelines.RegressionPipeline.clone

RegressionPipeline.**clone**()

Constructs a new pipeline with the same components, parameters, and random state.

Returns A new instance of this pipeline with identical components, parameters, and random state.

evalml.pipelines.RegressionPipeline.compute_estimator_features

RegressionPipeline.**compute_estimator_features**(*X*, *y=None*)

Transforms the data by applying all pre-processing components.

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

Returns New transformed features.

Return type pd.DataFrame

evalml.pipelines.RegressionPipeline.create_objectives

static RegressionPipeline.**create_objectives**(*objectives*)

evalml.pipelines.RegressionPipeline.describe

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

Outputs pipeline details including component parameters

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

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

Return type dict

evalml.pipelines.RegressionPipeline.fit

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

Build a regression model.

Parameters

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

Returns self

evalml.pipelines.RegressionPipeline.get_component

RegressionPipeline.**get_component**(*name*)

Returns component by name

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

Returns Component to return

Return type Component

evalml.pipelines.RegressionPipeline.graph

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

Generate an image representing the pipeline graph

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

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

Return type graphviz.Digraph

evalml.pipelines.RegressionPipeline.graph_feature_importance

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

Generate a bar graph of the pipeline's feature importance

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

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

evalml.pipelines.RegressionPipeline.inverse_transform

RegressionPipeline.**inverse_transform**(*y*)

Apply component inverse_transform methods to estimator predictions in reverse order.

Components that implement inverse_transform are PolynomialDetrender, LabelEncoder (tbd).

Parameters **y** (*pd.Series*) – Final component features

evalml.pipelines.RegressionPipeline.load

static RegressionPipeline.**load**(*file_path*)

Loads pipeline at file path

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

Returns PipelineBase object

evalml.pipelines.RegressionPipeline.new

RegressionPipeline.**new**(*parameters*, *random_seed=0*)

Constructs a new instance of the pipeline with the same component graph but with a different set of parameters.
Not to be confused with python's __new__ method.

Parameters

- **parameters** (*dict*) – Dictionary with component names as keys and dictionary of that component's parameters as values. An empty dictionary or None implies using all default values for component parameters. Defaults to None.
- **random_seed** (*int*) – Seed for the random number generator. Defaults to 0.

Returns A new instance of this pipeline with identical components.

evalml.pipelines.RegressionPipeline.predict

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

Make predictions using selected features.

Parameters

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

Returns Predicted values.

Return type pd.Series

evalml.pipelines.RegressionPipeline.save

RegressionPipeline.**save**(*file_path*, *pickle_protocol=5*)

Saves pipeline at file path

Parameters

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

Returns None

evalml.pipelines.RegressionPipeline.score

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

Evaluate model performance on current and additional objectives

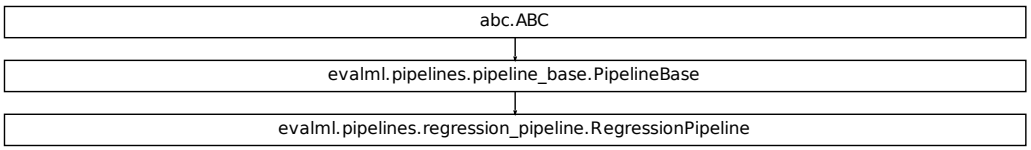
Parameters

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

Returns Ordered dictionary of objective scores

Return type dict

Class Inheritance



evalml.pipelines.TimeSeriesClassificationPipeline

class evalml.pipelines.**TimeSeriesClassificationPipeline**(*component_graph*, *parameters=None*, *custom_name=None*, *random_seed=0*)

Pipeline base class for time series classification problems.

Instance attributes

classes_	Gets the class names for the problem.
custom_name	Custom name of the pipeline.
feature_importance	Importance associated with each feature.
linearized_component_graph	this is not guaranteed to be in proper component computation order
model_family	Returns model family of this pipeline template
name	Name of the pipeline.
parameters	Parameter dictionary for this pipeline
problem_type	
summary	A short summary of the pipeline structure, describing the list of components used.

Methods:

__init__	Machine learning pipeline for time series classification problems made out of transformers and a classifier.
can_tune_threshold_with_objective	Determine whether the threshold of a binary classification pipeline can be tuned.
clone	Constructs a new pipeline with the same components, parameters, and random state.
compute_estimator_features	Transforms the data by applying all pre-processing components.
create_objectives	
describe	Outputs pipeline details including component parameters
fit	Fit a time series classification pipeline.
get_component	Returns component by name
graph	Generate an image representing the pipeline graph
graph_feature_importance	Generate a bar graph of the pipeline's feature importance
inverse_transform	Apply component inverse_transform methods to estimator predictions in reverse order.
load	Loads pipeline at file path
new	Constructs a new instance of the pipeline with the same component graph but with a different set of parameters.
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.TimeSeriesClassificationPipeline.__init__

TimeSeriesClassificationPipeline.**__init__**(*component_graph*, *parameters=None*, *custom_name=None*, *random_seed=0*)

Machine learning pipeline for time series classification problems made out of transformers and a classifier.

Parameters

- **component_graph** (*list or dict*) – List of components in order. Accepts strings or ComponentBase subclasses in the list. Note that when duplicate components are specified in a list, the duplicate component names will be modified with the component's index in the list. For example, the component graph [Imputer, One Hot Encoder, Imputer, Logistic Regression Classifier] will have names ["Imputer", "One Hot Encoder", "Imputer_2", "Logistic Regression Classifier"]
- **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. Pipeline-level parameters such as date_index, gap, and max_delay must be specified with the "pipeline" key. For example: Pipeline(parameters={"pipeline": {"date_index": "Date", "max_delay": 4, "gap": 2}}).
- **random_seed** (*int*) – Seed for the random number generator. Defaults to 0.

evalml.pipelines.TimeSeriesClassificationPipeline.can_tune_threshold_with_objective

TimeSeriesClassificationPipeline.**can_tune_threshold_with_objective**(*objective*)

Determine whether the threshold of a binary classification pipeline can be tuned.

Parameters

- **pipeline** (PipelineBase) – Binary classification pipeline.
- **objective** – Primary AutoMLSearch objective.

evalml.pipelines.TimeSeriesClassificationPipeline.clone

TimeSeriesClassificationPipeline.**clone**()

Constructs a new pipeline with the same components, parameters, and random state.

Returns A new instance of this pipeline with identical components, parameters, and random state.

evalml.pipelines.TimeSeriesClassificationPipeline.compute_estimator_features

TimeSeriesClassificationPipeline.**compute_estimator_features**(*X*, *y=None*)

Transforms the data by applying all pre-processing components.

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

Returns New transformed features.

Return type pd.DataFrame

evalml.pipelines.TimeSeriesClassificationPipeline.create_objectives

static TimeSeriesClassificationPipeline.**create_objectives**(*objectives*)

evalml.pipelines.TimeSeriesClassificationPipeline.describe

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

Outputs pipeline details including component parameters

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

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

Return type dict

evalml.pipelines.TimeSeriesClassificationPipeline.fit

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

Fit a time series classification pipeline.

Parameters

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

Returns self

evalml.pipelines.TimeSeriesClassificationPipeline.get_component

TimeSeriesClassificationPipeline.**get_component**(*name*)

Returns component by name

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

Returns Component to return

Return type Component

evalml.pipelines.TimeSeriesClassificationPipeline.graph

TimeSeriesClassificationPipeline.**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.TimeSeriesClassificationPipeline.graph_feature_importance

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

Generate a bar graph of the pipeline's feature importance

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

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

evalml.pipelines.TimeSeriesClassificationPipeline.inverse_transform

TimeSeriesClassificationPipeline.**inverse_transform**(*y*)

Apply component inverse_transform methods to estimator predictions in reverse order.

Components that implement inverse_transform are PolynomialDetrender, LabelEncoder (tbd).

Parameters **y** (*pd.Series*) – Final component features

evalml.pipelines.TimeSeriesClassificationPipeline.load

static TimeSeriesClassificationPipeline.**load**(*file_path*)

Loads pipeline at file path

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

Returns PipelineBase object

evalml.pipelines.TimeSeriesClassificationPipeline.new

TimeSeriesClassificationPipeline.**new**(*parameters*, *random_seed=0*)

Constructs a new instance of the pipeline with the same component graph but with a different set of parameters.

Not to be confused with python's __new__ method.

Parameters

- **parameters** (*dict*) – Dictionary with component names as keys and dictionary of that component's parameters as values. An empty dictionary or None implies using all default values for component parameters. Defaults to None.
- **random_seed** (*int*) – Seed for the random number generator. Defaults to 0.

Returns A new instance of this pipeline with identical components.

evalml.pipelines.TimeSeriesClassificationPipeline.predict

TimeSeriesClassificationPipeline.**predict**(*X*, *y=None*, *objective=None*)

Make predictions using selected features.

Parameters

- **X** (*pd.DataFrame, or np.ndarray*) – Data of shape [n_samples, n_features]
- **y** (*pd.Series, np.ndarray, None*) – The target training targets of length [n_samples]
- **objective** (*Object or string*) – The objective to use to make predictions

Returns Predicted values.

Return type pd.Series

evalml.pipelines.TimeSeriesClassificationPipeline.predict_proba

TimeSeriesClassificationPipeline.**predict_proba**(*X*, *y=None*)

Make probability estimates for labels.

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

Returns Probability estimates

Return type pd.DataFrame

evalml.pipelines.TimeSeriesClassificationPipeline.save

TimeSeriesClassificationPipeline.**save**(*file_path*, *pickle_protocol=5*)

Saves pipeline at file path

Parameters

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

Returns None

evalml.pipelines.TimeSeriesClassificationPipeline.score

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

Evaluate model performance on current and additional objectives.

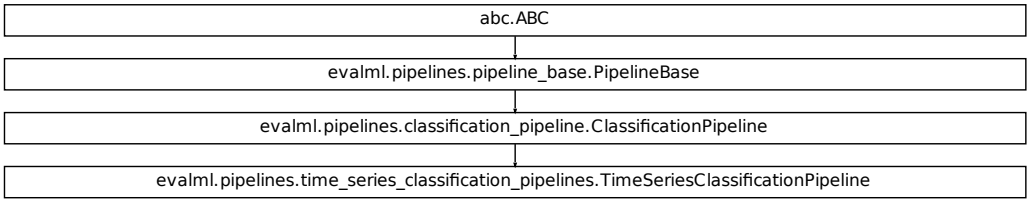
Parameters

- **X** (*pd.DataFrame or np.ndarray*) – 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

Class Inheritance



evalml.pipelines.TimeSeriesBinaryClassificationPipeline

class evalml.pipelines.**TimeSeriesBinaryClassificationPipeline**(*component_graph*, *parameters=None*, *custom_name=None*, *random_seed=0*)

Instance attributes

classes_	Gets the class names for the problem.
custom_name	Custom name of the pipeline.
feature_importance	Importance associated with each feature.
linearized_component_graph	this is not guaranteed to be in proper component computation order
model_family	Returns model family of this pipeline template
name	Name of the pipeline.
parameters	Parameter dictionary for this pipeline
problem_type	
summary	A short summary of the pipeline structure, describing the list of components used.
threshold	Threshold used to make a prediction.

Methods:

__init__	Machine learning pipeline for time series classification problems made out of transformers and a classifier.
can_tune_threshold_with_objective	Determine whether the threshold of a binary classification pipeline can be tuned.
clone	Constructs a new pipeline with the same components, parameters, and random state.
compute_estimator_features	Transforms the data by applying all pre-processing components.
create_objectives	
describe	Outputs pipeline details including component parameters
fit	Fit a time series classification pipeline.
get_component	Returns component by name
graph	Generate an image representing the pipeline graph
graph_feature_importance	Generate a bar graph of the pipeline's feature importance
inverse_transform	Apply component inverse_transform methods to estimator predictions in reverse order.
load	Loads pipeline at file path
new	Constructs a new instance of the pipeline with the same component graph but with a different set of parameters.
optimize_threshold	Optimize the pipeline threshold given the objective to use.
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.TimeSeriesBinaryClassificationPipeline.__init__

TimeSeriesBinaryClassificationPipeline.**__init__**(*component_graph*, *parameters=None*, *custom_name=None*, *random_seed=0*)

Machine learning pipeline for time series classification problems made out of transformers and a classifier.

Parameters

- **component_graph** (*list or dict*) – List of components in order. Accepts strings or ComponentBase subclasses in the list. Note that when duplicate components are specified in a list, the duplicate component names will be modified with the component's index in the list. For example, the component graph [Imputer, One Hot Encoder, Imputer, Logistic Regression Classifier] will have names ["Imputer", "One Hot Encoder", "Imputer_2", "Logistic Regression Classifier"]
- **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. Pipeline-level parameters such as

date_index, gap, and max_delay must be specified with the "pipeline" key. For example: Pipeline(parameters={"pipeline": {"date_index": "Date", "max_delay": 4, "gap": 2}}).

- **random_seed** (*int*) – Seed for the random number generator. Defaults to 0.

evalml.pipelines.TimeSeriesBinaryClassificationPipeline.can_tune_threshold_with_objective

TimeSeriesBinaryClassificationPipeline.**can_tune_threshold_with_objective**(*objective*)

Determine whether the threshold of a binary classification pipeline can be tuned.

Parameters

- **pipeline** (PipelineBase) – Binary classification pipeline.
- **objective** – Primary AutoMLSearch objective.

evalml.pipelines.TimeSeriesBinaryClassificationPipeline.clone

TimeSeriesBinaryClassificationPipeline.**clone**()

Constructs a new pipeline with the same components, parameters, and random state.

Returns A new instance of this pipeline with identical components, parameters, and random state.

evalml.pipelines.TimeSeriesBinaryClassificationPipeline.compute_estimator_features

TimeSeriesBinaryClassificationPipeline.**compute_estimator_features**(*X*, *y=None*)

Transforms the data by applying all pre-processing components.

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

Returns New transformed features.

Return type pd.DataFrame

evalml.pipelines.TimeSeriesBinaryClassificationPipeline.create_objectives

static TimeSeriesBinaryClassificationPipeline.**create_objectives**(*objectives*)

evalml.pipelines.TimeSeriesBinaryClassificationPipeline.describe

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

Outputs pipeline details including component parameters

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

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

Return type dict

evalml.pipelines.TimeSeriesBinaryClassificationPipeline.fit

TimeSeriesBinaryClassificationPipeline.**fit**(*X*, *y*)
Fit a time series classification pipeline.

Parameters

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

Returns self

evalml.pipelines.TimeSeriesBinaryClassificationPipeline.get_component

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

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

Returns Component to return

Return type Component

evalml.pipelines.TimeSeriesBinaryClassificationPipeline.graph

TimeSeriesBinaryClassificationPipeline.**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.TimeSeriesBinaryClassificationPipeline.graph_feature_importance

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

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

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

evalml.pipelines.TimeSeriesBinaryClassificationPipeline.inverse_transform

TimeSeriesBinaryClassificationPipeline.**inverse_transform**(*y*)

Apply component inverse_transform methods to estimator predictions in reverse order.

Components that implement inverse_transform are PolynomialDetrender, LabelEncoder (tbd).

Parameters **y** (*pd.Series*) – Final component features

evalml.pipelines.TimeSeriesBinaryClassificationPipeline.load

static TimeSeriesBinaryClassificationPipeline.**load**(*file_path*)

Loads pipeline at file path

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

Returns PipelineBase object

evalml.pipelines.TimeSeriesBinaryClassificationPipeline.new

TimeSeriesBinaryClassificationPipeline.**new**(*parameters*, *random_seed=0*)

Constructs a new instance of the pipeline with the same component graph but with a different set of parameters.
Not to be confused with python's __new__ method.

Parameters

- **parameters** (*dict*) – Dictionary with component names as keys and dictionary of that component's parameters as values. An empty dictionary or None implies using all default values for component parameters. Defaults to None.
- **random_seed** (*int*) – Seed for the random number generator. Defaults to 0.

Returns A new instance of this pipeline with identical components.

evalml.pipelines.TimeSeriesBinaryClassificationPipeline.optimize_threshold

TimeSeriesBinaryClassificationPipeline.**optimize_threshold**(*X*, *y*, *y_pred_proba*, *objective*)

Optimize the pipeline threshold given the objective to use. Only used for binary problems with objectives whose thresholds can be tuned.

Parameters

- **X** (*pd.DataFrame*) – Input features
- **y** (*pd.Series*) – Input target values
- **y_pred_proba** (*pd.Series*) – The predicted probabilities of the target outputted by the pipeline
- **objective** (ObjectiveBase) – The objective to threshold with. Must have a tunable threshold.

evalml.pipelines.TimeSeriesBinaryClassificationPipeline.predict

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

Parameters

- **X** (*pd.DataFrame, or np.ndarray*) – Data of shape [n_samples, n_features]
- **y** (*pd.Series, np.ndarray, None*) – The target training targets of length [n_samples]
- **objective** (*Object or string*) – The objective to use to make predictions

Returns Predicted values.

Return type pd.Series

evalml.pipelines.TimeSeriesBinaryClassificationPipeline.predict_proba

TimeSeriesBinaryClassificationPipeline.**predict_proba**(*X*, *y=None*)
Make probability estimates for labels.

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

Returns Probability estimates

Return type pd.DataFrame

evalml.pipelines.TimeSeriesBinaryClassificationPipeline.save

TimeSeriesBinaryClassificationPipeline.**save**(*file_path*, *pickle_protocol=5*)
Saves pipeline at file path

Parameters

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

Returns None

evalml.pipelines.TimeSeriesBinaryClassificationPipeline.score

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

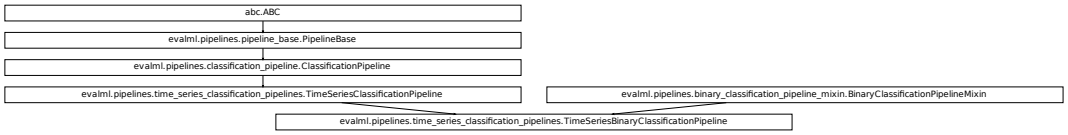
Parameters

- **X** (*pd.DataFrame or np.ndarray*) – 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

Class Inheritance



evalml.pipelines.TimeSeriesMulticlassClassificationPipeline

class evalml.pipelines.**TimeSeriesMulticlassClassificationPipeline**(*component_graph*, *parameters=None*, *custom_name=None*, *random_seed=0*)

Instance attributes

classes_	Gets the class names for the problem.
custom_name	Custom name of the pipeline.
feature_importance	Importance associated with each feature.
linearized_component_graph	this is not guaranteed to be in proper component computation order
model_family	Returns model family of this pipeline template
name	Name of the pipeline.
parameters	Parameter dictionary for this pipeline
problem_type	
summary	A short summary of the pipeline structure, describing the list of components used.

Methods:

__init__	Machine learning pipeline for time series classification problems made out of transformers and a classifier.
can_tune_threshold_with_objective	Determine whether the threshold of a binary classification pipeline can be tuned.
clone	Constructs a new pipeline with the same components, parameters, and random state.

continues on next page

Table 30 – continued from previous page

compute_estimator_features	Transforms the data by applying all pre-processing components.
create_objectives	
describe	Outputs pipeline details including component parameters
fit	Fit a time series classification pipeline.
get_component	Returns component by name
graph	Generate an image representing the pipeline graph
graph_feature_importance	Generate a bar graph of the pipeline's feature importance
inverse_transform	Apply component inverse_transform methods to estimator predictions in reverse order.
load	Loads pipeline at file path
new	Constructs a new instance of the pipeline with the same component graph but with a different set of parameters.
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.TimeSeriesMulticlassClassificationPipeline.__init__

TimeSeriesMulticlassClassificationPipeline.**__init__**(*component_graph*, *parameters=None*, *custom_name=None*, *random_seed=0*)

Machine learning pipeline for time series classification problems made out of transformers and a classifier.

Parameters

- **component_graph** (*list or dict*) – List of components in order. Accepts strings or ComponentBase subclasses in the list. Note that when duplicate components are specified in a list, the duplicate component names will be modified with the component's index in the list. For example, the component graph [Imputer, One Hot Encoder, Imputer, Logistic Regression Classifier] will have names ["Imputer", "One Hot Encoder", "Imputer_2", "Logistic Regression Classifier"]
- **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. Pipeline-level parameters such as date_index, gap, and max_delay must be specified with the "pipeline" key. For example: Pipeline(parameters={"pipeline": {"date_index": "Date", "max_delay": 4, "gap": 2}}).
- **random_seed** (*int*) – Seed for the random number generator. Defaults to 0.

evalml.pipelines.TimeSeriesMulticlassClassificationPipeline.can_tune_threshold_with_objective

TimeSeriesMulticlassClassificationPipeline.**can_tune_threshold_with_objective**(*objective*)

Determine whether the threshold of a binary classification pipeline can be tuned.

Parameters

- **pipeline** (PipelineBase) – Binary classification pipeline.
- **objective** – Primary AutoMLSearch objective.

evalml.pipelines.TimeSeriesMulticlassClassificationPipeline.clone

TimeSeriesMulticlassClassificationPipeline.**clone**()

Constructs a new pipeline with the same components, parameters, and random state.

Returns A new instance of this pipeline with identical components, parameters, and random state.

evalml.pipelines.TimeSeriesMulticlassClassificationPipeline.compute_estimator_features

TimeSeriesMulticlassClassificationPipeline.**compute_estimator_features**(*X*, *y=None*)

Transforms the data by applying all pre-processing components.

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

Returns New transformed features.

Return type pd.DataFrame

evalml.pipelines.TimeSeriesMulticlassClassificationPipeline.create_objectives

static TimeSeriesMulticlassClassificationPipeline.**create_objectives**(*objectives*)

evalml.pipelines.TimeSeriesMulticlassClassificationPipeline.describe

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

Outputs pipeline details including component parameters

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

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

Return type dict

evalml.pipelines.TimeSeriesMulticlassClassificationPipeline.fit

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

Fit a time series classification pipeline.

Parameters

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

Returns self

evalml.pipelines.TimeSeriesMulticlassClassificationPipeline.get_component

TimeSeriesMulticlassClassificationPipeline.**get_component**(*name*)

Returns component by name

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

Returns Component to return

Return type Component

evalml.pipelines.TimeSeriesMulticlassClassificationPipeline.graph

TimeSeriesMulticlassClassificationPipeline.**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.TimeSeriesMulticlassClassificationPipeline.graph_feature_importance

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

Generate a bar graph of the pipeline's feature importance

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

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

evalml.pipelines.TimeSeriesMulticlassClassificationPipeline.inverse_transform

TimeSeriesMulticlassClassificationPipeline.**inverse_transform**(*y*)

Apply component inverse_transform methods to estimator predictions in reverse order.

Components that implement inverse_transform are PolynomialDetrender, LabelEncoder (tbd).

Parameters **y** (*pd.Series*) – Final component features

evalml.pipelines.TimeSeriesMulticlassClassificationPipeline.load

static TimeSeriesMulticlassClassificationPipeline.**load**(*file_path*)

Loads pipeline at file path

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

Returns PipelineBase object

evalml.pipelines.TimeSeriesMulticlassClassificationPipeline.new

TimeSeriesMulticlassClassificationPipeline.**new**(*parameters*, *random_seed=0*)

Constructs a new instance of the pipeline with the same component graph but with a different set of parameters.

Not to be confused with python's __new__ method.

Parameters

- **parameters** (*dict*) – Dictionary with component names as keys and dictionary of that component's parameters as values. An empty dictionary or None implies using all default values for component parameters. Defaults to None.
- **random_seed** (*int*) – Seed for the random number generator. Defaults to 0.

Returns A new instance of this pipeline with identical components.

evalml.pipelines.TimeSeriesMulticlassClassificationPipeline.predict

TimeSeriesMulticlassClassificationPipeline.**predict**(*X*, *y=None*, *objective=None*)

Make predictions using selected features.

Parameters

- **X** (*pd.DataFrame, or np.ndarray*) – Data of shape [n_samples, n_features]
- **y** (*pd.Series, np.ndarray, None*) – The target training targets of length [n_samples]
- **objective** (*Object or string*) – The objective to use to make predictions

Returns Predicted values.

Return type pd.Series

evalml.pipelines.TimeSeriesMulticlassClassificationPipeline.predict_proba

TimeSeriesMulticlassClassificationPipeline.**predict_proba**(*X*, *y=None*)
Make probability estimates for labels.

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

Returns Probability estimates

Return type pd.DataFrame

evalml.pipelines.TimeSeriesMulticlassClassificationPipeline.save

TimeSeriesMulticlassClassificationPipeline.**save**(*file_path*, *pickle_protocol=5*)
Saves pipeline at file path

Parameters

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

Returns None

evalml.pipelines.TimeSeriesMulticlassClassificationPipeline.score

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

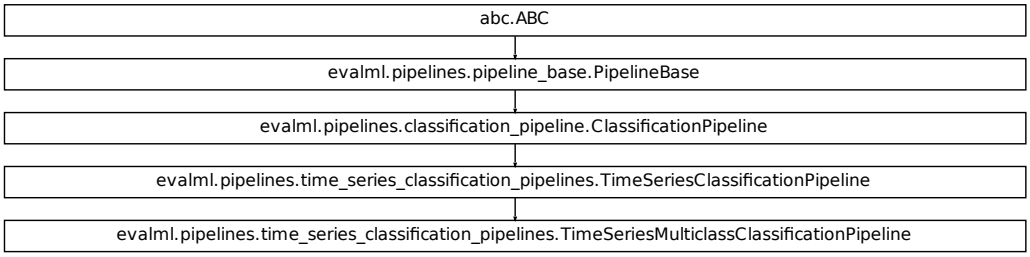
Parameters

- **X** (*pd.DataFrame or np.ndarray*) – 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

Class Inheritance



evalml.pipelines.TimeSeriesRegressionPipeline

class evalml.pipelines.**TimeSeriesRegressionPipeline**(*component_graph*, *parameters=None*, *custom_name=None*, *random_seed=0*)

Pipeline base class for time series regression problems.

Instance attributes

custom_name	Custom name of the pipeline.
feature_importance	Importance associated with each feature.
linearized_component_graph	this is not guaranteed to be in proper component computation order
model_family	Returns model family of this pipeline template
name	Name of the pipeline.
parameters	Parameter dictionary for this pipeline
problem_type	
summary	A short summary of the pipeline structure, describing the list of components used.

Methods:

__init__	Machine learning pipeline for time series regression problems made out of transformers and a classifier.
can_tune_threshold_with_objective	Determine whether the threshold of a binary classification pipeline can be tuned.
clone	Constructs a new pipeline with the same components, parameters, and random state.

continues on next page

Table 32 – continued from previous page

compute_estimator_features	Transforms the data by applying all pre-processing components.
create_objectives	
describe	Outputs pipeline details including component parameters
fit	Fit a time series regression pipeline.
get_component	Returns component by name
graph	Generate an image representing the pipeline graph
graph_feature_importance	Generate a bar graph of the pipeline's feature importance
inverse_transform	Apply component inverse_transform methods to estimator predictions in reverse order.
load	Loads pipeline at file path
new	Constructs a new instance of the pipeline with the same component graph but with a different set of parameters.
predict	Make predictions using selected features.
save	Saves pipeline at file path
score	Evaluate model performance on current and additional objectives.

evalml.pipelines.TimeSeriesRegressionPipeline.__init__

TimeSeriesRegressionPipeline.**__init__**(*component_graph*, *parameters=None*, *custom_name=None*, *random_seed=0*)

Machine learning pipeline for time series regression problems made out of transformers and a classifier.

Parameters

- **component_graph** (*list or dict*) – List of components in order. Accepts strings or ComponentBase subclasses in the list. Note that when duplicate components are specified in a list, the duplicate component names will be modified with the component's index in the list. For example, the component graph [Imputer, One Hot Encoder, Imputer, Logistic Regression Classifier] will have names ["Imputer", "One Hot Encoder", "Imputer_2", "Logistic Regression Classifier"]
- **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. Pipeline-level parameters such as date_index, gap, and max_delay must be specified with the "pipeline" key. For example: Pipeline(parameters={"pipeline": {"date_index": "Date", "max_delay": 4, "gap": 2}}).
- **random_seed** (*int*) – Seed for the random number generator. Defaults to 0.

evalml.pipelines.TimeSeriesRegressionPipeline.can_tune_threshold_with_objective

TimeSeriesRegressionPipeline.**can_tune_threshold_with_objective**(*objective*)

Determine whether the threshold of a binary classification pipeline can be tuned.

Parameters

- **pipeline** (PipelineBase) – Binary classification pipeline.
- **objective** – Primary AutoMLSearch objective.

evalml.pipelines.TimeSeriesRegressionPipeline.clone

TimeSeriesRegressionPipeline.**clone**()

Constructs a new pipeline with the same components, parameters, and random state.

Returns A new instance of this pipeline with identical components, parameters, and random state.

evalml.pipelines.TimeSeriesRegressionPipeline.compute_estimator_features

TimeSeriesRegressionPipeline.**compute_estimator_features**(*X*, *y=None*)

Transforms the data by applying all pre-processing components.

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

Returns New transformed features.

Return type pd.DataFrame

evalml.pipelines.TimeSeriesRegressionPipeline.create_objectives

static TimeSeriesRegressionPipeline.**create_objectives**(*objectives*)

evalml.pipelines.TimeSeriesRegressionPipeline.describe

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

Outputs pipeline details including component parameters

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

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

Return type dict

evalml.pipelines.TimeSeriesRegressionPipeline.fit

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

Fit a time series regression pipeline.

Parameters

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

Returns self

evalml.pipelines.TimeSeriesRegressionPipeline.get_component

TimeSeriesRegressionPipeline.**get_component**(*name*)

Returns component by name

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

Returns Component to return

Return type Component

evalml.pipelines.TimeSeriesRegressionPipeline.graph

TimeSeriesRegressionPipeline.**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.TimeSeriesRegressionPipeline.graph_feature_importance

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

Generate a bar graph of the pipeline's feature importance

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

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

evalml.pipelines.TimeSeriesRegressionPipeline.inverse_transform

TimeSeriesRegressionPipeline.**inverse_transform**(*y*)

Apply component inverse_transform methods to estimator predictions in reverse order.

Components that implement inverse_transform are PolynomialDetrender, LabelEncoder (tbd).

Parameters **y** (*pd.Series*) – Final component features

evalml.pipelines.TimeSeriesRegressionPipeline.load

static TimeSeriesRegressionPipeline.**load**(*file_path*)

Loads pipeline at file path

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

Returns PipelineBase object

evalml.pipelines.TimeSeriesRegressionPipeline.new

TimeSeriesRegressionPipeline.**new**(*parameters*, *random_seed=0*)

Constructs a new instance of the pipeline with the same component graph but with a different set of parameters.
Not to be confused with python's __new__ method.

Parameters

- **parameters** (*dict*) – Dictionary with component names as keys and dictionary of that component's parameters as values. An empty dictionary or None implies using all default values for component parameters. Defaults to None.
- **random_seed** (*int*) – Seed for the random number generator. Defaults to 0.

Returns A new instance of this pipeline with identical components.

evalml.pipelines.TimeSeriesRegressionPipeline.predict

TimeSeriesRegressionPipeline.**predict**(*X*, *y=None*, *objective=None*)

Make predictions using selected features.

Parameters

- **X** (*pd.DataFrame, or np.ndarray*) – Data of shape [n_samples, n_features]
- **y** (*pd.Series, np.ndarray, None*) – The target training targets of length [n_samples]
- **objective** (*Object or string*) – The objective to use to make predictions

Returns Predicted values.

Return type pd.Series

evalml.pipelines.TimeSeriesRegressionPipeline.save

TimeSeriesRegressionPipeline.**save**(*file_path*, *pickle_protocol=5*)

Saves pipeline at file path

Parameters

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

Returns None

evalml.pipelines.TimeSeriesRegressionPipeline.score

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

Evaluate model performance on current and additional objectives.

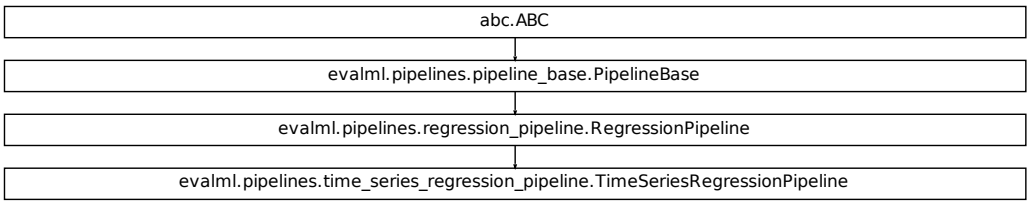
Parameters

- **X** (*pd.DataFrame or np.ndarray*) – 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

Class Inheritance



5.5.2 Pipeline Utils

make_pipeline	Given input data, target data, an estimator class and the problem type,
generate_pipeline_code	Creates and returns a string that contains the Python imports and code required for running the EvalML pipeline.

evalml.pipelines.utils.make_pipeline

evalml.pipelines.utils.**make_pipeline**(*X*, *y*, *estimator*, *problem_type*, *parameters=None*, *sampler_name=None*)

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

Parameters

- **X** (*pd.DataFrame*) – The input data of shape [n_samples, n_features]
- **y** (*pd.Series*) – The target data of length [n_samples]
- **estimator** (*Estimator*) – Estimator for pipeline
- **problem_type** (*ProblemTypes or str*) – Problem type for pipeline to generate
- **parameters** (*dict*) – Dictionary with component names as keys and dictionary of that component's parameters as values. An empty dictionary or None implies using all default values for component parameters.
- **sampler_name** – The name of the sampler component to add to the pipeline. Only used in classification problems. Defaults to None

evalml.pipelines.utils.generate_pipeline_code

evalml.pipelines.utils.**generate_pipeline_code**(*element*)

Creates and returns a string that contains the Python imports and code required for running the EvalML pipeline.

Parameters **element** (*pipeline instance*) – The instance of the pipeline to generate string Python code

Returns String representation of Python code that can be run separately in order to recreate the pipeline instance. Does not include code for custom component implementation.

5.6 Components

5.6.1 Component Base Classes

Components represent a step in a pipeline.

ComponentBase	Base class for all components.

continues on next page

Table 34 – continued from previous page

Transformer	A component that may or may not need fitting that transforms data.
Estimator	A component that fits and predicts given data.

evalml.pipelines.components.ComponentBase

class evalml.pipelines.components.**ComponentBase**(*parameters=None*, *component_obj=None*, *random_seed=0*, ***kwargs*)

Base class for all components.

Methods

__init__	Initialize self.
clone	Constructs a new component with the same parameters and random state.
describe	Describe a component and its parameters
fit	Fits component to data
load	Loads component at file path
save	Saves component at file path

evalml.pipelines.components.ComponentBase.__init__

ComponentBase.**__init__**(*parameters=None*, *component_obj=None*, *random_seed=0*, ***kwargs*)

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

evalml.pipelines.components.ComponentBase.clone

ComponentBase.**clone**()

Constructs a new component with the same parameters and random state.

Returns A new instance of this component with identical parameters and random state.

evalml.pipelines.components.ComponentBase.describe

ComponentBase.**describe**(*print_name=False*, *return_dict=False*)

Describe a component and its parameters

Parameters

- **print_name** (*bool, optional*) – whether to print name of component
- **return_dict** (*bool, optional*) – whether to return description as dictionary in the format {"name": name, "parameters": parameters}

Returns prints and returns dictionary

Return type None or dict

evalml.pipelines.components.ComponentBase.fit

ComponentBase.**fit**(*X*, *y=None*)

Fits component to data

Parameters

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

Returns self

evalml.pipelines.components.ComponentBase.load

static ComponentBase.**load**(*file_path*)

Loads component at file path

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

Returns ComponentBase object

evalml.pipelines.components.ComponentBase.save

ComponentBase.**save**(*file_path*, *pickle_protocol=5*)

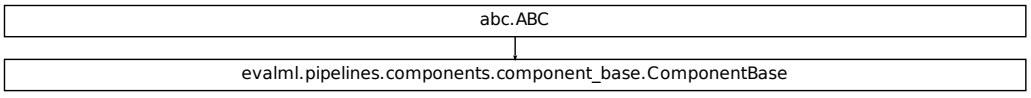
Saves component at file path

Parameters

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

Returns None

Class Inheritance



evalml.pipelines.components.Transformer

class evalml.pipelines.components.**Transformer**(*parameters=None*, *component_obj=None*, *random_seed=0*, ***kwargs*)

A component that may or may not need fitting that transforms data. These components are used before an estimator.

To implement a new Transformer, define your own class which is a subclass of Transformer, including a name and a list of acceptable ranges for any parameters to be tuned during the automl search (hyperparameters). Define an *__init__* method which sets up any necessary state and objects. Make sure your *__init__* only uses standard keyword arguments and calls *super().__init__()* with a parameters dict. You may also override the *fit*, *transform*, *fit_transform* and other methods in this class if appropriate.

To see some examples, check out the definitions of any Transformer component.

Methods

__init__	Initialize self.
clone	Constructs a new component with the same parameters and random state.
describe	Describe a component and its parameters
fit	Fits component to data
fit_transform	Fits on X and transforms X
load	Loads component at file path
save	Saves component at file path
transform	Transforms data X.

evalml.pipelines.components.Transformer.__init__

Transformer.**__init__**(*parameters=None*, *component_obj=None*, *random_seed=0*, ***kwargs*)

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

evalml.pipelines.components.Transformer.clone

Transformer.**clone**()

Constructs a new component with the same parameters and random state.

Returns A new instance of this component with identical parameters and random state.

evalml.pipelines.components.Transformer.describe

Transformer.**describe**(*print_name=False*, *return_dict=False*)

Describe a component and its parameters

Parameters

- **print_name** (*bool, optional*) – whether to print name of component
- **return_dict** (*bool, optional*) – whether to return description as dictionary in the format {"name": name, "parameters": parameters}

Returns prints and returns dictionary

Return type None or dict

evalml.pipelines.components.Transformer.fit

Transformer.**fit**(*X*, *y=None*)

Fits component to data

Parameters

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

Returns self

evalml.pipelines.components.Transformer.fit_transform

Transformer.**fit_transform**(*X*, *y=None*)

Fits on X and transforms X

Parameters

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

Returns Transformed X

Return type pd.DataFrame

evalml.pipelines.components.Transformer.load

static Transformer.**load**(*file_path*)

Loads component at file path

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

Returns ComponentBase object

evalml.pipelines.components.Transformer.save

Transformer.**save**(*file_path*, *pickle_protocol=5*)

Saves component at file path

Parameters

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

Returns None

evalml.pipelines.components.Transformer.transform

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

Transforms data X.

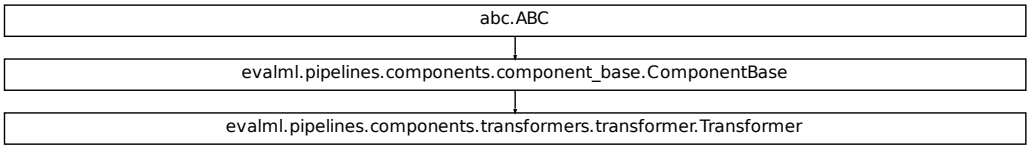
Parameters

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

Returns Transformed X

Return type pd.DataFrame

Class Inheritance



evalml.pipelines.components.Estimator

class evalml.pipelines.components.**Estimator**(*parameters=None*, *component_obj=None*, *random_seed=0*, ***kwargs*)

A component that fits and predicts given data.

To implement a new Estimator, define your own class which is a subclass of Estimator, including a name and a list of acceptable ranges for any parameters to be tuned during the automl search (hyperparameters). Define an *__init__* method which sets up any necessary state and objects. Make sure your *__init__* only uses standard keyword arguments and calls *super().__init__()* with a parameters dict. You may also override the *fit*, *transform*, *fit_transform* and other methods in this class if appropriate.

To see some examples, check out the definitions of any Estimator component.

Methods

__init__	Initialize self.
clone	Constructs a new component with the same parameters and random state.
describe	Describe a component and its parameters
fit	Fits component to data
load	Loads component at file path
predict	Make predictions using selected features.

continues on next page

Table 37 – continued from previous page

predict_proba	Make probability estimates for labels.
save	Saves component at file path

evalml.pipelines.components.Estimator.__init__

Estimator.**__init__**(*parameters=None*, *component_obj=None*, *random_seed=0*, ***kwargs*)
Initialize self. See help(type(self)) for accurate signature.

evalml.pipelines.components.Estimator.clone

Estimator.**clone**()
Constructs a new component with the same parameters and random state.

Returns A new instance of this component with identical parameters and random state.

evalml.pipelines.components.Estimator.describe

Estimator.**describe**(*print_name=False*, *return_dict=False*)
Describe a component and its parameters

Parameters

- **print_name** (*bool, optional*) – whether to print name of component
- **return_dict** (*bool, optional*) – whether to return description as dictionary in the format {"name": name, "parameters": parameters}

Returns prints and returns dictionary

Return type None or dict

evalml.pipelines.components.Estimator.fit

Estimator.**fit**(*X*, *y=None*)
Fits component to data

Parameters

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

Returns self

evalml.pipelines.components.Estimator.load

static Estimator.**load**(*file_path*)
Loads component at file path

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

Returns ComponentBase object

evalml.pipelines.components.Estimator.predict

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

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

Returns Predicted values

Return type pd.Series

evalml.pipelines.components.Estimator.predict_proba

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

Parameters **X** (*pd.DataFrame, or np.ndarray*) – Features

Returns Probability estimates

Return type pd.Series

evalml.pipelines.components.Estimator.save

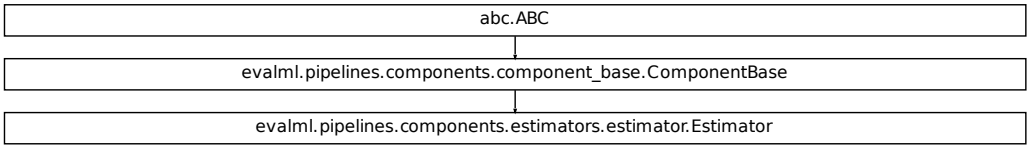
Estimator.**save**(*file_path*, *pickle_protocol=5*)
Saves component at file path

Parameters

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

Returns None

Class Inheritance



5.6.2 Component Utils

allowed_model_families	List the model types allowed for a particular problem type.
get_estimators	Returns the estimators allowed for a particular problem type.
generate_component_code	Creates and returns a string that contains the Python imports and code required for running the EvalML component.

evalml.pipelines.components.utils.allowed_model_families

evalml.pipelines.components.utils.**allowed_model_families**(*problem_type*)

List the model types allowed for a particular problem type.

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

Returns a list of model families

Return type list[*ModelFamily*]

evalml.pipelines.components.utils.get_estimators

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

Returns the estimators allowed for a particular problem type.

Can also optionally filter by a list of model types.

Parameters

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

Returns a list of estimator subclasses

Return type list[class]

evalml.pipelines.components.utils.generate_component_code

evalml.pipelines.components.utils.**generate_component_code**(*element*)

Creates and returns a string that contains the Python imports and code required for running the EvalML component.

Parameters **element** (*component instance*) – The instance of the component to generate string Python code for

Returns String representation of Python code that can be run separately in order to recreate the component instance. Does not include code for custom component implementation.

5.6.3 Transformers

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

DropColumns	Drops specified columns in input data.
SelectColumns	Selects specified columns in input data.
OneHotEncoder	One-hot encoder to encode non-numeric data.
TargetEncoder	Target encoder to encode categorical data
PerColumnImputer	Imputes missing data according to a specified imputation strategy per column
Imputer	Imputes missing data according to a specified imputation strategy.
SimpleImputer	Imputes missing data according to a specified imputation strategy.
StandardScaler	Standardize features: removes mean and scales to unit variance.
RFRegressorSelectFromModel	Selects top features based on importance weights using a Random Forest regressor.
RFClassifierSelectFromModel	Selects top features based on importance weights using a Random Forest classifier.
DropNullColumns	Transformer to drop features whose percentage of NaN values exceeds a specified threshold
DateTimeFeaturizer	Transformer that can automatically featurize DateTime columns.
TextFeaturizer	Transformer that can automatically featurize text columns.
DelayedFeatureTransformer	Transformer that delayes input features and target variable for time series problems.
DFSTransformer	Featuretools DFS component that generates features for pd.DataFrames
PolynomialDetrender	Removes trends from time series by fitting a polynomial to the data.
Undersampler	Random undersampler component.
SMOTESampler	SMOTE Oversampler component.
SMOTENCSampler	SMOTENC Oversampler component.
SMOTENSampler	SMOTEN Oversampler component.

evalml.pipelines.components.DropColumns

class evalml.pipelines.components.**DropColumns**(*columns=None*, *random_seed=0*, ***kwargs*)

Drops specified columns in input data.

name = 'Drop Columns Transformer'

model_family = 'none'

hyperparameter_ranges = {}

default_parameters = {'columns': None}

Instance attributes

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

Methods:

__init__	Initalizes an transformer that drops specified columns in input data.
clone	Constructs a new component with the same parameters and random state.
describe	Describe a component and its parameters
fit	Fits the transformer by checking if column names are present in the dataset.
fit_transform	Fits on X and transforms X
load	Loads component at file path
save	Saves component at file path
transform	Transforms data X by dropping columns.

evalml.pipelines.components.DropColumns.__init__

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

Initalizes an transformer that drops specified columns in input data.

Parameters **columns** (*list (string)*) – List of column names, used to determine which columns to drop.

evalml.pipelines.components.DropColumns.clone

DropColumns.**clone**()

Constructs a new component with the same parameters and random state.

Returns A new instance of this component with identical parameters and random state.

evalml.pipelines.components.DropColumns.describe

DropColumns.**describe**(*print_name=False*, *return_dict=False*)

Describe a component and its parameters

Parameters

- **print_name** (*bool, optional*) – whether to print name of component
- **return_dict** (*bool, optional*) – whether to return description as dictionary in the format {"name": name, "parameters": parameters}

Returns prints and returns dictionary

Return type None or dict

evalml.pipelines.components.DropColumns.fit

DropColumns.**fit**(*X*, *y=None*)

Fits the transformer by checking if column names are present in the dataset.

Parameters

- **X** (*pd.DataFrame*) – Data to check.
- **y** (*pd.Series, optional*) – Targets.

Returns self

evalml.pipelines.components.DropColumns.fit_transform

DropColumns.**fit_transform**(*X*, *y=None*)

Fits on X and transforms X

Parameters

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

Returns Transformed X

Return type pd.DataFrame

evalml.pipelines.components.DropColumns.load

static DropColumns.**load**(*file_path*)
Loads component at file path

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

Returns ComponentBase object

evalml.pipelines.components.DropColumns.save

DropColumns.**save**(*file_path*, *pickle_protocol=5*)
Saves component at file path

Parameters

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

Returns None

evalml.pipelines.components.DropColumns.transform

DropColumns.**transform**(*X*, *y=None*)
Transforms data X by dropping columns.

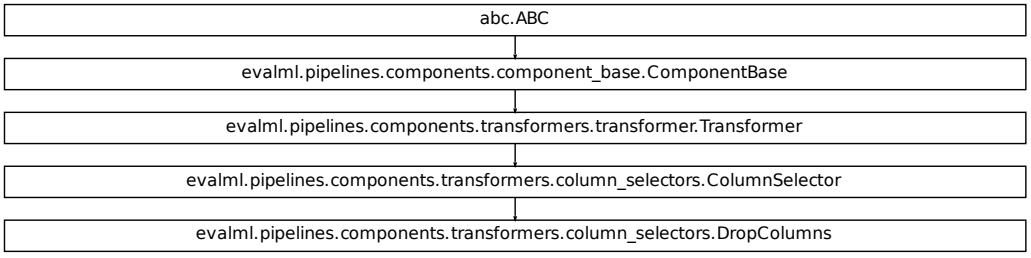
Parameters

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

Returns Transformed X.

Return type pd.DataFrame

Class Inheritance



evalml.pipelines.components.SelectColumns

class evalml.pipelines.components.**SelectColumns**(*columns=None*, *random_seed=0*, ***kwargs*)

Selects specified columns in input data.

name = 'Select Columns Transformer'

model_family = 'none'

hyperparameter_ranges = {}

default_parameters = {'columns': None}

Instance attributes

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

Methods:

__init__	Initalizes an transformer that drops specified columns in input data.
clone	Constructs a new component with the same parameters and random state.
describe	Describe a component and its parameters
fit	Fits the transformer by checking if column names are present in the dataset.
fit_transform	Fits on X and transforms X
load	Loads component at file path
save	Saves component at file path
transform	Transforms data X by selecting columns.

evalml.pipelines.components.SelectColumns.__init__

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

Initalizes an transformer that drops specified columns in input data.

Parameters **columns** (*list (string)*) – List of column names, used to determine which columns to drop.

evalml.pipelines.components.SelectColumns.clone

SelectColumns.**clone**()

Constructs a new component with the same parameters and random state.

Returns A new instance of this component with identical parameters and random state.

evalml.pipelines.components.SelectColumns.describe

SelectColumns.**describe**(*print_name=False*, *return_dict=False*)

Describe a component and its parameters

Parameters

- **print_name** (*bool, optional*) – whether to print name of component
- **return_dict** (*bool, optional*) – whether to return description as dictionary in the format {"name": name, "parameters": parameters}

Returns prints and returns dictionary

Return type None or dict

evalml.pipelines.components.SelectColumns.fit

SelectColumns.**fit**(*X*, *y=None*)

Fits the transformer by checking if column names are present in the dataset.

Parameters

- **X** (*pd.DataFrame*) – Data to check.
- **y** (*pd.Series, optional*) – Targets.

Returns self

evalml.pipelines.components.SelectColumns.fit_transform

SelectColumns.**fit_transform**(*X*, *y=None*)

Fits on X and transforms X

Parameters

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

Returns Transformed X

Return type pd.DataFrame

evalml.pipelines.components.SelectColumns.load

static SelectColumns.**load**(*file_path*)
Loads component at file path

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

Returns ComponentBase object

evalml.pipelines.components.SelectColumns.save

SelectColumns.**save**(*file_path*, *pickle_protocol=5*)
Saves component at file path

Parameters

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

Returns None

evalml.pipelines.components.SelectColumns.transform

SelectColumns.**transform**(*X*, *y=None*)
Transforms data X by selecting columns.

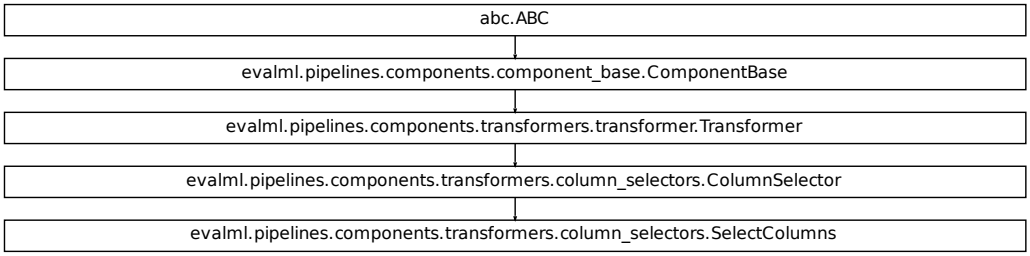
Parameters

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

Returns Transformed X.

Return type pd.DataFrame

Class Inheritance



evalml.pipelines.components.OneHotEncoder

class evalml.pipelines.components.**OneHotEncoder**(*top_n=10*, *features_to_encode=None*, *categories=None*, *drop='if_binary'*, *handle_unknown='ignore'*, *handle_missing='error'*, *random_seed=0*, ***kwargs*)

One-hot encoder to encode non-numeric data.

name = 'One Hot Encoder'

model_family = 'none'

hyperparameter_ranges = {}

default_parameters = {'categories': None, 'drop': 'if_binary', 'features_to_encode': N

Instance attributes

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

Methods:

__init__	Initalizes an transformer that encodes categorical features in a one-hot numeric array."
categories	Returns a list of the unique categories to be encoded for the particular feature, in order.
clone	Constructs a new component with the same parameters and random state.
describe	Describe a component and its parameters
fit	Fits component to data
fit_transform	Fits on X and transforms X
get_feature_names	Return feature names for the categorical features after fitting.
load	Loads component at file path
save	Saves component at file path
transform	One-hot encode the input data.

evalml.pipelines.components.OneHotEncoder.__init__

OneHotEncoder.**__init__**(*top_n=10*, *features_to_encode=None*, *categories=None*, *drop='if_binary'*, *handle_unknown='ignore'*, *handle_missing='error'*, *random_seed=0*, ***kwargs*)

Initalizes an transformer that encodes categorical features in a one-hot numeric array."

Parameters

- **top_n** (*int*) – Number of categories per column to encode. If None, all categories will be encoded. Otherwise, the *n* most frequent will be encoded and all others will be dropped.

Defaults to 10.

- **features_to_encode** (*list[str]*) – List of columns to encode. All other columns will remain untouched. If None, all appropriate columns will be encoded. Defaults to None.
- **categories** (*list*) – A two dimensional list of categories, where *categories[i]* is a list of the categories for the column at index *i*. This can also be *None*, or *"auto"* if *top_n* is not None. Defaults to None.
- **drop** (*string, list*) – Method ("first" or "if_binary") to use to drop one category per feature. Can also be a list specifying which categories to drop for each feature. Defaults to 'if_binary'.
- **handle_unknown** (*string*) – Whether to ignore or error for unknown categories for a feature encountered during *fit* or *transform*. If either *top_n* or *categories* is used to limit the number of categories per column, this must be "ignore". Defaults to "ignore".
- **handle_missing** (*string*) – Options for how to handle missing (NaN) values encountered during *fit* or *transform*. If this is set to "as_category" and NaN values are within the *n* most frequent, "nan" values will be encoded as their own column. If this is set to "error", any missing values encountered will raise an error. Defaults to "error".
- **random_seed** (*int*) – Seed for the random number generator. Defaults to 0.

evalml.pipelines.components.OneHotEncoder.categories

OneHotEncoder.**categories**(*feature_name*)

Returns a list of the unique categories to be encoded for the particular feature, in order.

Parameters **feature_name** (*str*) – the name of any feature provided to one-hot encoder during fit

Returns the unique categories, in the same dtype as they were provided during fit

Return type np.ndarray

evalml.pipelines.components.OneHotEncoder.clone

OneHotEncoder.**clone**()

Constructs a new component with the same parameters and random state.

Returns A new instance of this component with identical parameters and random state.

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

Returns self

evalml.pipelines.components.OneHotEncoder.fit_transform

OneHotEncoder.**fit_transform**(*X*, *y=None*)

Fits on X and transforms X

Parameters

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

Returns Transformed X

Return type pd.DataFrame

evalml.pipelines.components.OneHotEncoder.get_feature_names

OneHotEncoder.**get_feature_names**()

Return feature names for the categorical features after fitting.

Feature names are formatted as {column name}_{category name}. In the event of a duplicate name, an integer will be added at the end of the feature name to distinguish it.

For example, consider a dataframe with a column called "A" and category "x_y" and another column called "A_x" with "y". In this example, the feature names would be "A_x_y" and "A_x_y_1".

Returns The feature names after encoding, provided in the same order as input_features.

Return type np.ndarray

evalml.pipelines.components.OneHotEncoder.load

static OneHotEncoder.**load**(*file_path*)

Loads component at file path

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

Returns ComponentBase object

evalml.pipelines.components.OneHotEncoder.save

OneHotEncoder.**save**(*file_path*, *pickle_protocol=5*)

Saves component at file path

Parameters

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

Returns None

evalml.pipelines.components.OneHotEncoder.transform

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

One-hot encode the input data.

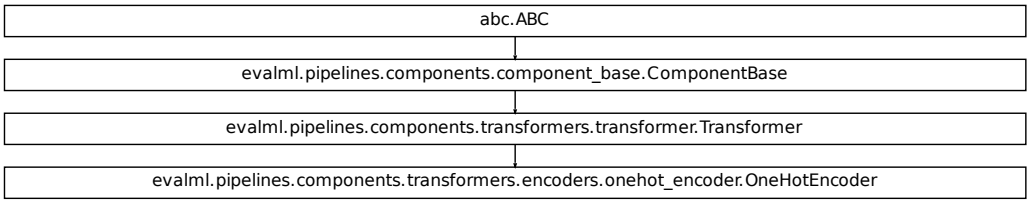
Parameters

- **X** (*pd.DataFrame*) – Features to one-hot encode.
- **y** (*pd.Series*) – Ignored.

Returns Transformed data, where each categorical feature has been encoded into numerical columns using one-hot encoding.

Return type pd.DataFrame

Class Inheritance



evalml.pipelines.components.TargetEncoder

class evalml.pipelines.components.**TargetEncoder**(*cols=None*, *smoothing=1.0*, *handle_unknown='value'*, *handle_missing='value'*, *random_seed=0*, ***kwargs*)

Target encoder to encode categorical data

name = 'Target Encoder'

model_family = 'none'

hyperparameter_ranges = {}

default_parameters = {'cols': None, 'handle_missing': 'value', 'handle_unknown': 'valu

Instance attributes

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

Methods:

__init__	Initializes a transformer that encodes categorical features into target encodings.
clone	Constructs a new component with the same parameters and random state.
describe	Describe a component and its parameters
fit	Fits component to data
fit_transform	Fits on X and transforms X
get_feature_names	Return feature names for the input features after fitting.
load	Loads component at file path
save	Saves component at file path
transform	Transforms data X.

evalml.pipelines.components.TargetEncoder.__init__

TargetEncoder.**__init__**(*cols=None*, *smoothing=1.0*, *handle_unknown='value'*, *handle_missing='value'*, *random_seed=0*, ***kwargs*)

Initializes a transformer that encodes categorical features into target encodings.

Parameters

- **cols** (*list*) – Columns to encode. If None, all string columns will be encoded, otherwise only the columns provided will be encoded. Defaults to None
- **smoothing** (*float*) – The smoothing factor to apply. The larger this value is, the more influence the expected target value has on the resulting target encodings. Must be strictly larger than 0. Defaults to 1.0

- **handle_unknown** (*string*) – Determines how to handle unknown categories for a feature encountered. Options are 'value', 'error', nd 'return_nan'. Defaults to 'value', which replaces with the target mean
- **handle_missing** (*string*) – Determines how to handle missing values encountered during *fit* or *transform*. Options are 'value', 'error', and 'return_nan'. Defaults to 'value', which replaces with the target mean
- **random_seed** (*int*) – Seed for the random number generator. Defaults to 0.

evalml.pipelines.components.TargetEncoder.clone

TargetEncoder.**clone**()

Constructs a new component with the same parameters and random state.

Returns A new instance of this component with identical parameters and random state.

evalml.pipelines.components.TargetEncoder.describe

TargetEncoder.**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.TargetEncoder.fit

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

Fits component to data

Parameters

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

Returns self

evalml.pipelines.components.TargetEncoder.fit_transform

TargetEncoder.**fit_transform**(*X*, *y*)
Fits on X and transforms X

Parameters

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

Returns Transformed X

Return type pd.DataFrame

evalml.pipelines.components.TargetEncoder.get_feature_names

TargetEncoder.**get_feature_names**()
Return feature names for the input features after fitting.

Returns The feature names after encoding

Return type np.array

evalml.pipelines.components.TargetEncoder.load

static TargetEncoder.**load**(*file_path*)
Loads component at file path

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

Returns ComponentBase object

evalml.pipelines.components.TargetEncoder.save

TargetEncoder.**save**(*file_path*, *pickle_protocol=5*)
Saves component at file path

Parameters

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

Returns None

evalml.pipelines.components.TargetEncoder.transform

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

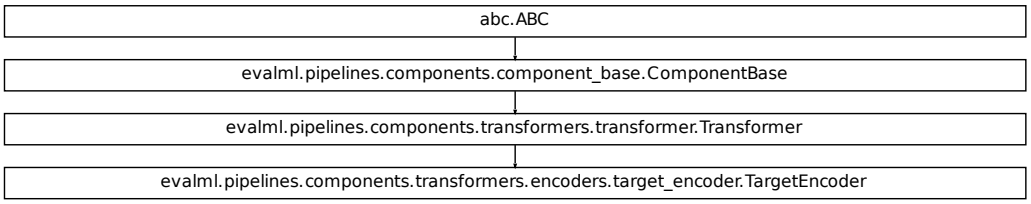
Parameters

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

Returns Transformed X

Return type pd.DataFrame

Class Inheritance



evalml.pipelines.components.PerColumnImputer

class evalml.pipelines.components.**PerColumnImputer**(*impute_strategies=None*, *default_impute_strategy='most_frequent'*, *random_seed=0*, ***kwargs*)

Imputes missing data according to a specified imputation strategy per column

name = 'Per Column Imputer'

model_family = 'none'

hyperparameter_ranges = {}

default_parameters = {'default_impute_strategy': 'most_frequent', 'impute_strategies':

Instance attributes

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

Methods:

__init__	Initializes a transformer that imputes missing data according to the specified imputation strategy per column."
clone	Constructs a new component with the same parameters and random state.
describe	Describe a component and its parameters
fit	Fits imputers on input data

continues on next page

Table 49 – continued from previous page

fit_transform	Fits on X and transforms X
load	Loads component at file path
save	Saves component at file path
transform	Transforms input data by imputing missing values.

evalml.pipelines.components.PerColumnImputer.__init__

PerColumnImputer.**__init__**(*impute_strategies=None*, *default_impute_strategy='most_frequent'*, *random_seed=0*, ***kwargs*)

Initializes a transformer that imputes missing data according to the specified imputation strategy per column."

Parameters

- **impute_strategies** (*dict*) – Column and {"impute_strategy": strategy, "fill_value":value} pairings. Valid values for impute strategy include "mean", "median", "most_frequent", "constant" for numerical data, and "most_frequent", "constant" for object data types. Defaults to "most_frequent" for all columns.

  When impute_strategy == "constant", fill_value is used to replace missing data. Defaults to 0 when imputing numerical data and "missing_value" for strings or object data types.

- **default_impute_strategy** (*str*) – Impute strategy to fall back on when none is provided for a certain column. Valid values include "mean", "median", "most_frequent", "constant" for numerical data, and "most_frequent", "constant" for object data types. Defaults to "most_frequent"

- **random_seed** (*int*) – Seed for the random number generator. Defaults to 0.

evalml.pipelines.components.PerColumnImputer.clone

PerColumnImputer.**clone**()

Constructs a new component with the same parameters and random state.

Returns A new instance of this component with identical parameters and random state.

evalml.pipelines.components.PerColumnImputer.describe

PerColumnImputer.**describe**(*print_name=False*, *return_dict=False*)

Describe a component and its parameters

Parameters

- **print_name** (*bool, optional*) – whether to print name of component
- **return_dict** (*bool, optional*) – whether to return description as dictionary in the format {"name": name, "parameters": parameters}

Returns prints and returns dictionary

Return type None or dict

evalml.pipelines.components.PerColumnImputer.fit

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

Fits imputers on input data

Parameters

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

Returns self

evalml.pipelines.components.PerColumnImputer.fit_transform

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

Fits on X and transforms X

Parameters

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

Returns Transformed X

Return type pd.DataFrame

evalml.pipelines.components.PerColumnImputer.load

static PerColumnImputer.**load**(*file_path*)

Loads component at file path

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

Returns ComponentBase object

evalml.pipelines.components.PerColumnImputer.save

PerColumnImputer.**save**(*file_path*, *pickle_protocol=5*)

Saves component at file path

Parameters

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

Returns None

evalml.pipelines.components.PerColumnImputer.transform

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

Transforms input data by imputing missing values.

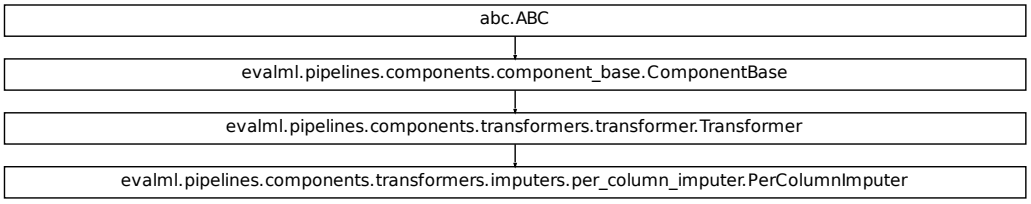
Parameters

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

Returns Transformed X

Return type pd.DataFrame

Class Inheritance



evalml.pipelines.components.Imputer

class evalml.pipelines.components.**Imputer**(*categorical_impute_strategy='most_frequent'*, *categorical_fill_value=None*, *numeric_impute_strategy='mean'*, *numeric_fill_value=None*, *random_seed=0*, ***kwargs*)

Imputes missing data according to a specified imputation strategy.

name = 'Imputer'

model_family = 'none'

hyperparameter_ranges = {'categorical_impute_strategy': ['most_frequent'], 'numeric_imp

default_parameters = {'categorical_fill_value': None, 'categorical_impute_strategy': '

Instance attributes

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

Methods:

__init__	Initalizes an transformer that imputes missing data according to the specified imputation strategy."
clone	Constructs a new component with the same parameters and random state.
describe	Describe a component and its parameters
fit	Fits imputer to data. 'None' values are converted to np.nan before imputation and are
fit_transform	Fits on X and transforms X
load	Loads component at file path
save	Saves component at file path
transform	Transforms data X by imputing missing values. 'None' values are converted to np.nan before imputation and are

evalml.pipelines.components.Imputer.__init__

Imputer.**__init__**(*categorical_impute_strategy='most_frequent'*, *categorical_fill_value=None*, *numeric_impute_strategy='mean'*, *numeric_fill_value=None*, *random_seed=0*, ***kwargs*)

Initalizes an transformer that imputes missing data according to the specified imputation strategy."

Parameters

- **categorical_impute_strategy** (*string*) – Impute strategy to use for string, object, boolean, categorical dtypes. Valid values include "most_frequent" and "constant".
- **numeric_impute_strategy** (*string*) – Impute strategy to use for numeric columns. Valid values include "mean", "median", "most_frequent", and "constant".
- **categorical_fill_value** (*string*) – When categorical_impute_strategy == "constant", fill_value is used to replace missing data. The default value of None will fill with the string "missing_value".
- **numeric_fill_value** (*int, float*) – When numeric_impute_strategy == "constant", fill_value is used to replace missing data. The default value of None will fill with 0.
- **random_seed** (*int*) – Seed for the random number generator. Defaults to 0.

evalml.pipelines.components.Imputer.clone

Imputer.**clone**()

Constructs a new component with the same parameters and random state.

Returns A new instance of this component with identical parameters and random state.

evalml.pipelines.components.Imputer.describe

Imputer.**describe**(*print_name=False*, *return_dict=False*)

Describe a component and its parameters

Parameters

- **print_name** (*bool, optional*) – whether to print name of component
- **return_dict** (*bool, optional*) – whether to return description as dictionary in the format {"name": name, "parameters": parameters}

Returns prints and returns dictionary

Return type None or dict

evalml.pipelines.components.Imputer.fit

Imputer.**fit**(*X*, *y=None*)

Fits imputer to data. 'None' values are converted to np.nan before imputation and are treated as the same.

Parameters

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

Returns self

evalml.pipelines.components.Imputer.fit_transform

Imputer.**fit_transform**(*X*, *y=None*)

Fits on X and transforms X

Parameters

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

Returns Transformed X

Return type pd.DataFrame

evalml.pipelines.components.Imputer.load

static Imputer.**load**(*file_path*)

Loads component at file path

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

Returns ComponentBase object

evalml.pipelines.components.Imputer.save

Imputer.**save**(*file_path*, *pickle_protocol=5*)

Saves component at file path

Parameters

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

Returns None

evalml.pipelines.components.Imputer.transform

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

Transforms data X by imputing missing values. 'None' values are converted to np.nan before imputation and are treated as the same.

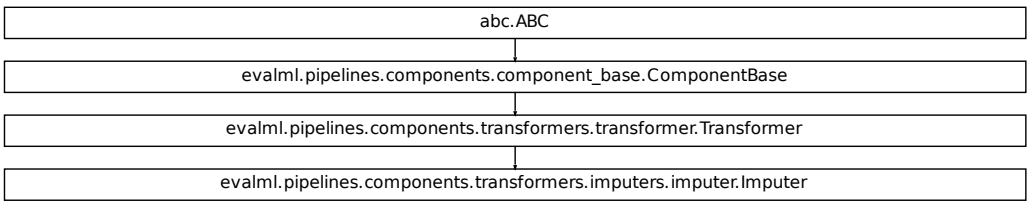
Parameters

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

Returns Transformed X

Return type pd.DataFrame

Class Inheritance



evalml.pipelines.components.SimpleImputer

class evalml.pipelines.components.**SimpleImputer**(*impute_strategy='most_frequent'*, *fill_value=None*, *random_seed=0*, ***kwargs*)

Imputes missing data according to a specified imputation strategy.

name = 'Simple Imputer'

model_family = 'none'

hyperparameter_ranges = {'impute_strategy': ['mean', 'median', 'most_frequent']}

default_parameters = {'fill_value': None, 'impute_strategy': 'most_frequent'}

Instance attributes

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

Methods:

__init__	Initalizes an transformer that imputes missing data according to the specified imputation strategy."
clone	Constructs a new component with the same parameters and random state.
describe	Describe a component and its parameters
fit	Fits imputer to data. 'None' values are converted to np.nan before imputation and are
fit_transform	Fits on X and transforms X
load	Loads component at file path
save	Saves component at file path
transform	Transforms input by imputing missing values.

evalml.pipelines.components.SimpleImputer.__init__

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

Initalizes an transformer that imputes missing data according to the specified imputation strategy."

Parameters

- **impute_strategy** (*string*) – Impute strategy to use. Valid values include "mean", "median", "most_frequent", "constant" for numerical data, and "most_frequent", "constant" for object data types.
- **fill_value** (*string*) – When impute_strategy == "constant", fill_value is used to replace missing data. Defaults to 0 when imputing numerical data and "missing_value" for strings or object data types.
- **random_seed** (*int*) – Seed for the random number generator. Defaults to 0.

evalml.pipelines.components.SimpleImputer.clone

SimpleImputer.**clone**()

Constructs a new component with the same parameters and random state.

Returns A new instance of this component with identical parameters and random state.

evalml.pipelines.components.SimpleImputer.describe

SimpleImputer.**describe**(*print_name=False*, *return_dict=False*)

Describe a component and its parameters

Parameters

- **print_name** (*bool, optional*) – whether to print name of component
- **return_dict** (*bool, optional*) – whether to return description as dictionary in the format {"name": name, "parameters": parameters}

Returns prints and returns dictionary

Return type None or dict

evalml.pipelines.components.SimpleImputer.fit

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

Fits imputer to data. 'None' values are converted to np.nan before imputation and are treated as the same.

Parameters

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

Returns self

evalml.pipelines.components.SimpleImputer.fit_transform

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

Fits on X and transforms X

Parameters

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

Returns Transformed X

Return type pd.DataFrame

evalml.pipelines.components.SimpleImputer.load

static SimpleImputer.**load**(*file_path*)

Loads component at file path

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

Returns ComponentBase object

evalml.pipelines.components.SimpleImputer.save

SimpleImputer.**save**(*file_path*, *pickle_protocol=5*)

Saves component at file path

Parameters

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

Returns None

evalml.pipelines.components.SimpleImputer.transform

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

Transforms input by imputing missing values. 'None' and np.nan values are treated as the same.

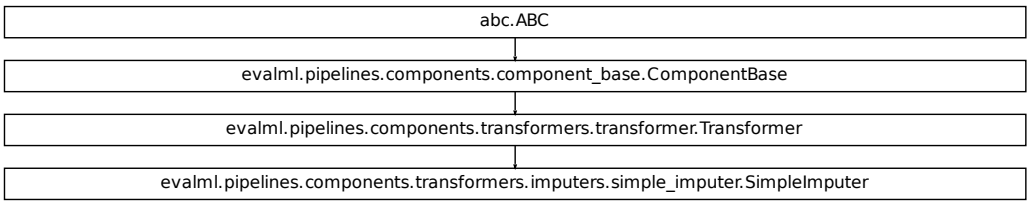
Parameters

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

Returns Transformed X

Return type pd.DataFrame

Class Inheritance



evalml.pipelines.components.StandardScaler

class evalml.pipelines.components.**StandardScaler**(*random_seed=0*, ***kwargs*)

Standardize features: removes mean and scales to unit variance.

name = 'Standard Scaler'

model_family = 'none'

hyperparameter_ranges = {}

default_parameters = {}

Instance attributes

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

Methods:

__init__	Initialize self.
clone	Constructs a new component with the same parameters and random state.
describe	Describe a component and its parameters
fit	Fits component to data
fit_transform	Fits on X and transforms X
load	Loads component at file path
save	Saves component at file path
transform	Transforms data X.

evalml.pipelines.components.StandardScaler.__init__

StandardScaler.**__init__**(*random_seed=0*, ***kwargs*)

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

evalml.pipelines.components.StandardScaler.clone

StandardScaler.**clone**()

Constructs a new component with the same parameters and random state.

Returns A new instance of this component with identical parameters and random state.

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

Returns self

evalml.pipelines.components.StandardScaler.fit_transform

StandardScaler.**fit_transform**(*X*, *y=None*)

Fits on X and transforms X

Parameters

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

Returns Transformed X

Return type pd.DataFrame

evalml.pipelines.components.StandardScaler.load

static StandardScaler.**load**(*file_path*)
Loads component at file path

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

Returns ComponentBase object

evalml.pipelines.components.StandardScaler.save

StandardScaler.**save**(*file_path*, *pickle_protocol=5*)
Saves component at file path

Parameters

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

Returns None

evalml.pipelines.components.StandardScaler.transform

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

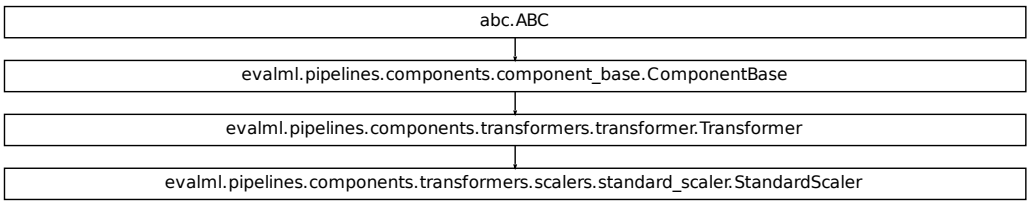
Parameters

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

Returns Transformed X

Return type pd.DataFrame

Class Inheritance



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_seed=0*, ***kwargs*)

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

name = 'RF Regressor Select From Model'

model_family = 'none'

hyperparameter_ranges = {'percent_features': Real(low=0.01, high=1, prior='uniform', t

default_parameters = {'max_depth': None, 'n_estimators': 10, 'n_jobs': -1, 'number_fea

Instance attributes

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

Methods:

__init__	Initialize self.
clone	Constructs a new component with the same parameters and random state.
describe	Describe a component and its parameters
fit	Fits component to data
fit_transform	Fits on X and transforms X
get_names	Get names of selected features.
load	Loads component at file path
save	Saves component at file path
transform	Transforms input data 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_seed=0*, ***kwargs*)

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

evalml.pipelines.components.RFRegressorSelectFromModel.clone

RFRegressorSelectFromModel.**clone**()

Constructs a new component with the same parameters and random state.

Returns A new instance of this component with identical parameters and random state.

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

Returns self

evalml.pipelines.components.RFRegressorSelectFromModel.fit_transform

RFRegressorSelectFromModel.**fit_transform**(*X*, *y=None*)

Fits on X and transforms X

Parameters

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

Returns Transformed X

Return type pd.DataFrame

evalml.pipelines.components.RFRegressorSelectFromModel.get_names

RFRegressorSelectFromModel.**get_names**()

Get names of selected features.

Returns List of the names of features selected

Return type list[str]

evalml.pipelines.components.RFRegressorSelectFromModel.load

static RFRegressorSelectFromModel.**load**(*file_path*)

Loads component at file path

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

Returns ComponentBase object

evalml.pipelines.components.RFRegressorSelectFromModel.save

RFRegressorSelectFromModel.**save**(*file_path*, *pickle_protocol=5*)

Saves component at file path

Parameters

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

Returns None

evalml.pipelines.components.RFRegressorSelectFromModel.transform

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

Transforms input data by selecting features. If the component_obj does not have a transform method, will raise an MethodPropertyNotFoundError exception.

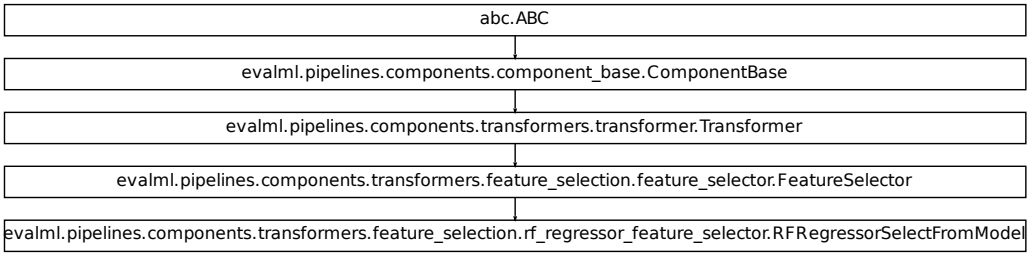
Parameters

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

Returns Transformed X

Return type pd.DataFrame

Class Inheritance



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_seed=0*, ***kwargs*)

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

```
name = 'RF Classifier Select From Model'
model_family = 'none'
hyperparameter_ranges = {'percent_features': Real(low=0.01, high=1, prior='uniform', t
default_parameters = {'max_depth': None, 'n_estimators': 10, 'n_jobs': -1, 'number_fea
```

Instance attributes

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

Methods:

__init__	Initialize self.
clone	Constructs a new component with the same parameters and random state.
describe	Describe a component and its parameters
fit	Fits component to data
fit_transform	Fits on X and transforms X
get_names	Get names of selected features.
load	Loads component at file path
save	Saves component at file path
transform	Transforms input data 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_seed=0*, ***kwargs*)

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

evalml.pipelines.components.RFClassifierSelectFromModel.clone

RFClassifierSelectFromModel.**clone**()

Constructs a new component with the same parameters and random state.

Returns A new instance of this component with identical parameters and random state.

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

Returns self

evalml.pipelines.components.RFClassifierSelectFromModel.fit_transform

RFClassifierSelectFromModel.**fit_transform**(*X*, *y=None*)

Fits on X and transforms X

Parameters

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

Returns Transformed X

Return type pd.DataFrame

evalml.pipelines.components.RFClassifierSelectFromModel.get_names

RFClassifierSelectFromModel.**get_names**()

Get names of selected features.

Returns List of the names of features selected

Return type list[str]

evalml.pipelines.components.RFClassifierSelectFromModel.load

static RFClassifierSelectFromModel.**load**(*file_path*)

Loads component at file path

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

Returns ComponentBase object

evalml.pipelines.components.RFClassifierSelectFromModel.save

RFClassifierSelectFromModel.**save**(*file_path*, *pickle_protocol=5*)

Saves component at file path

Parameters

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

Returns None

evalml.pipelines.components.RFClassifierSelectFromModel.transform

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

Transforms input data by selecting features. If the component_obj does not have a transform method, will raise an MethodPropertyNotFoundError exception.

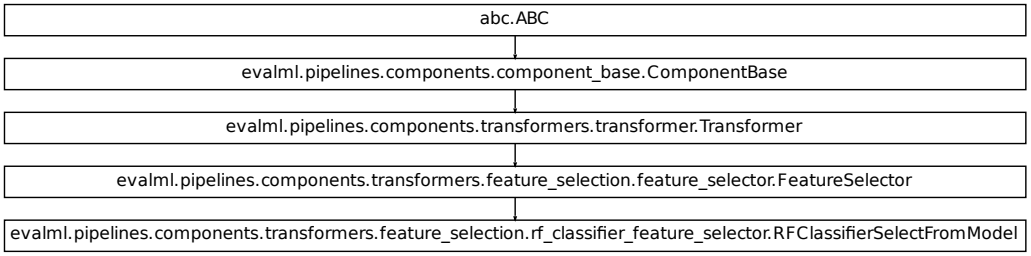
Parameters

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

Returns Transformed X

Return type pd.DataFrame

Class Inheritance



evalml.pipelines.components.DropNullColumns

class evalml.pipelines.components.**DropNullColumns**(*pct_null_threshold=1.0*, *random_seed=0*, ***kwargs*)

Transformer to drop features whose percentage of NaN values exceeds a specified threshold

name = 'Drop Null Columns Transformer'

model_family = 'none'

hyperparameter_ranges = {}

default_parameters = {'pct_null_threshold': 1.0}

Instance attributes

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

Methods:

__init__	Initalizes an transformer to drop features whose percentage of NaN values exceeds a specified threshold.
clone	Constructs a new component with the same parameters and random state.
describe	Describe a component and its parameters
fit	Fits component to data
fit_transform	Fits on X and transforms X
load	Loads component at file path
save	Saves component at file path
transform	Transforms data X by dropping columns that exceed the threshold of null values.

evalml.pipelines.components.DropNullColumns.__init__

DropNullColumns.**__init__**(*pct_null_threshold=1.0*, *random_seed=0*, ***kwargs*)

Initalizes an transformer to drop features whose percentage of NaN values exceeds a specified threshold.

Parameters

- **pct_null_threshold** (*float*) – The percentage of NaN values in an input feature to drop. Must be a value between [0, 1] inclusive. If equal to 0.0, will drop columns with any null values. If equal to 1.0, will drop columns with all null values. Defaults to 0.95.
- **random_seed** (*int*) – Seed for the random number generator. Defaults to 0.

evalml.pipelines.components.DropNullColumns.clone

DropNullColumns.**clone**()

Constructs a new component with the same parameters and random state.

Returns A new instance of this component with identical parameters and random state.

evalml.pipelines.components.DropNullColumns.describe

DropNullColumns.**describe**(*print_name=False*, *return_dict=False*)

Describe a component and its parameters

Parameters

- **print_name** (*bool, optional*) – whether to print name of component
- **return_dict** (*bool, optional*) – whether to return description as dictionary in the format {"name": name, "parameters": parameters}

Returns prints and returns dictionary

Return type None or dict

evalml.pipelines.components.DropNullColumns.fit

DropNullColumns.**fit**(*X*, *y=None*)

Fits component to data

Parameters

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

Returns self

evalml.pipelines.components.DropNullColumns.fit_transform

DropNullColumns.**fit_transform**(*X*, *y=None*)

Fits on X and transforms X

Parameters

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

Returns Transformed X

Return type pd.DataFrame

evalml.pipelines.components.DropNullColumns.load

static DropNullColumns.**load**(*file_path*)
Loads component at file path

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

Returns ComponentBase object

evalml.pipelines.components.DropNullColumns.save

DropNullColumns.**save**(*file_path*, *pickle_protocol=5*)
Saves component at file path

Parameters

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

Returns None

evalml.pipelines.components.DropNullColumns.transform

DropNullColumns.**transform**(*X*, *y=None*)
Transforms data X by dropping columns that exceed the threshold of null values.

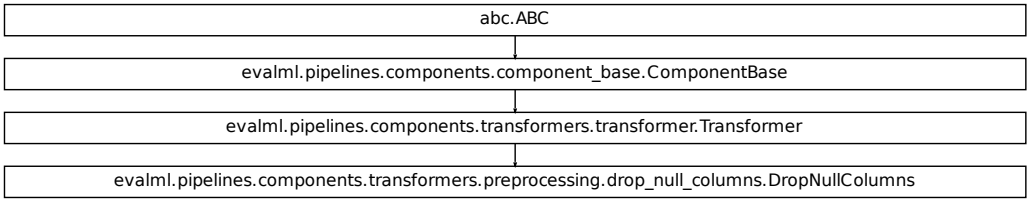
Parameters

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

Returns Transformed X

Return type pd.DataFrame

Class Inheritance



evalml.pipelines.components.DateTimeFeaturizer

class evalml.pipelines.components.**DateTimeFeaturizer**(*features_to_extract=None*, *encode_as_categories=False*, *date_index=None*, *random_seed=0*, ***kwargs*)

Transformer that can automatically featurize DateTime columns.

name = 'DateTime Featurization Component'

model_family = 'none'

hyperparameter_ranges = {}

default_parameters = {'date_index': None, 'encode_as_categories': False, 'features_to_

Instance attributes

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

Methods:

__init__	Extracts features from DateTime columns
clone	Constructs a new component with the same parameters and random state.
describe	Describe a component and its parameters
fit	Fits component to data
fit_transform	Fits on X and transforms X
get_feature_names	Gets the categories of each datetime feature.
load	Loads component at file path
save	Saves component at file path
transform	Transforms data X by creating new features using existing DateTime columns, and then dropping those DateTime columns

evalml.pipelines.components.DateTimeFeaturizer.__init__

DateTimeFeaturizer.**__init__**(*features_to_extract=None*, *encode_as_categories=False*, *date_index=None*, *random_seed=0*, ***kwargs*)

Extracts features from DateTime columns

Parameters

- **features_to_extract** (*list*) – List of features to extract. Valid options include "year", "month", "day_of_week", "hour".
- **encode_as_categories** (*bool*) – Whether day-of-week and month features should be encoded as pandas "category" dtype. This allows OneHotEncoders to encode these features.

- **random_seed** (*int*) – Seed for the random number generator. Defaults to 0.
- **date_index** (*str*) – Name of the column containing the datetime information used to order the data. Ignored.

evalml.pipelines.components.DateTimeFeaturizer.clone

DateTimeFeaturizer.**clone**()

Constructs a new component with the same parameters and random state.

Returns A new instance of this component with identical parameters and random state.

evalml.pipelines.components.DateTimeFeaturizer.describe

DateTimeFeaturizer.**describe**(*print_name=False*, *return_dict=False*)

Describe a component and its parameters

Parameters

- **print_name** (*bool, optional*) – whether to print name of component
- **return_dict** (*bool, optional*) – whether to return description as dictionary in the format {"name": name, "parameters": parameters}

Returns prints and returns dictionary

Return type None or dict

evalml.pipelines.components.DateTimeFeaturizer.fit

DateTimeFeaturizer.**fit**(*X*, *y=None*)

Fits component to data

Parameters

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

Returns self

evalml.pipelines.components.DateTimeFeaturizer.fit_transform

DateTimeFeaturizer.**fit_transform**(*X*, *y=None*)

Fits on X and transforms X

Parameters

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

Returns Transformed X

Return type pd.DataFrame

evalml.pipelines.components.DateTimeFeaturizer.get_feature_names

DateTimeFeaturizer.**get_feature_names**()

Gets the categories of each datetime feature.

Returns Dictionary, where each key-value pair is a column name and a dictionary mapping the unique feature values to their integer encoding.

evalml.pipelines.components.DateTimeFeaturizer.load

static DateTimeFeaturizer.**load**(*file_path*)

Loads component at file path

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

Returns ComponentBase object

evalml.pipelines.components.DateTimeFeaturizer.save

DateTimeFeaturizer.**save**(*file_path*, *pickle_protocol=5*)

Saves component at file path

Parameters

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

Returns None

evalml.pipelines.components.DateTimeFeaturizer.transform

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

Transforms data X by creating new features using existing DateTime columns, and then dropping those DateTime columns

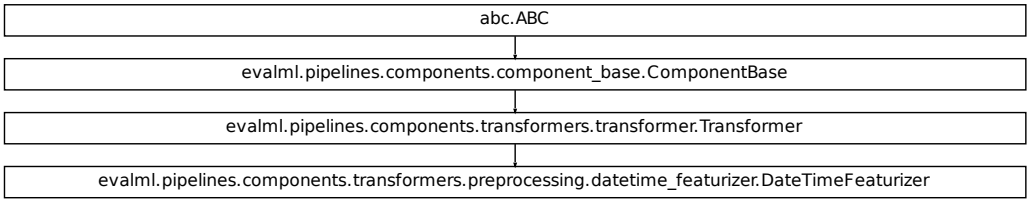
Parameters

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

Returns Transformed X

Return type pd.DataFrame

Class Inheritance



evalml.pipelines.components.TextFeaturizer

class evalml.pipelines.components.**TextFeaturizer**(*random_seed=0*, ***kwargs*)

Transformer that can automatically featurize text columns.

name = 'Text Featurization Component'

model_family = 'none'

hyperparameter_ranges = {}

default_parameters = {}

Instance attributes

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

Methods:

__init__	Extracts features from text columns using featuretools' nlp_primitives
clone	Constructs a new component with the same parameters and random state.
describe	Describe a component and its parameters
fit	Fits component to data
fit_transform	Fits on X and transforms X
load	Loads component at file path
save	Saves component at file path
transform	Transforms data X by creating new features using existing text columns

evalml.pipelines.components.TextFeaturizer.__init__

TextFeaturizer.**__init__**(*random_seed=0*, ***kwargs*)
Extracts features from text columns using featuretools' nlp_primitives

Parameters **random_seed** (*int*) – Seed for the random number generator. Defaults to 0.

evalml.pipelines.components.TextFeaturizer.clone

TextFeaturizer.**clone**()
Constructs a new component with the same parameters and random state.

Returns A new instance of this component with identical parameters and random state.

evalml.pipelines.components.TextFeaturizer.describe

TextFeaturizer.**describe**(*print_name=False*, *return_dict=False*)
Describe a component and its parameters

Parameters

- **print_name** (*bool, optional*) – whether to print name of component
- **return_dict** (*bool, optional*) – whether to return description as dictionary in the format {"name": name, "parameters": parameters}

Returns prints and returns dictionary

Return type None or dict

evalml.pipelines.components.TextFeaturizer.fit

TextFeaturizer.**fit**(*X*, *y=None*)
Fits component to data

Parameters

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

Returns self

evalml.pipelines.components.TextFeaturizer.fit_transform

TextFeaturizer.**fit_transform**(*X*, *y=None*)
Fits on X and transforms X

Parameters

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

Returns Transformed X

Return type pd.DataFrame

evalml.pipelines.components.TextFeaturizer.load

static TextFeaturizer.**load**(*file_path*)

Loads component at file path

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

Returns ComponentBase object

evalml.pipelines.components.TextFeaturizer.save

TextFeaturizer.**save**(*file_path*, *pickle_protocol=5*)

Saves component at file path

Parameters

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

Returns None

evalml.pipelines.components.TextFeaturizer.transform

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

Transforms data X by creating new features using existing text columns

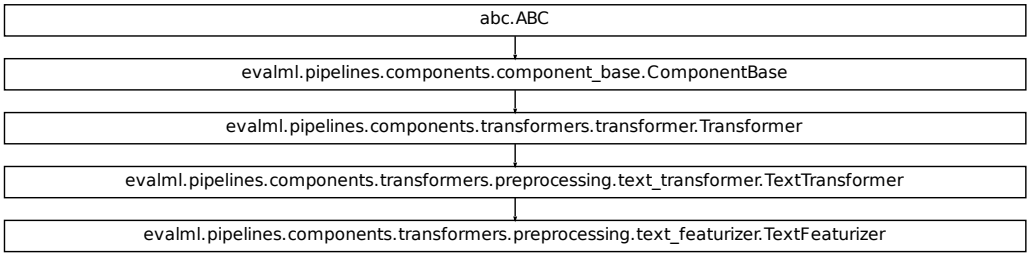
Parameters

- **X** (*pd.DataFrame*) – The data to transform.
- **y** (*pd.Series, optional*) – Ignored.

Returns Transformed X

Return type pd.DataFrame

Class Inheritance



evalml.pipelines.components.DelayedFeatureTransformer

class evalml.pipelines.components.**DelayedFeatureTransformer**(*date_index=None*, *max_delay=2*, *delay_features=True*, *delay_target=True*, *gap=1*, *random_seed=0*, ***kwargs*)

Transformer that delayes input features and target variable for time series problems.

name = 'Delayed Feature Transformer'

model_family = 'none'

hyperparameter_ranges = {}

default_parameters = {'date_index': None, 'delay_features': True, 'delay_target': True

Instance attributes

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

Methods:

__init__	Creates a DelayedFeatureTransformer.
clone	Constructs a new component with the same parameters and random state.
describe	Describe a component and its parameters
fit	Fits the DelayFeatureTransformer.
fit_transform	Fits on X and transforms X
load	Loads component at file path
save	Saves component at file path
transform	Computes the delayed features for all features in X and y.

evalml.pipelines.components.DelayedFeatureTransformer.__init__

DelayedFeatureTransformer.**__init__**(*date_index=None*, *max_delay=2*, *delay_features=True*, *delay_target=True*, *gap=1*, *random_seed=0*, ***kwargs*)

Creates a DelayedFeatureTransformer.

Parameters

- **date_index** (*str*) – Name of the column containing the datetime information used to order the data. Ignored.
- **max_delay** (*int*) – Maximum number of time units to delay each feature.
- **delay_features** (*bool*) – Whether to delay the input features.
- **delay_target** (*bool*) – Whether to delay the target.
- **gap** (*int*) – The number of time units between when the features are collected and when the target is collected. For example, if you are predicting the next time step's target, gap=1. This is only needed because when gap=0, we need to be sure to start the lagging of the target variable at 1.
- **random_seed** (*int*) – Seed for the random number generator. This transformer performs the same regardless of the random seed provided.

evalml.pipelines.components.DelayedFeatureTransformer.clone

DelayedFeatureTransformer.**clone**()

Constructs a new component with the same parameters and random state.

Returns A new instance of this component with identical parameters and random state.

evalml.pipelines.components.DelayedFeatureTransformer.describe

DelayedFeatureTransformer.**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.DelayedFeatureTransformer.fit

DelayedFeatureTransformer.**fit**(*X*, *y=None*)

Fits the DelayFeatureTransformer.

Parameters

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

Returns self

evalml.pipelines.components.DelayedFeatureTransformer.fit_transform

DelayedFeatureTransformer.**fit_transform**(*X*, *y*)

Fits on X and transforms X

Parameters

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

Returns Transformed X

Return type pd.DataFrame

evalml.pipelines.components.DelayedFeatureTransformer.load

static DelayedFeatureTransformer.**load**(*file_path*)

Loads component at file path

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

Returns ComponentBase object

evalml.pipelines.components.DelayedFeatureTransformer.save

DelayedFeatureTransformer.**save**(*file_path*, *pickle_protocol=5*)

Saves component at file path

Parameters

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

Returns None

evalml.pipelines.components.DelayedFeatureTransformer.transform

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

Computes the delayed features for all features in X and y.

For each feature in X, it will add a column to the output dataframe for each delay in the (inclusive) range [1, max_delay]. The values of each delayed feature are simply the original feature shifted forward in time by the delay amount. For example, a delay of 3 units means that the feature value at row n will be taken from the n-3rd row of that feature

If y is not None, it will also compute the delayed values for the target variable.

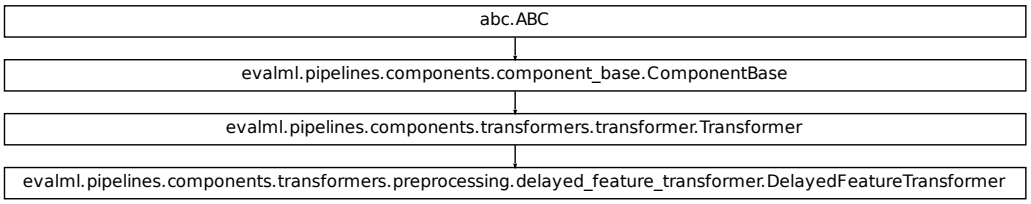
Parameters

- **X** (*pd.DataFrame or None*) – Data to transform. None is expected when only the target variable is being used.
- **y** (*pd.Series, or None*) – Target.

Returns Transformed X.

Return type pd.DataFrame

Class Inheritance



evalml.pipelines.components.DFSTransformer

class evalml.pipelines.components.**DFSTransformer**(*index='index'*, *random_seed=0*, ***kwargs*)

Featuretools DFS component that generates features for pd.DataFrames

name = 'DFS Transformer'

model_family = 'none'

hyperparameter_ranges = {}

default_parameters = {'index': 'index'}

Instance attributes

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

Methods:

__init__	Allows for featuretools to be used in EvalML.
clone	Constructs a new component with the same parameters and random state.
describe	Describe a component and its parameters
fit	Fits the DFSTransformer Transformer component.
fit_transform	Fits on X and transforms X
load	Loads component at file path
save	Saves component at file path
transform	Computes the feature matrix for the input X using featuretools' dfs algorithm.

evalml.pipelines.components.DFSTransformer.__init__

DFSTransformer.**__init__**(*index='index'*, *random_seed=0*, ***kwargs*)

Allows for featuretools to be used in EvalML.

Parameters

- **index** (*string*) – The name of the column that contains the indices. If no column with this name exists, then featuretools.EntitySet() creates a column with this name to serve as the index column. Defaults to 'index'
- **random_seed** (*int*) – Seed for the random number generator

evalml.pipelines.components.DFSTransformer.clone

DFSTransformer.**clone**()

Constructs a new component with the same parameters and random state.

Returns A new instance of this component with identical parameters and random state.

evalml.pipelines.components.DFSTransformer.describe

DFSTransformer.**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.DFSTransformer.fit

DFSTransformer.**fit**(*X*, *y=None*)

Fits the DFSTransformer Transformer component.

Parameters

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

Returns self

evalml.pipelines.components.DFSTransformer.fit_transform

DFSTransformer.**fit_transform**(*X*, *y=None*)

Fits on X and transforms X

Parameters

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

Returns Transformed X

Return type pd.DataFrame

evalml.pipelines.components.DFSTransformer.load

static DFSTransformer.**load**(*file_path*)

Loads component at file path

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

Returns ComponentBase object

evalml.pipelines.components.DFSTransformer.save

DFSTransformer.**save**(*file_path*, *pickle_protocol=5*)

Saves component at file path

Parameters

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

Returns None

evalml.pipelines.components.DFSTransformer.transform

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

Computes the feature matrix for the input X using featuretools' dfs algorithm.

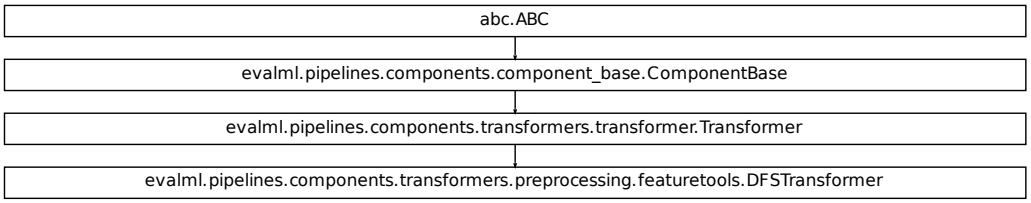
Parameters

- **X** (*pd.DataFrame or np.ndarray*) – The input training data to transform. Has shape [n_samples, n_features]
- **y** (*pd.Series, optional*) – Ignored.

Returns Feature matrix

Return type pd.DataFrame

Class Inheritance



evalml.pipelines.components.PolynomialDetrender

class evalml.pipelines.components.**PolynomialDetrender**(*degree=1*, *random_seed=0*, ***kwargs*)

Removes trends from time series by fitting a polynomial to the data.

name = 'Polynomial Detrender'

model_family = 'none'

hyperparameter_ranges = {'degree': Integer(low=1, high=3, prior='uniform', transform='

default_parameters = {'degree': 1}

Instance attributes

needs_fitting	

parameters	Returns the parameters which were used to initialize the component

Methods:

__init__	Initialize the PolynomialDetrender.
clone	Constructs a new component with the same parameters and random state.
describe	Describe a component and its parameters
fit	Fits the PolynomialDetrender.
fit_transform	Removes fitted trend from target variable.
inverse_transform	Adds back fitted trend to target variable.
load	Loads component at file path
save	Saves component at file path
transform	Removes fitted trend from target variable.

evalml.pipelines.components.PolynomialDetrender.__init__

PolynomialDetrender.**__init__**(*degree=1*, *random_seed=0*, ***kwargs*)

Initialize the PolynomialDetrender.

Parameters

- **degree** (*int*) – Degree for the polynomial. If 1, linear model is fit to the data. If 2, quadratic model is fit, etc. Default of 1.
- **random_seed** (*int*) – Seed for the random number generator. Defaults to 0.

evalml.pipelines.components.PolynomialDetrender.clone

PolynomialDetrender.**clone**()

Constructs a new component with the same parameters and random state.

Returns A new instance of this component with identical parameters and random state.

evalml.pipelines.components.PolynomialDetrender.describe

PolynomialDetrender.**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.PolynomialDetrender.fit

PolynomialDetrender.**fit**(*X*, *y=None*)

Fits the PolynomialDetrender.

Parameters

- **X** (*pd.DataFrame, optional*) – Ignored.
- **y** (*pd.Series*) – Target variable to detrend.

Returns self

evalml.pipelines.components.PolynomialDetrender.fit_transform

PolynomialDetrender.**fit_transform**(*X*, *y=None*)

Removes fitted trend from target variable.

Parameters

- **X** (*pd.DataFrame, optional*) – Ignored.
- **y** (*pd.Series*) – Target variable to detrend.

Returns

The first element are the input features returned without modification. The second element is the target variable y with the fitted trend removed.

Return type tuple of pd.DataFrame, pd.Series

evalml.pipelines.components.PolynomialDetrender.inverse_transform

PolynomialDetrender.**inverse_transform**(*y*)

Adds back fitted trend to target variable.

Parameters

- **X** (*pd.DataFrame, optional*) – Ignored.
- **y** (*pd.Series*) – Target variable.

Returns

The first element are the input features returned without modification. The second element is the target variable y with the trend added back.

Return type tuple of pd.DataFrame, pd.Series

evalml.pipelines.components.PolynomialDetrender.load

static PolynomialDetrender.**load**(*file_path*)

Loads component at file path

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

Returns ComponentBase object

evalml.pipelines.components.PolynomialDetrender.save

PolynomialDetrender.**save**(*file_path*, *pickle_protocol=5*)

Saves component at file path

Parameters

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

Returns None

evalml.pipelines.components.PolynomialDetrender.transform

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

Removes fitted trend from target variable.

Parameters

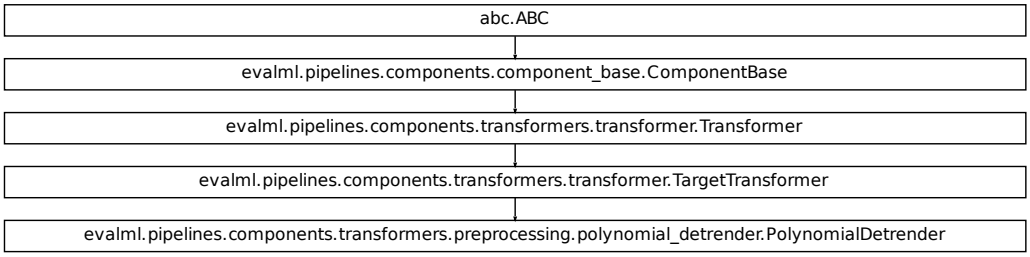
- **X** (*pd.DataFrame, optional*) – Ignored.
- **y** (*pd.Series*) – Target variable to detrend.

Returns

The input features are returned without modification. The target variable y is detrended

Return type tuple of pd.DataFrame, pd.Series

Class Inheritance



evalml.pipelines.components.Undersampler

class evalml.pipelines.components.**Undersampler**(*sampling_ratio=0.25*, *sampling_ratio_dict=None*, *min_samples=100*, *min_percentage=0.1*, *random_seed=0*, ***kwargs*)

Random undersampler component. This component is only run during training and not during predict.

name = 'Undersampler'

model_family = 'none'

hyperparameter_ranges = {}

default_parameters = {'min_percentage': 0.1, 'min_samples': 100, 'sampling_ratio': 0.2

Instance attributes

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

Methods:

__init__	Initializes an undersampling transformer to downsample the majority classes in the dataset.
clone	Constructs a new component with the same parameters and random state.
describe	Describe a component and its parameters
fit	Resample the data using the sampler.

continues on next page

Table 73 – continued from previous page

fit_transform	Fit and transform the data using the undersampler.
load	Loads component at file path
save	Saves component at file path
transform	No transformation needs to be done here.

evalml.pipelines.components.Undersampler.__init__

Undersampler.**__init__**(*sampling_ratio=0.25*, *sampling_ratio_dict=None*, *min_samples=100*, *min_percentage=0.1*, *random_seed=0*, ***kwargs*)

Initializes an undersampling transformer to downsample the majority classes in the dataset.

Parameters

- **sampling_ratio** (*float*) – The smallest minority:majority ratio that is accepted as 'balanced'. For instance, a 1:4 ratio would be represented as 0.25, while a 1:1 ratio is 1.0. Must be between 0 and 1, inclusive. Defaults to 0.25.
- **sampling_ratio_dict** (*dict*) – A dictionary specifying the desired balanced ratio for each target value. For instance, in a binary case where class 1 is the minority, we could specify: *sampling_ratio_dict={0: 0.5, 1: 1}*, which means we would undersample class 0 to have twice the number of samples as class 1 (minority:majority ratio = 0.5), and don't sample class 1. Overrides sampling_ratio if provided. Defaults to None.
- **min_samples** (*int*) – The minimum number of samples that we must have for any class, pre or post sampling. If a class must be downsampled, it will not be downsampled past this value. To determine severe imbalance, the minority class must occur less often than this and must have a class ratio below min_percentage. Must be greater than 0. Defaults to 100.
- **min_percentage** (*float*) – The minimum percentage of the minimum class to total dataset that we tolerate, as long as it is above min_samples. If min_percentage and min_samples are not met, treat this as severely imbalanced, and we will not resample the data. Must be between 0 and 0.5, inclusive. Defaults to 0.1.
- **random_seed** (*int*) – The seed to use for random sampling. Defaults to 0.

evalml.pipelines.components.Undersampler.clone

Undersampler.**clone**()

Constructs a new component with the same parameters and random state.

Returns A new instance of this component with identical parameters and random state.

evalml.pipelines.components.Undersampler.describe

Undersampler.**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.Undersampler.fit

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

Resample the data using the sampler. Since our sampler doesn't need to be fit, we do nothing here.

Parameters

- **X** (*pd.DataFrame*) – Training features
- **y** (*pd.Series*) – Target features

Returns self

evalml.pipelines.components.Undersampler.fit_transform

Undersampler.**fit_transform**(*X*, *y*)

Fit and transform the data using the undersampler. Used during training of the pipeline

Parameters

- **X** (*pd.DataFrame*) – Training features
- **y** – Target features

evalml.pipelines.components.Undersampler.load

static Undersampler.**load**(*file_path*)

Loads component at file path

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

Returns ComponentBase object

evalml.pipelines.components.Undersampler.save

Undersampler.**save**(*file_path*, *pickle_protocol=5*)

Saves component at file path

Parameters

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

Returns None

evalml.pipelines.components.Undersampler.transform

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

No transformation needs to be done here.

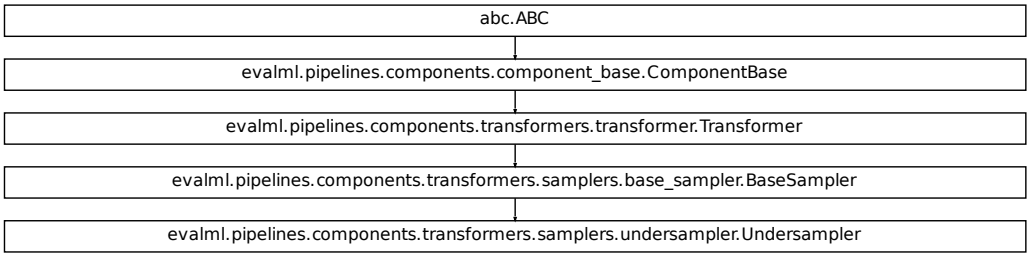
Parameters

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

Returns X and y data that was passed in.

Return type pd.DataFrame, pd.Series

Class Inheritance



evalml.pipelines.components.SMOTESampler

class evalml.pipelines.components.**SMOTESampler**(*sampling_ratio=0.25*, *k_neighbors_default=5*, *n_jobs=-1*, *random_seed=0*, ***kwargs*)

SMOTE Oversampler component. Works on numerical datasets only. This component is only run during training and not during predict.

name = 'SMOTE Oversampler'

model_family = 'none'

hyperparameter_ranges = {}

default_parameters = {'k_neighbors_default': 5, 'n_jobs': -1, 'sampling_ratio': 0.25,

Instance attributes

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

Methods:

__init__	Initializes the oversampler component.
clone	Constructs a new component with the same parameters and random state.
describe	Describe a component and its parameters
fit	Fits the Oversampler to the data.
fit_transform	Fit and transform the data using the data sampler.
load	Loads component at file path
save	Saves component at file path
transform	No transformation needs to be done here.

evalml.pipelines.components.SMOTESampler.__init__

SMOTESampler.**__init__**(*sampling_ratio=0.25*, *k_neighbors_default=5*, *n_jobs=-1*, *random_seed=0*, ***kwargs*)

Initializes the oversampler component.

Parameters

- **sampling_ratio** (*float*) – This is the goal ratio of the minority to majority class, with range (0, 1]. A value of 0.25 means we want a 1:4 ratio of the minority to majority class after oversampling. We will create the a sampling dictionary using this ratio, with the keys corresponding to the class and the values responding to the number of samples. Defaults to 0.25.
- **k_neighbors_default** (*int*) – The number of nearest neighbors used to construct synthetic samples. This is the default value used, but the actual k_neighbors value might be smaller if there are less samples. Defaults to 5.
- **n_jobs** (*int*) – The number of CPU cores to use. Defaults to -1.

evalml.pipelines.components.SMOTESampler.clone

SMOTESampler.**clone**()

Constructs a new component with the same parameters and random state.

Returns A new instance of this component with identical parameters and random state.

evalml.pipelines.components.SMOTESampler.describe

SMOTESampler.**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.SMOTESampler.fit

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

Fits the Oversampler to the data.

Parameters

- **X** (*pd.DataFrame*) – Training features
- **y** (*pd.Series*) – Target features

Returns self

evalml.pipelines.components.SMOTESampler.fit_transform

SMOTESampler.**fit_transform**(*X*, *y*)

Fit and transform the data using the data sampler. Used during training of the pipeline

Parameters

- **X** (*pd.DataFrame*) – Training features
- **y** – Target features

evalml.pipelines.components.SMOTESampler.load

static SMOTESampler.**load**(*file_path*)

Loads component at file path

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

Returns ComponentBase object

evalml.pipelines.components.SMOTESampler.save

SMOTESampler.**save**(*file_path*, *pickle_protocol=5*)

Saves component at file path

Parameters

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

Returns None

evalml.pipelines.components.SMOTESampler.transform

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

No transformation needs to be done here.

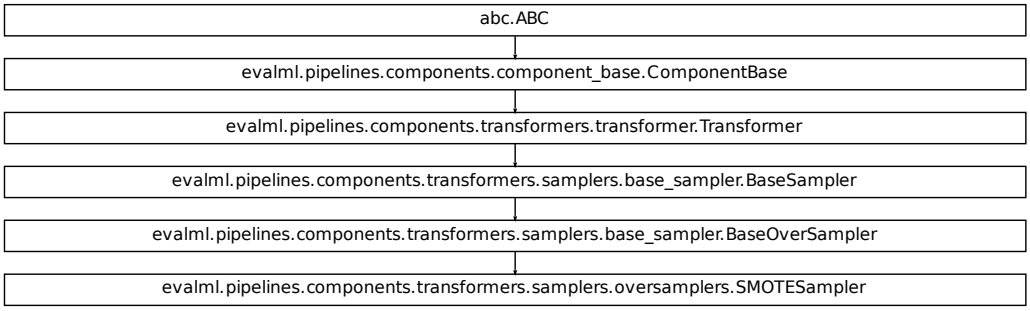
Parameters

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

Returns X and y data that was passed in.

Return type pd.DataFrame, pd.Series

Class Inheritance



evalml.pipelines.components.SMOTENCSampler

class evalml.pipelines.components.**SMOTENCSampler**(*sampling_ratio=0.25*, *k_neighbors_default=5*, *n_jobs=- 1*, *random_seed=0*, ***kwargs*)

SMOTENC Oversampler component. Uses SMOTENC to generate synthetic samples. Works on a mix of nomerical and categorical columns. Input data must be Woodwork type, and this component is only run during training and not during predict.

name = 'SMOTENC Oversampler'

model_family = 'none'

hyperparameter_ranges = {}

default_parameters = {'k_neighbors_default': 5, 'n_jobs': -1, 'sampling_ratio': 0.25,

Instance attributes

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

Methods:

__init__	Initializes the oversampler component.
clone	Constructs a new component with the same parameters and random state.
describe	Describe a component and its parameters
fit	Fits the Oversampler to the data.
fit_transform	Fit and transform the data using the data sampler.
load	Loads component at file path
save	Saves component at file path
transform	No transformation needs to be done here.

evalml.pipelines.components.SMOTENCSampler.__init__

SMOTENCSampler.**__init__**(*sampling_ratio=0.25*, *k_neighbors_default=5*, *n_jobs=- 1*, *random_seed=0*, ***kwargs*)

Initializes the oversampler component.

Parameters

- **sampling_ratio** (*float*) – This is the goal ratio of the minority to majority class, with range (0, 1]. A value of 0.25 means we want a 1:4 ratio of the minority to majority class after oversampling. We will create the a sampling dictionary using this ratio, with the keys corresponding to the class and the values responding to the number of samples. Defaults to 0.25.
- **k_neighbors_default** (*int*) – The number of nearest neighbors used to construct synthetic samples. This is the default value used, but the actual k_neighbors value might be smaller if there are less samples. Defaults to 5.

- **n_jobs** (*int*) – The number of CPU cores to use. Defaults to -1.

evalml.pipelines.components.SMOTENCSampler.clone

SMOTENCSampler.**clone**()

Constructs a new component with the same parameters and random state.

Returns A new instance of this component with identical parameters and random state.

evalml.pipelines.components.SMOTENCSampler.describe

SMOTENCSampler.**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.SMOTENCSampler.fit

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

Fits the Oversampler to the data.

Parameters

- **X** (*pd.DataFrame*) – Training features
- **y** (*pd.Series*) – Target features

Returns self

evalml.pipelines.components.SMOTENCSampler.fit_transform

SMOTENCSampler.**fit_transform**(*X*, *y*)

Fit and transform the data using the data sampler. Used during training of the pipeline

Parameters

- **X** (*pd.DataFrame*) – Training features
- **y** – Target features

evalml.pipelines.components.SMOTENCSampler.load

static SMOTENCSampler.**load**(*file_path*)

Loads component at file path

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

Returns ComponentBase object

evalml.pipelines.components.SMOTENCSampler.save

SMOTENCSampler.**save**(*file_path*, *pickle_protocol=5*)

Saves component at file path

Parameters

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

Returns None

evalml.pipelines.components.SMOTENCSampler.transform

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

No transformation needs to be done here.

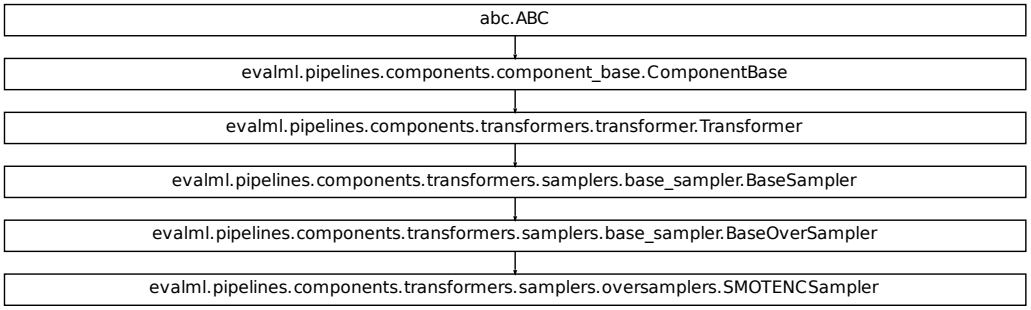
Parameters

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

Returns X and y data that was passed in.

Return type pd.DataFrame, pd.Series

Class Inheritance



evalml.pipelines.components.SMOTENSampler

class evalml.pipelines.components.**SMOTENSampler**(*sampling_ratio=0.25*, *k_neighbors_default=5*, *n_jobs=-1*, *random_seed=0*, ***kwargs*)

SMOTEN Oversampler component. Uses SMOTEN to generate synthetic samples. Works for purely categorical datasets. This component is only run during training and not during predict.

name = 'SMOTEN Oversampler'

model_family = 'none'

hyperparameter_ranges = {}

default_parameters = {'k_neighbors_default': 5, 'n_jobs': -1, 'sampling_ratio': 0.25,

Instance attributes

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

Methods:

__init__	Initializes the oversampler component.
clone	Constructs a new component with the same parameters and random state.
describe	Describe a component and its parameters
fit	Fits the Oversampler to the data.
fit_transform	Fit and transform the data using the data sampler.
load	Loads component at file path
save	Saves component at file path
transform	No transformation needs to be done here.

evalml.pipelines.components.SMOTENSampler.__init__

SMOTENSampler.**__init__**(*sampling_ratio=0.25*, *k_neighbors_default=5*, *n_jobs=-1*, *random_seed=0*, ***kwargs*)

Initializes the oversampler component.

Parameters

- **sampling_ratio** (*float*) – This is the goal ratio of the minority to majority class, with range (0, 1]. A value of 0.25 means we want a 1:4 ratio of the minority to majority class after oversampling. We will create the a sampling dictionary using this ratio, with the keys corresponding to the class and the values responding to the number of samples. Defaults to 0.25.
- **k_neighbors_default** (*int*) – The number of nearest neighbors used to construct synthetic samples. This is the default value used, but the actual k_neighbors value might be smaller if there are less samples. Defaults to 5.

- **n_jobs** (*int*) – The number of CPU cores to use. Defaults to -1.

evalml.pipelines.components.SMOTENSampler.clone

SMOTENSampler.**clone**()

Constructs a new component with the same parameters and random state.

Returns A new instance of this component with identical parameters and random state.

evalml.pipelines.components.SMOTENSampler.describe

SMOTENSampler.**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.SMOTENSampler.fit

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

Fits the Oversampler to the data.

Parameters

- **X** (*pd.DataFrame*) – Training features
- **y** (*pd.Series*) – Target features

Returns self

evalml.pipelines.components.SMOTENSampler.fit_transform

SMOTENSampler.**fit_transform**(*X*, *y*)

Fit and transform the data using the data sampler. Used during training of the pipeline

Parameters

- **X** (*pd.DataFrame*) – Training features
- **y** – Target features

evalml.pipelines.components.SMOTENSampler.load

static SMOTENSampler.**load**(*file_path*)

Loads component at file path

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

Returns ComponentBase object

evalml.pipelines.components.SMOTENSampler.save

SMOTENSampler.**save**(*file_path*, *pickle_protocol=5*)

Saves component at file path

Parameters

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

Returns None

evalml.pipelines.components.SMOTENSampler.transform

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

No transformation needs to be done here.

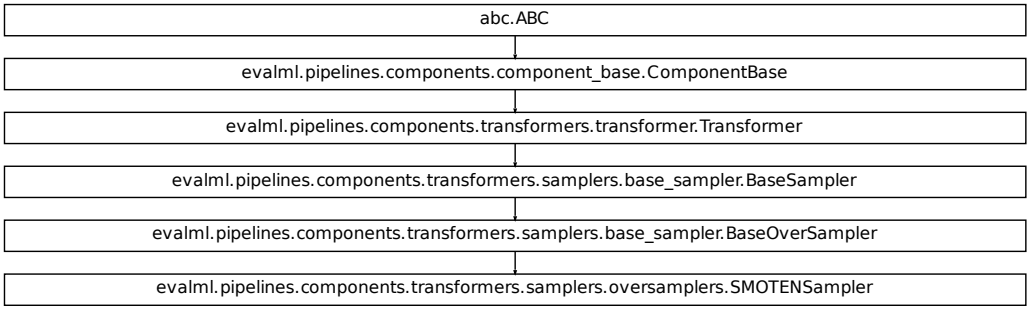
Parameters

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

Returns X and y data that was passed in.

Return type pd.DataFrame, pd.Series

Class Inheritance



5.6.4 Estimators

Classifiers

Classifiers are components that output a predicted class label.

CatBoostClassifier	CatBoost Classifier, a classifier that uses gradient-boosting on decision trees.
ElasticNetClassifier	Elastic Net Classifier.
ExtraTreesClassifier	Extra Trees Classifier.
RandomForestClassifier	Random Forest Classifier.
LightGBMClassifier	LightGBM Classifier
LogisticRegressionClassifier	Logistic Regression Classifier.
XGBoostClassifier	XGBoost Classifier.
BaselineClassifier	Classifier that predicts using the specified strategy.
StackedEnsembleClassifier	Stacked Ensemble Classifier.
DecisionTreeClassifier	Decision Tree Classifier.
KNeighborsClassifier	K-Nearest Neighbors Classifier.
SVMClassifier	Support Vector Machine Classifier.

evalml.pipelines.components.CatBoostClassifier

class evalml.pipelines.components.**CatBoostClassifier**(*n_estimators=10*, *eta=0.03*, *max_depth=6*, *bootstrap_type=None*, *silent=True*, *allow_writing_files=False*, *random_seed=0*, *n_jobs=- 1*, ***kwargs*)

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

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

name = 'CatBoost Classifier'

model_family = 'catboost'

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

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

default_parameters = {'allow_writing_files': False, 'bootstrap_type': None, 'eta': 0.0

predict_uses_y = False

Instance attributes

feature_importance	Return an attribute of instance, which is of type owner.
needs_fitting	
parameters	Returns the parameters which were used to initialize the component

Methods:

__init__	CatBoost Classifier.
clone	Constructs a new component with the same parameters and random state.
describe	Describe a component and its parameters
fit	Fits component to data
load	Loads component at file path
predict	Make predictions using selected features.
predict_proba	Make probability estimates for labels.
save	Saves component at file path

evalml.pipelines.components.CatBoostClassifier.__init__

CatBoostClassifier.**__init__**(*n_estimators=10*, *eta=0.03*, *max_depth=6*, *bootstrap_type=None*, *silent=True*, *allow_writing_files=False*, *random_seed=0*, *n_jobs=- 1*, ***kwargs*)

CatBoost Classifier.

Parameters

- **n_estimators** (*int*) – Number of gradient boosted trees. Equivalent to number of boosting rounds. Defaults to 100.
- **eta** (*float*) – Learning rate. Defaults to 0.1.
- **max_depth** (*int*) – Maximum tree depth for base learners. Defaults to 6.
- **bootstrap_type** (*string*) – Defines the method for sampling the weights of objects. Defaults to None.
- **silent** (*bool*) – Whether to emit logging while training. Default to False.
- **allow_writing_files** (*bool*) – Whether to allow writing of analytical and snapshot files during training. Defaults to False.
- **random_seed** (*int*) – Seed for the random number generator. Defaults to 0.
- **n_jobs** (*int*) – Number of parallel threads used to run CatBoost. This will be passed to CatBoost as the *thread_count* parameter. Defaults to -1.

evalml.pipelines.components.CatBoostClassifier.clone

CatBoostClassifier.**clone**()

Constructs a new component with the same parameters and random state.

Returns A new instance of this component with identical parameters and random state.

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

Returns self

evalml.pipelines.components.CatBoostClassifier.load

static CatBoostClassifier.**load**(*file_path*)

Loads component at file path

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

Returns ComponentBase object

evalml.pipelines.components.CatBoostClassifier.predict

CatBoostClassifier.**predict**(*X*)

Make predictions using selected features.

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

Returns Predicted values

Return type pd.Series

evalml.pipelines.components.CatBoostClassifier.predict_proba

CatBoostClassifier.**predict_proba**(*X*)

Make probability estimates for labels.

Parameters **X** (*pd.DataFrame, or np.ndarray*) – Features

Returns Probability estimates

Return type pd.Series

evalml.pipelines.components.CatBoostClassifier.save

CatBoostClassifier.**save**(*file_path*, *pickle_protocol=5*)

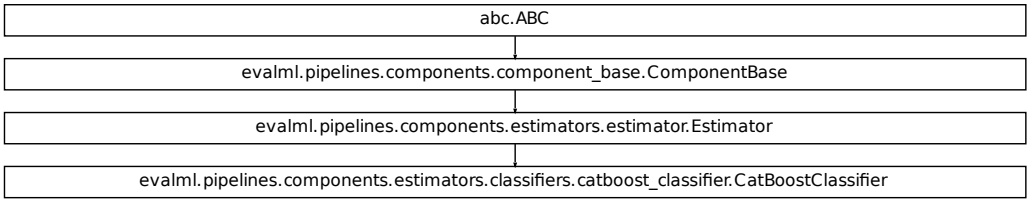
Saves component at file path

Parameters

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

Returns None

Class Inheritance



evalml.pipelines.components.ElasticNetClassifier

class evalml.pipelines.components.**ElasticNetClassifier**(*penalty='elasticnet', C=1.0, l1_ratio=0.15, n_jobs=-1, multi_class='auto', solver='saga', random_seed=0, **kwargs*)

Elastic Net Classifier. Uses Logistic Regression with elasticnet penalty as the base estimator.

name = 'Elastic Net Classifier'

model_family = 'linear_model'

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

hyperparameter_ranges = {'C': Real(low=0.01, high=10, prior='uniform', transform='iden

default_parameters = {'C': 1.0, 'l1_ratio': 0.15, 'multi_class': 'auto', 'n_jobs': -1,

predict_uses_y = False

Instance attributes

feature_importance	Return an attribute of instance, which is of type owner.
needs_fitting	
parameters	Returns the parameters which were used to initialize the component

Methods:

__init__	Initialize self.
clone	Constructs a new component with the same parameters and random state.
describe	Describe a component and its parameters
fit	Fits component to data
load	Loads component at file path
predict	Make predictions using selected features.
predict_proba	Make probability estimates for labels.
save	Saves component at file path

evalml.pipelines.components.ElasticNetClassifier.__init__

ElasticNetClassifier.**__init__**(*penalty='elasticnet', C=1.0, l1_ratio=0.15, n_jobs=-1, multi_class='auto', solver='saga', random_seed=0, **kwargs*)

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

evalml.pipelines.components.ElasticNetClassifier.clone

ElasticNetClassifier.**clone**()

Constructs a new component with the same parameters and random state.

Returns A new instance of this component with identical parameters and random state.

evalml.pipelines.components.ElasticNetClassifier.describe

ElasticNetClassifier.**describe**(*print_name=False*, *return_dict=False*)

Describe a component and its parameters

Parameters

- **print_name** (*bool, optional*) – whether to print name of component
- **return_dict** (*bool, optional*) – whether to return description as dictionary in the format {"name": name, "parameters": parameters}

Returns prints and returns dictionary

Return type None or dict

evalml.pipelines.components.ElasticNetClassifier.fit

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

Fits component to data

Parameters

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

Returns self

evalml.pipelines.components.ElasticNetClassifier.load

static ElasticNetClassifier.**load**(*file_path*)

Loads component at file path

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

Returns ComponentBase object

evalml.pipelines.components.ElasticNetClassifier.predict

ElasticNetClassifier.**predict**(*X*)

Make predictions using selected features.

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

Returns Predicted values

Return type pd.Series

evalml.pipelines.components.ElasticNetClassifier.predict_proba

ElasticNetClassifier.**predict_proba**(*X*)

Make probability estimates for labels.

Parameters **X** (*pd.DataFrame, or np.ndarray*) – Features

Returns Probability estimates

Return type pd.Series

evalml.pipelines.components.ElasticNetClassifier.save

ElasticNetClassifier.**save**(*file_path*, *pickle_protocol=5*)

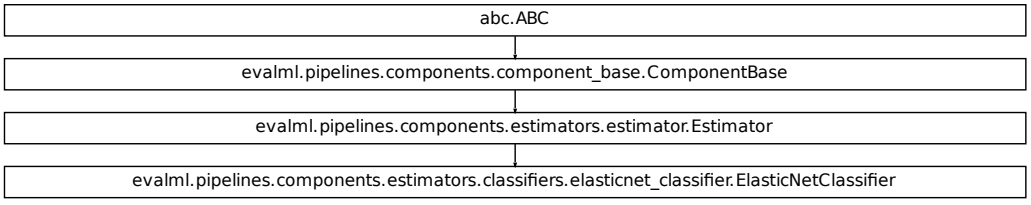
Saves component at file path

Parameters

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

Returns None

Class Inheritance



evalml.pipelines.components.ExtraTreesClassifier

class evalml.pipelines.components.**ExtraTreesClassifier**(*n_estimators=100*, *max_features='auto'*, *max_depth=6*, *min_samples_split=2*, *min_weight_fraction_leaf=0.0*, *n_jobs=-1*, *random_seed=0*, ***kwargs*)

Extra Trees Classifier.

name = 'Extra Trees Classifier'

model_family = 'extra_trees'

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

hyperparameter_ranges = {'max_depth': Integer(low=4, high=10, prior='uniform', transfo

default_parameters = {'max_depth': 6, 'max_features': 'auto', 'min_samples_split': 2,

predict_uses_y = False

Instance attributes

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

Methods:

__init__	Initialize self.
clone	Constructs a new component with the same parameters and random state.
describe	Describe a component and its parameters
fit	Fits component to data
load	Loads component at file path
predict	Make predictions using selected features.
predict_proba	Make probability estimates for labels.
save	Saves component at file path

evalml.pipelines.components.ExtraTreesClassifier.__init__

ExtraTreesClassifier.**__init__**(*n_estimators=100*, *max_features='auto'*, *max_depth=6*, *min_samples_split=2*, *min_weight_fraction_leaf=0.0*, *n_jobs=- 1*, *random_seed=0*, ***kwargs*)

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

evalml.pipelines.components.ExtraTreesClassifier.clone

ExtraTreesClassifier.**clone**()

Constructs a new component with the same parameters and random state.

Returns A new instance of this component with identical parameters and random state.

evalml.pipelines.components.ExtraTreesClassifier.describe

ExtraTreesClassifier.**describe**(*print_name=False*, *return_dict=False*)

Describe a component and its parameters

Parameters

- **print_name** (*bool, optional*) – whether to print name of component
- **return_dict** (*bool, optional*) – whether to return description as dictionary in the format {"name": name, "parameters": parameters}

Returns prints and returns dictionary

Return type None or dict

evalml.pipelines.components.ExtraTreesClassifier.fit

ExtraTreesClassifier.**fit**(*X*, *y=None*)

Fits component to data

Parameters

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

Returns self

evalml.pipelines.components.ExtraTreesClassifier.load

static ExtraTreesClassifier.**load**(*file_path*)

Loads component at file path

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

Returns ComponentBase object

evalml.pipelines.components.ExtraTreesClassifier.predict

ExtraTreesClassifier.**predict**(*X*)

Make predictions using selected features.

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

Returns Predicted values

Return type pd.Series

evalml.pipelines.components.ExtraTreesClassifier.predict_proba

ExtraTreesClassifier.**predict_proba**(*X*)

Make probability estimates for labels.

Parameters **X** (*pd.DataFrame, or np.ndarray*) – Features

Returns Probability estimates

Return type pd.Series

evalml.pipelines.components.ExtraTreesClassifier.save

ExtraTreesClassifier.**save**(*file_path*, *pickle_protocol=5*)

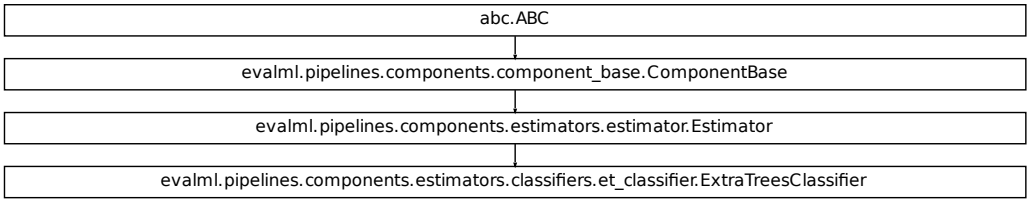
Saves component at file path

Parameters

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

Returns None

Class Inheritance



evalml.pipelines.components.RandomForestClassifier

class evalml.pipelines.components.**RandomForestClassifier**(*n_estimators=100*, *max_depth=6*, *n_jobs=-1*, *random_seed=0*, ***kwargs*)

Random Forest Classifier.

name = 'Random Forest Classifier'

model_family = 'random_forest'

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

hyperparameter_ranges = {'max_depth': Integer(low=1, high=10, prior='uniform', transfo

default_parameters = {'max_depth': 6, 'n_estimators': 100, 'n_jobs': -1}

predict_uses_y = False

Instance attributes

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

Methods:

__init__	Initialize self.
clone	Constructs a new component with the same parameters and random state.
describe	Describe a component and its parameters
fit	Fits component to data
load	Loads component at file path
predict	Make predictions using selected features.
predict_proba	Make probability estimates for labels.
save	Saves component at file path

evalml.pipelines.components.RandomForestClassifier.__init__

RandomForestClassifier.**__init__**(*n_estimators=100*, *max_depth=6*, *n_jobs=-1*, *random_seed=0*, ***kwargs*)

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

evalml.pipelines.components.RandomForestClassifier.clone

RandomForestClassifier.**clone**()

Constructs a new component with the same parameters and random state.

Returns A new instance of this component with identical parameters and random state.

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

Returns self

evalml.pipelines.components.RandomForestClassifier.load

static RandomForestClassifier.**load**(*file_path*)

Loads component at file path

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

Returns ComponentBase object

evalml.pipelines.components.RandomForestClassifier.predict

RandomForestClassifier.**predict**(*X*)

Make predictions using selected features.

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

Returns Predicted values

Return type pd.Series

evalml.pipelines.components.RandomForestClassifier.predict_proba

RandomForestClassifier.**predict_proba**(*X*)

Make probability estimates for labels.

Parameters **X** (*pd.DataFrame, or np.ndarray*) – Features

Returns Probability estimates

Return type pd.Series

evalml.pipelines.components.RandomForestClassifier.save

RandomForestClassifier.**save**(*file_path*, *pickle_protocol=5*)

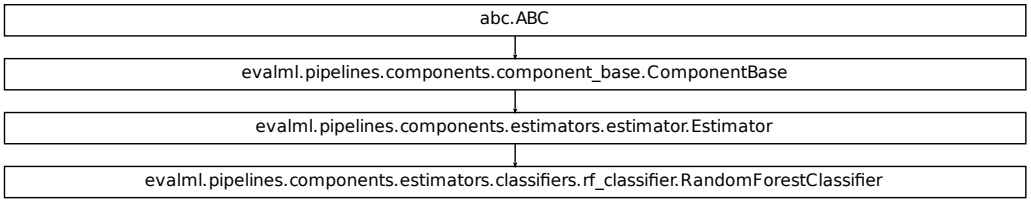
Saves component at file path

Parameters

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

Returns None

Class Inheritance



evalml.pipelines.components.LightGBMClassifier

class evalml.pipelines.components.**LightGBMClassifier**(*boosting_type='gbdt', learning_rate=0.1, n_estimators=100, max_depth=0, num_leaves=31, min_child_samples=20, n_jobs=- 1, random_seed=0, bagging_fraction=0.9, bagging_freq=0, **kwargs*)

LightGBM Classifier

name = 'LightGBM Classifier'

model_family = 'lightgbm'

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

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

default_parameters = {'bagging_fraction': 0.9, 'bagging_freq': 0, 'boosting_type': 'gb

predict_uses_y = False

Instance attributes

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

Methods:

__init__	Initialize self.
clone	Constructs a new component with the same parameters and random state.
describe	Describe a component and its parameters
fit	Fits component to data
load	Loads component at file path
predict	Make predictions using selected features.
predict_proba	Make probability estimates for labels.
save	Saves component at file path

evalml.pipelines.components.LightGBMClassifier.__init__

LightGBMClassifier.**__init__**(*boosting_type='gbdt'*, *learning_rate=0.1*, *n_estimators=100*, *max_depth=0*, *num_leaves=31*, *min_child_samples=20*, *n_jobs=- 1*, *random_seed=0*, *bagging_fraction=0.9*, *bagging_freq=0*, ***kwargs*)

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

evalml.pipelines.components.LightGBMClassifier.clone

LightGBMClassifier.**clone**()

Constructs a new component with the same parameters and random state.

Returns A new instance of this component with identical parameters and random state.

evalml.pipelines.components.LightGBMClassifier.describe

LightGBMClassifier.**describe**(*print_name=False*, *return_dict=False*)

Describe a component and its parameters

Parameters

- **print_name** (*bool, optional*) – whether to print name of component
- **return_dict** (*bool, optional*) – whether to return description as dictionary in the format {"name": name, "parameters": parameters}

Returns prints and returns dictionary

Return type None or dict

evalml.pipelines.components.LightGBMClassifier.fit

LightGBMClassifier.**fit**(*X*, *y=None*)

Fits component to data

Parameters

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

Returns self

evalml.pipelines.components.LightGBMClassifier.load

static LightGBMClassifier.**load**(*file_path*)
Loads component at file path

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

Returns ComponentBase object

evalml.pipelines.components.LightGBMClassifier.predict

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

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

Returns Predicted values

Return type pd.Series

evalml.pipelines.components.LightGBMClassifier.predict_proba

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

Parameters **X** (*pd.DataFrame, or np.ndarray*) – Features

Returns Probability estimates

Return type pd.Series

evalml.pipelines.components.LightGBMClassifier.save

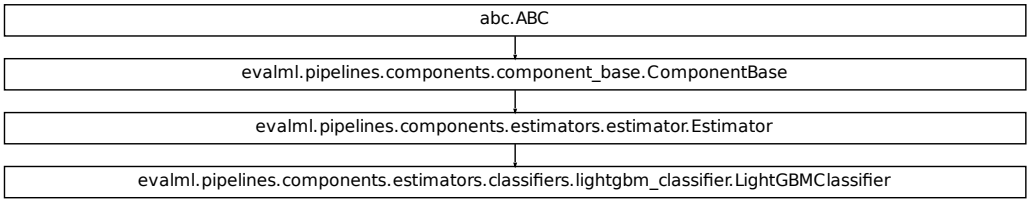
LightGBMClassifier.**save**(*file_path*, *pickle_protocol=5*)
Saves component at file path

Parameters

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

Returns None

Class Inheritance



evalml.pipelines.components.LogisticRegressionClassifier

class evalml.pipelines.components.**LogisticRegressionClassifier**(*penalty='l2'*, *C=1.0*, *n_jobs=-1*, *multi_class='auto'*, *solver='lbfgs'*, *random_seed=0*, ***kwargs*)

Logistic Regression Classifier.

name = 'Logistic Regression Classifier'

model_family = 'linear_model'

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

hyperparameter_ranges = {'C': Real(low=0.01, high=10, prior='uniform', transform='iden

default_parameters = {'C': 1.0, 'multi_class': 'auto', 'n_jobs': -1, 'penalty': 'l2',

predict_uses_y = False

Instance attributes

feature_importance	Return an attribute of instance, which is of type owner.
needs_fitting	
parameters	Returns the parameters which were used to initialize the component

Methods:

__init__	Initialize self.
clone	Constructs a new component with the same parameters and random state.
describe	Describe a component and its parameters
fit	Fits component to data
load	Loads component at file path
predict	Make predictions using selected features.
predict_proba	Make probability estimates for labels.
save	Saves component at file path

evalml.pipelines.components.LogisticRegressionClassifier.__init__

LogisticRegressionClassifier.**__init__**(*penalty='l2'*, *C=1.0*, *n_jobs=-1*, *multi_class='auto'*, *solver='lbfgs'*, *random_seed=0*, ***kwargs*)

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

evalml.pipelines.components.LogisticRegressionClassifier.clone

LogisticRegressionClassifier.**clone**()

Constructs a new component with the same parameters and random state.

Returns A new instance of this component with identical parameters and random state.

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** (*list, pd.DataFrame or np.ndarray*) – The input training data of shape [n_samples, n_features]

- **y** (*list, pd.Series, np.ndarray, optional*) – The target training data of length [n_samples]

Returns self

evalml.pipelines.components.LogisticRegressionClassifier.load

static LogisticRegressionClassifier.**load**(*file_path*)

Loads component at file path

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

Returns ComponentBase object

evalml.pipelines.components.LogisticRegressionClassifier.predict

LogisticRegressionClassifier.**predict**(*X*)

Make predictions using selected features.

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

Returns Predicted values

Return type pd.Series

evalml.pipelines.components.LogisticRegressionClassifier.predict_proba

LogisticRegressionClassifier.**predict_proba**(*X*)

Make probability estimates for labels.

Parameters **X** (*pd.DataFrame, or np.ndarray*) – Features

Returns Probability estimates

Return type pd.Series

evalml.pipelines.components.LogisticRegressionClassifier.save

LogisticRegressionClassifier.**save**(*file_path*, *pickle_protocol=5*)

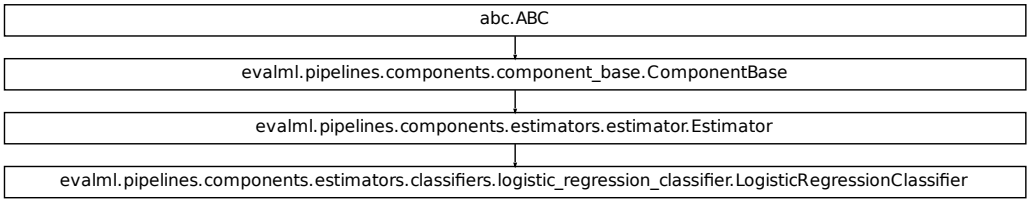
Saves component at file path

Parameters

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

Returns None

Class Inheritance



evalml.pipelines.components.XGBoostClassifier

class evalml.pipelines.components.**XGBoostClassifier**(*eta=0.1*, *max_depth=6*, *min_child_weight=1*, *n_estimators=100*, *random_seed=0*, *n_jobs=-1*, ***kwargs*)

XGBoost Classifier.

name = 'XGBoost Classifier'

model_family = 'xgboost'

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

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

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

predict_uses_y = False

Instance attributes

SEED_MAX	
SEED_MIN	
feature_importance	Return an attribute of instance, which is of type owner.
needs_fitting	
parameters	Returns the parameters which were used to initialize the component

Methods:

__init__	XGBoost Classifier.
clone	Constructs a new component with the same parameters and random state.
describe	Describe a component and its parameters
fit	Fits component to data
load	Loads component at file path
predict	Make predictions using selected features.
predict_proba	Make probability estimates for labels.
save	Saves component at file path

evalml.pipelines.components.XGBoostClassifier.__init__

XGBoostClassifier.**__init__**(*eta=0.1*, *max_depth=6*, *min_child_weight=1*, *n_estimators=100*, *random_seed=0*, *n_jobs=- 1*, ***kwargs*)

XGBoost Classifier.

Parameters

- **eta** (*float*) – Learning rate. Defaults to 0.1.
- **max_depth** (*int*) – Maximum tree depth for base learners. Defaults to 6.
- **min_child_weight** (*float*) – Minimum sum of instance weight(hessian) needed in a child. Defaults to 1.
- **n_estimators** (*int*) – Number of gradient boosted trees. Equivalent to number of boosting rounds. Defaults to 100.
- **random_seed** (*int*) – Seed for the random number generator. Defaults to 0.
- **n_jobs** (*int*) – Number of parallel threads used to run xgboost. Note that creating thread contention will significantly slow down the algorithm. Defaults to -1.

evalml.pipelines.components.XGBoostClassifier.clone

XGBoostClassifier.**clone**()

Constructs a new component with the same parameters and random state.

Returns A new instance of this component with identical parameters and random state.

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

Returns self

evalml.pipelines.components.XGBoostClassifier.load

static XGBoostClassifier.**load**(*file_path*)

Loads component at file path

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

Returns ComponentBase object

evalml.pipelines.components.XGBoostClassifier.predict

XGBoostClassifier.**predict**(*X*)

Make predictions using selected features.

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

Returns Predicted values

Return type pd.Series

evalml.pipelines.components.XGBoostClassifier.predict_proba

XGBoostClassifier.**predict_proba**(*X*)

Make probability estimates for labels.

Parameters **X** (*pd.DataFrame, or np.ndarray*) – Features

Returns Probability estimates

Return type pd.Series

evalml.pipelines.components.XGBoostClassifier.save

XGBoostClassifier.**save**(*file_path*, *pickle_protocol=5*)

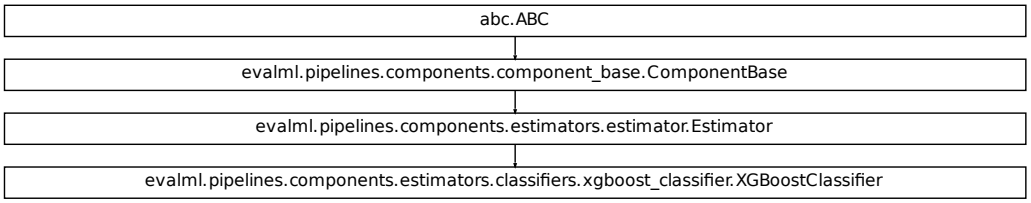
Saves component at file path

Parameters

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

Returns None

Class Inheritance



evalml.pipelines.components.BaselineClassifier

class evalml.pipelines.components.**BaselineClassifier**(*strategy='mode'*, *random_seed=0*, ***kwargs*)

Classifier that predicts using the specified strategy.

This is useful as a simple baseline classifier to compare with other classifiers.

name = 'Baseline Classifier'

model_family = 'baseline'

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

hyperparameter_ranges = {}

default_parameters = {'strategy': 'mode'}

predict_uses_y = False

Instance attributes

classes_	Returns class labels.
feature_importance	Returns importance associated with each feature.
needs_fitting	
parameters	Returns the parameters which were used to initialize the component

Methods:

__init__	Baseline classifier that uses a simple strategy to make predictions.
clone	Constructs a new component with the same parameters and random state.
describe	Describe a component and its parameters
fit	Fits component to data
load	Loads component at file path
predict	Make predictions using selected features.
predict_proba	Make probability estimates for labels.
save	Saves component at file path

evalml.pipelines.components.BaselineClassifier.__init__

BaselineClassifier.**__init__**(*strategy='mode'*, *random_seed=0*, ***kwargs*)

Baseline classifier that uses a simple strategy to make predictions.

Parameters

- **strategy** (*str*) – Method used to predict. Valid options are "mode", "random" and "random_weighted". Defaults to "mode".
- **random_seed** (*int*) – Seed for the random number generator. Defaults to 0.

evalml.pipelines.components.BaselineClassifier.clone

BaselineClassifier.**clone**()

Constructs a new component with the same parameters and random state.

Returns A new instance of this component with identical parameters and random state.

evalml.pipelines.components.BaselineClassifier.describe

BaselineClassifier.**describe**(*print_name=False*, *return_dict=False*)

Describe a component and its parameters

Parameters

- **print_name** (*bool, optional*) – whether to print name of component
- **return_dict** (*bool, optional*) – whether to return description as dictionary in the format {"name": name, "parameters": parameters}

Returns prints and returns dictionary

Return type None or dict

evalml.pipelines.components.BaselineClassifier.fit

BaselineClassifier.**fit**(*X*, *y=None*)

Fits component to data

Parameters

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

Returns self

evalml.pipelines.components.BaselineClassifier.load

static BaselineClassifier.**load**(*file_path*)

Loads component at file path

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

Returns ComponentBase object

evalml.pipelines.components.BaselineClassifier.predict

BaselineClassifier.**predict**(*X*)

Make predictions using selected features.

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

Returns Predicted values

Return type pd.Series

evalml.pipelines.components.BaselineClassifier.predict_proba

BaselineClassifier.**predict_proba**(*X*)

Make probability estimates for labels.

Parameters **X** (*pd.DataFrame, or np.ndarray*) – Features

Returns Probability estimates

Return type pd.Series

evalml.pipelines.components.BaselineClassifier.save

BaselineClassifier.**save**(*file_path*, *pickle_protocol=5*)

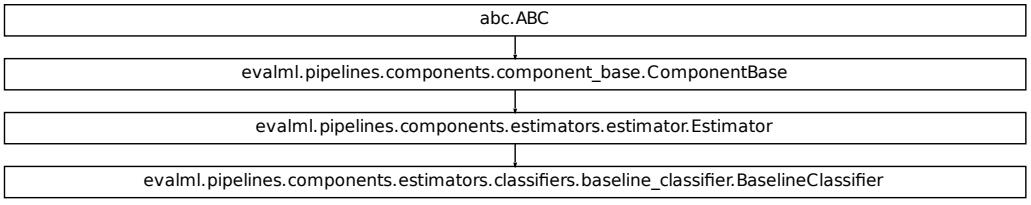
Saves component at file path

Parameters

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

Returns None

Class Inheritance



evalml.pipelines.components.StackedEnsembleClassifier

class evalml.pipelines.components.**StackedEnsembleClassifier**(*input_pipelines=None*, *final_estimator=None*, *cv=None*, *n_jobs=-1*, *random_seed=0*, ***kwargs*)

Stacked Ensemble Classifier.

name = 'Stacked Ensemble Classifier'

model_family = 'ensemble'

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

hyperparameter_ranges = {}

default_parameters = {'cv': None, 'final_estimator': None, 'n_jobs': -1}

predict_uses_y = False

Instance attributes

feature_importance	Not implemented for StackedEnsembleClassifier and StackedEnsembleRegressor
needs_fitting	
parameters	Returns the parameters which were used to initialize the component

Methods:

__init__	Stacked ensemble classifier.
clone	Constructs a new component with the same parameters and random state.
describe	Describe a component and its parameters
fit	Fits component to data
load	Loads component at file path
predict	Make predictions using selected features.
predict_proba	Make probability estimates for labels.
save	Saves component at file path

evalml.pipelines.components.StackedEnsembleClassifier.__init__

StackedEnsembleClassifier.**__init__**(*input_pipelines=None*, *final_estimator=None*, *cv=None*, *n_jobs=-1*, *random_seed=0*, ***kwargs*)

Stacked ensemble classifier.

Parameters

- **input_pipelines** (*list(PipelineBase or subclass obj)*) – List of pipeline instances to use as the base estimators. This must not be None or an empty list or else EnsembleMissingPipelinesError will be raised.
- **final_estimator** (*Estimator or subclass*) – The classifier used to combine the base estimators. If None, uses LogisticRegressionClassifier.
- **cv** (*int, cross-validation generator or an iterable*) – Determines the cross-validation splitting strategy used to train final_estimator. For int/None inputs, if the estimator is a classifier and y is either binary or multiclass, StratifiedKFold is used. Defaults to None. Possible inputs for cv are:
  - None: 3-fold cross validation
  - int: the number of folds in a (Stratified) KFold
  - An scikit-learn cross-validation generator object
  - An iterable yielding (train, test) splits
- **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. Defaults to None. - Note: there could be some multi-process errors thrown for values of *n_jobs != 1*. If this is the case, please use *n_jobs = 1*.
- **random_seed** (*int*) – Seed for the random number generator. Defaults to 0.

evalml.pipelines.components.StackedEnsembleClassifier.clone

StackedEnsembleClassifier.**clone**()

Constructs a new component with the same parameters and random state.

Returns A new instance of this component with identical parameters and random state.

evalml.pipelines.components.StackedEnsembleClassifier.describe

StackedEnsembleClassifier.**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.StackedEnsembleClassifier.fit

StackedEnsembleClassifier.**fit**(*X*, *y=None*)

Fits component to data

Parameters

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

Returns self

evalml.pipelines.components.StackedEnsembleClassifier.load

static StackedEnsembleClassifier.**load**(*file_path*)

Loads component at file path

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

Returns ComponentBase object

evalml.pipelines.components.StackedEnsembleClassifier.predict

StackedEnsembleClassifier.**predict**(*X*)

Make predictions using selected features.

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

Returns Predicted values

Return type pd.Series

evalml.pipelines.components.StackedEnsembleClassifier.predict_proba

StackedEnsembleClassifier.**predict_proba**(*X*)

Make probability estimates for labels.

Parameters **X** (*pd.DataFrame, or np.ndarray*) – Features

Returns Probability estimates

Return type pd.Series

evalml.pipelines.components.StackedEnsembleClassifier.save

StackedEnsembleClassifier.**save**(*file_path*, *pickle_protocol=5*)

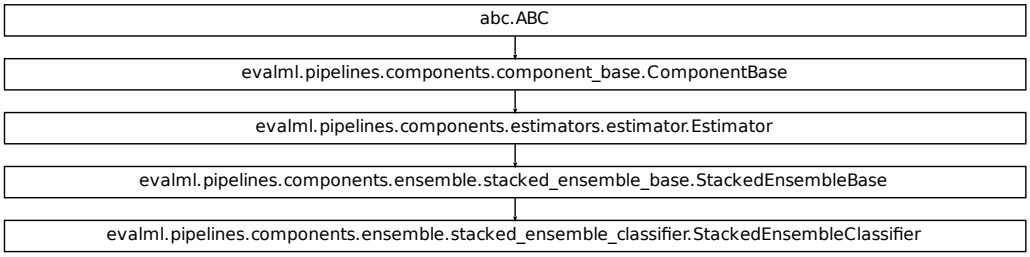
Saves component at file path

Parameters

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

Returns None

Class Inheritance



evalml.pipelines.components.DecisionTreeClassifier

class evalml.pipelines.components.**DecisionTreeClassifier**(*criterion='gini'*, *max_features='auto'*, *max_depth=6*, *min_samples_split=2*, *min_weight_fraction_leaf=0.0*, *random_seed=0*, ***kwargs*)

Decision Tree Classifier.

name = 'Decision Tree Classifier'

model_family = 'decision_tree'

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

hyperparameter_ranges = {'criterion': ['gini', 'entropy'], 'max_depth': Integer(low=4,

default_parameters = {'criterion': 'gini', 'max_depth': 6, 'max_features': 'auto', 'mi

predict_uses_y = False

Instance attributes

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

Methods:

__init__	Initialize self.
clone	Constructs a new component with the same parameters and random state.
describe	Describe a component and its parameters
fit	Fits component to data
load	Loads component at file path
predict	Make predictions using selected features.
predict_proba	Make probability estimates for labels.
save	Saves component at file path

evalml.pipelines.components.DecisionTreeClassifier.__init__

DecisionTreeClassifier.**__init__**(*criterion='gini'*, *max_features='auto'*, *max_depth=6*, *min_samples_split=2*, *min_weight_fraction_leaf=0.0*, *random_seed=0*, ***kwargs*)

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

evalml.pipelines.components.DecisionTreeClassifier.clone

DecisionTreeClassifier.**clone**()

Constructs a new component with the same parameters and random state.

Returns A new instance of this component with identical parameters and random state.

evalml.pipelines.components.DecisionTreeClassifier.describe

DecisionTreeClassifier.**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.DecisionTreeClassifier.fit

DecisionTreeClassifier.**fit**(*X*, *y=None*)

Fits component to data

Parameters

- **X** (*list, pd.DataFrame or np.ndarray*) – The input training data of shape [n_samples, n_features]

- **y** (*list, pd.Series, np.ndarray, optional*) – The target training data of length [n_samples]

Returns self

evalml.pipelines.components.DecisionTreeClassifier.load

static DecisionTreeClassifier.**load**(*file_path*)

Loads component at file path

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

Returns ComponentBase object

evalml.pipelines.components.DecisionTreeClassifier.predict

DecisionTreeClassifier.**predict**(*X*)

Make predictions using selected features.

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

Returns Predicted values

Return type pd.Series

evalml.pipelines.components.DecisionTreeClassifier.predict_proba

DecisionTreeClassifier.**predict_proba**(*X*)

Make probability estimates for labels.

Parameters **X** (*pd.DataFrame, or np.ndarray*) – Features

Returns Probability estimates

Return type pd.Series

evalml.pipelines.components.DecisionTreeClassifier.save

DecisionTreeClassifier.**save**(*file_path*, *pickle_protocol=5*)

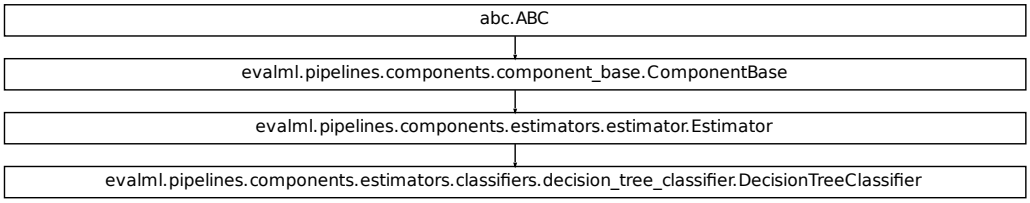
Saves component at file path

Parameters

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

Returns None

Class Inheritance



evalml.pipelines.components.KNeighborsClassifier

class evalml.pipelines.components.**KNeighborsClassifier**(*n_neighbors=5*, *weights='uniform'*, *algorithm='auto'*, *leaf_size=30*, *p=2*, *random_seed=0*, ***kwargs*)

K-Nearest Neighbors Classifier.

name = 'KNN Classifier'

model_family = 'k_neighbors'

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

hyperparameter_ranges = {'algorithm': ['auto', 'ball_tree', 'kd_tree', 'brute'], 'leaf

default_parameters = {'algorithm': 'auto', 'leaf_size': 30, 'n_neighbors': 5, 'p': 2,

predict_uses_y = False

Instance attributes

feature_importance	Returns array of 0's matching the input number of features as feature_importance is not defined for KNN classifiers.
needs_fitting	
parameters	Returns the parameters which were used to initialize the component

Methods:

__init__	Initialize self.
clone	Constructs a new component with the same parameters and random state.
describe	Describe a component and its parameters
fit	Fits component to data
load	Loads component at file path
predict	Make predictions using selected features.
predict_proba	Make probability estimates for labels.
save	Saves component at file path

evalml.pipelines.components.KNeighborsClassifier.__init__

KNeighborsClassifier.**__init__**(*n_neighbors=5*, *weights='uniform'*, *algorithm='auto'*, *leaf_size=30*, *p=2*, *random_seed=0*, ***kwargs*)

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

evalml.pipelines.components.KNeighborsClassifier.clone

KNeighborsClassifier.**clone**()

Constructs a new component with the same parameters and random state.

Returns A new instance of this component with identical parameters and random state.

evalml.pipelines.components.KNeighborsClassifier.describe

KNeighborsClassifier.**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.KNeighborsClassifier.fit

KNeighborsClassifier.**fit**(*X*, *y=None*)

Fits component to data

Parameters

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

Returns self

evalml.pipelines.components.KNeighborsClassifier.load

static KNeighborsClassifier.**load**(*file_path*)
Loads component at file path

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

Returns ComponentBase object

evalml.pipelines.components.KNeighborsClassifier.predict

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

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

Returns Predicted values

Return type pd.Series

evalml.pipelines.components.KNeighborsClassifier.predict_proba

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

Parameters **X** (*pd.DataFrame, or np.ndarray*) – Features

Returns Probability estimates

Return type pd.Series

evalml.pipelines.components.KNeighborsClassifier.save

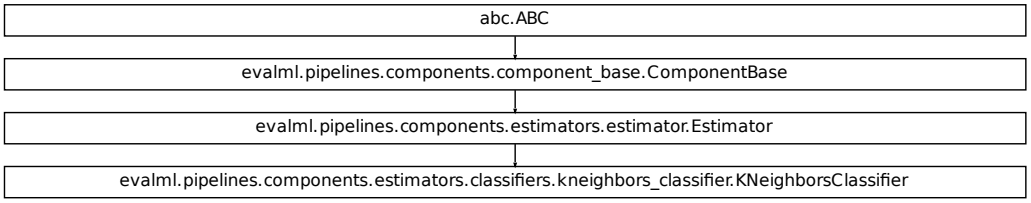
KNeighborsClassifier.**save**(*file_path*, *pickle_protocol=5*)
Saves component at file path

Parameters

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

Returns None

Class Inheritance



evalml.pipelines.components.SVMClassifier

class evalml.pipelines.components.**SVMClassifier**(*C=1.0*, *kernel='rbf'*, *gamma='scale'*, *probability=True*, *random_seed=0*, ***kwargs*)

Support Vector Machine Classifier.

name = 'SVM Classifier'

model_family = 'svm'

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

hyperparameter_ranges = {'C': Real(low=0, high=10, prior='uniform', transform='identity

default_parameters = {'C': 1.0, 'gamma': 'scale', 'kernel': 'rbf', 'probability': True

predict_uses_y = False

Instance attributes

feature_importance	Feature importance only works with linear kernels.
needs_fitting	
parameters	Returns the parameters which were used to initialize the component

Methods:

__init__	Initialize self.
clone	Constructs a new component with the same parameters and random state.
describe	Describe a component and its parameters
fit	Fits component to data
load	Loads component at file path
predict	Make predictions using selected features.
predict_proba	Make probability estimates for labels.
save	Saves component at file path

evalml.pipelines.components.SVMClassifier.__init__

SVMClassifier.**__init__**(*C=1.0*, *kernel='rbf'*, *gamma='scale'*, *probability=True*, *random_seed=0*, ***kwargs*)

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

evalml.pipelines.components.SVMClassifier.clone

SVMClassifier.**clone**()

Constructs a new component with the same parameters and random state.

Returns A new instance of this component with identical parameters and random state.

evalml.pipelines.components.SVMClassifier.describe

SVMClassifier.**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.SVMClassifier.fit

SVMClassifier.**fit**(*X*, *y=None*)

Fits component to data

Parameters

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

Returns self

evalml.pipelines.components.SVMClassifier.load

static SVMClassifier.**load**(*file_path*)
Loads component at file path

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

Returns ComponentBase object

evalml.pipelines.components.SVMClassifier.predict

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

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

Returns Predicted values

Return type pd.Series

evalml.pipelines.components.SVMClassifier.predict_proba

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

Parameters **X** (*pd.DataFrame, or np.ndarray*) – Features

Returns Probability estimates

Return type pd.Series

evalml.pipelines.components.SVMClassifier.save

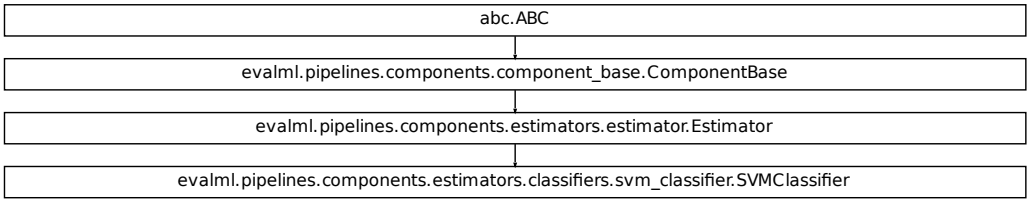
SVMClassifier.**save**(*file_path*, *pickle_protocol=5*)
Saves component at file path

Parameters

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

Returns None

Class Inheritance



Regressors

Regressors are components that output a predicted target value.

ARIMARegressor	Autoregressive Integrated Moving Average Model.
CatBoostRegressor	CatBoost Regressor, a regressor that uses gradient-boosting on decision trees.
ElasticNetRegressor	Elastic Net Regressor.
LinearRegressor	Linear Regressor.
ExtraTreesRegressor	Extra Trees Regressor.
RandomForestRegressor	Random Forest Regressor.
XGBoostRegressor	XGBoost Regressor.
BaselineRegressor	Regressor that predicts using the specified strategy.
TimeSeriesBaselineEstimator	Time series estimator that predicts using the naive forecasting approach.
StackedEnsembleRegressor	Stacked Ensemble Regressor.
DecisionTreeRegressor	Decision Tree Regressor.
LightGBMRegressor	LightGBM Regressor
SVMRegressor	Support Vector Machine Regressor.

evalml.pipelines.components.ARIMARegressor

class evalml.pipelines.components.**ARIMARegressor**(*date_index=None*, *trend=None*, *start_p=2*, *d=0*, *start_q=2*, *max_p=5*, *max_d=2*, *max_q=5*, *seasonal=True*, *n_jobs=- 1*, *random_seed=0*, ***kwargs*)

Autoregressive Integrated Moving Average Model. The three parameters (p, d, q) are the AR order, the degree of differencing, and the MA order. More information here: https://www.statsmodels.org/devel/generated/statsmodels.tsa.arima_model.ARIMA.html

Currently ARIMARegressor isn't supported via conda install. It's recommended that it be installed via PyPI.

name = 'ARIMA Regressor'

model_family = 'arima'

```
supported_problem_types = [<ProblemTypes.TIME_SERIES_REGRESSION: 'time series regressi
hyperparameter_ranges = {'d': Integer(low=0, high=2, prior='uniform', transform='ident
default_parameters = {'d': 0, 'date_index': None, 'max_d': 2, 'max_p': 5, 'max_q': 5,
predict_uses_y = False
```

Instance attributes

feature_importance	Returns array of 0's with a length of 1 as feature_importance is not defined for ARIMA regressor.
needs_fitting	
parameters	Returns the parameters which were used to initialize the component

Methods:

__init__	**param date_index** Specifies the name of the column in X that provides the datetime objects. Defaults to None.
clone	Constructs a new component with the same parameters and random state.
describe	Describe a component and its parameters
fit	Fits component to data
load	Loads component at file path
predict	Make predictions using selected features.
predict_proba	Make probability estimates for labels.
save	Saves component at file path

evalml.pipelines.components.ARIMARegressor.__init__

ARIMARegressor.**__init__**(*date_index=None*, *trend=None*, *start_p=2*, *d=0*, *start_q=2*, *max_p=5*, *max_d=2*, *max_q=5*, *seasonal=True*, *n_jobs=-1*, *random_seed=0*, ***kwargs*)

Parameters

- **date_index** (*str*) – Specifies the name of the column in X that provides the datetime objects. Defaults to None.
- **trend** (*str*) – Controls the deterministic trend. Options are [‘n’, ‘c’, ‘t’, ‘ct’] where ‘c’ is a constant term, ‘t’ indicates a linear trend, and ‘ct’ is both. Can also be an iterable when defining a polynomial, such as [1, 1, 0, 1].
- **start_p** (*int*) – Minimum Autoregressive order.
- **d** (*int*) – Minimum Differencing degree.
- **start_q** (*int*) – Minimum Moving Average order.

- **max_p** (*int*) – Maximum Autoregressive order.
- **max_d** (*int*) – Maximum Differencing degree.
- **max_q** (*int*) – Maximum Moving Average order.
- **seasonal** (*bool*) – Whether to fit a seasonal model to ARIMA.

evalml.pipelines.components.ARIMARegressor.clone

ARIMARegressor.**clone**()

Constructs a new component with the same parameters and random state.

Returns A new instance of this component with identical parameters and random state.

evalml.pipelines.components.ARIMARegressor.describe

ARIMARegressor.**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.ARIMARegressor.fit

ARIMARegressor.**fit**(*X*, *y=None*)

Fits component to data

Parameters

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

Returns self

evalml.pipelines.components.ARIMARegressor.load

static ARIMARegressor.**load**(*file_path*)

Loads component at file path

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

Returns ComponentBase object

evalml.pipelines.components.ARIMARegressor.predict

ARIMARegressor.**predict**(*X*, *y=None*)

Make predictions using selected features.

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

Returns Predicted values

Return type pd.Series

evalml.pipelines.components.ARIMARegressor.predict_proba

ARIMARegressor.**predict_proba**(*X*)

Make probability estimates for labels.

Parameters **X** (*pd.DataFrame, or np.ndarray*) – Features

Returns Probability estimates

Return type pd.Series

evalml.pipelines.components.ARIMARegressor.save

ARIMARegressor.**save**(*file_path*, *pickle_protocol=5*)

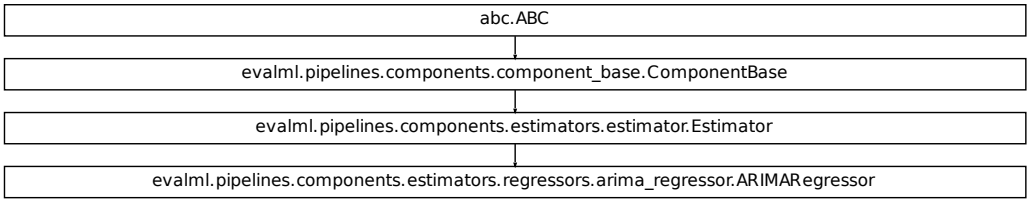
Saves component at file path

Parameters

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

Returns None

Class Inheritance



evalml.pipelines.components.CatBoostRegressor

class evalml.pipelines.components.**CatBoostRegressor**(*n_estimators=10*, *eta=0.03*, *max_depth=6*, *bootstrap_type=None*, *silent=False*, *allow_writing_files=False*, *random_seed=0*, *n_jobs=-1*, ***kwargs*)

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

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

name = 'CatBoost Regressor'

model_family = 'catboost'

supported_problem_types = [<ProblemTypes.REGRESSION: 'regression'>, <ProblemTypes.TIME

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

default_parameters = {'allow_writing_files': False, 'bootstrap_type': None, 'eta': 0.0

predict_uses_y = False

Instance attributes

feature_importance	Return an attribute of instance, which is of type owner.
needs_fitting	
parameters	Returns the parameters which were used to initialize the component

Methods:

__init__	CatBoost Regressor.
clone	Constructs a new component with the same parameters and random state.
describe	Describe a component and its parameters
fit	Fits component to data
load	Loads component at file path
predict	Make predictions using selected features.
predict_proba	Make probability estimates for labels.
save	Saves component at file path

evalml.pipelines.components.CatBoostRegressor.__init__

CatBoostRegressor.**__init__**(*n_estimators=10*, *eta=0.03*, *max_depth=6*, *bootstrap_type=None*, *silent=False*, *allow_writing_files=False*, *random_seed=0*, *n_jobs=- 1*, ***kwargs*)

CatBoost Regressor.

Parameters

- **n_estimators** (*int*) – Number of gradient boosted trees. Equivalent to number of boosting rounds. Defaults to 100.
- **eta** (*float*) – Learning rate. Defaults to 0.1.
- **max_depth** (*int*) – Maximum tree depth for base learners. Defaults to 6.
- **bootstrap_type** (*string*) – Defines the method for sampling the weights of objects. Defaults to None.
- **silent** (*bool*) – Whether to emit logging while training. Default to False.
- **allow_writing_files** (*bool*) – Whether to allow writing of analytical and snapshot files during training. Defaults to False.
- **random_seed** (*int*) – Seed for the random number generator. Defaults to 0.
- **n_jobs** (*int*) – Number of parallel threads used to run CatBoost. This will be passed to CatBoost as the *thread_count* parameter. Defaults to -1.

evalml.pipelines.components.CatBoostRegressor.clone

CatBoostRegressor.**clone**()

Constructs a new component with the same parameters and random state.

Returns A new instance of this component with identical parameters and random state.

evalml.pipelines.components.CatBoostRegressor.describe

CatBoostRegressor.**describe**(*print_name=False*, *return_dict=False*)

Describe a component and its parameters

Parameters

- **print_name** (*bool, optional*) – whether to print name of component
- **return_dict** (*bool, optional*) – whether to return description as dictionary in the format {"name": name, "parameters": parameters}

Returns prints and returns dictionary

Return type None or dict

evalml.pipelines.components.CatBoostRegressor.fit

CatBoostRegressor.**fit**(*X*, *y=None*)

Fits component to data

Parameters

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

Returns self

evalml.pipelines.components.CatBoostRegressor.load

static CatBoostRegressor.**load**(*file_path*)

Loads component at file path

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

Returns ComponentBase object

evalml.pipelines.components.CatBoostRegressor.predict

CatBoostRegressor.**predict**(*X*)

Make predictions using selected features.

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

Returns Predicted values

Return type pd.Series

evalml.pipelines.components.CatBoostRegressor.predict_proba

CatBoostRegressor.**predict_proba**(*X*)

Make probability estimates for labels.

Parameters **X** (*pd.DataFrame, or np.ndarray*) – Features

Returns Probability estimates

Return type pd.Series

evalml.pipelines.components.CatBoostRegressor.save

CatBoostRegressor.**save**(*file_path*, *pickle_protocol=5*)

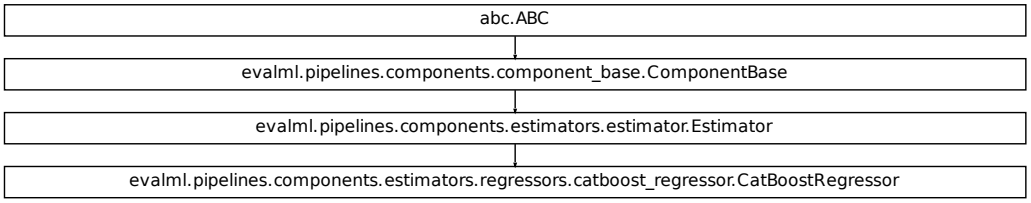
Saves component at file path

Parameters

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

Returns None

Class Inheritance



evalml.pipelines.components.ElasticNetRegressor

class evalml.pipelines.components.**ElasticNetRegressor**(*alpha=0.0001*, *l1_ratio=0.15*, *max_iter=1000*, *normalize=False*, *random_seed=0*, ***kwargs*)

Elastic Net Regressor.

name = 'Elastic Net Regressor'

model_family = 'linear_model'

supported_problem_types = [<ProblemTypes.REGRESSION: 'regression'>, <ProblemTypes.TIME

hyperparameter_ranges = {'alpha': Real(low=0, high=1, prior='uniform', transform='iden

default_parameters = {'alpha': 0.0001, 'l1_ratio': 0.15, 'max_iter': 1000, 'normalize'

predict_uses_y = False

Instance attributes

feature_importance	Return an attribute of instance, which is of type owner.
needs_fitting	
parameters	Returns the parameters which were used to initialize the component

Methods:

__init__	Initialize self.
clone	Constructs a new component with the same parameters and random state.
describe	Describe a component and its parameters
fit	Fits component to data
load	Loads component at file path
predict	Make predictions using selected features.
predict_proba	Make probability estimates for labels.
save	Saves component at file path

evalml.pipelines.components.ElasticNetRegressor.__init__

ElasticNetRegressor.**__init__**(*alpha=0.0001*, *l1_ratio=0.15*, *max_iter=1000*, *normalize=False*, *random_seed=0*, ***kwargs*)

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

evalml.pipelines.components.ElasticNetRegressor.clone

ElasticNetRegressor.**clone**()

Constructs a new component with the same parameters and random state.

Returns A new instance of this component with identical parameters and random state.

evalml.pipelines.components.ElasticNetRegressor.describe

ElasticNetRegressor.**describe**(*print_name=False*, *return_dict=False*)

Describe a component and its parameters

Parameters

- **print_name** (*bool, optional*) – whether to print name of component
- **return_dict** (*bool, optional*) – whether to return description as dictionary in the format {"name": name, "parameters": parameters}

Returns prints and returns dictionary

Return type None or dict

evalml.pipelines.components.ElasticNetRegressor.fit

ElasticNetRegressor.**fit**(*X*, *y=None*)

Fits component to data

Parameters

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

Returns self

evalml.pipelines.components.ElasticNetRegressor.load

static ElasticNetRegressor.**load**(*file_path*)
Loads component at file path

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

Returns ComponentBase object

evalml.pipelines.components.ElasticNetRegressor.predict

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

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

Returns Predicted values

Return type pd.Series

evalml.pipelines.components.ElasticNetRegressor.predict_proba

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

Parameters **X** (*pd.DataFrame, or np.ndarray*) – Features

Returns Probability estimates

Return type pd.Series

evalml.pipelines.components.ElasticNetRegressor.save

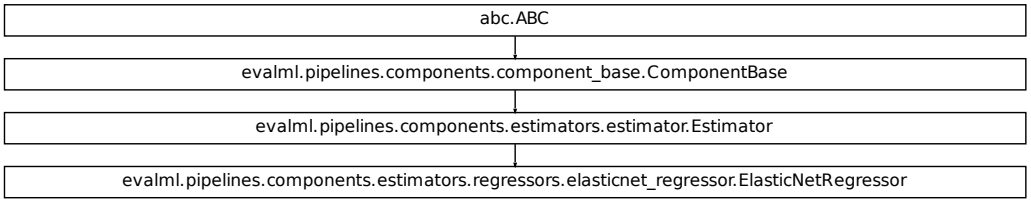
ElasticNetRegressor.**save**(*file_path*, *pickle_protocol=5*)
Saves component at file path

Parameters

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

Returns None

Class Inheritance



evalml.pipelines.components.LinearRegressor

class evalml.pipelines.components.**LinearRegressor**(*fit_intercept=True*, *normalize=False*, *n_jobs=- 1*, *random_seed=0*, ***kwargs*)

Linear Regressor.

name = 'Linear Regressor'

model_family = 'linear_model'

supported_problem_types = [<ProblemTypes.REGRESSION: 'regression'>, <ProblemTypes.TIME

hyperparameter_ranges = {'fit_intercept': [True, False], 'normalize': [True, False]}

default_parameters = {'fit_intercept': True, 'n_jobs': -1, 'normalize': False}

predict_uses_y = False

Instance attributes

feature_importance	Return an attribute of instance, which is of type owner.
needs_fitting	
parameters	Returns the parameters which were used to initialize the component

Methods:

__init__	Initialize self.
clone	Constructs a new component with the same parameters and random state.
describe	Describe a component and its parameters
fit	Fits component to data
load	Loads component at file path
predict	Make predictions using selected features.
predict_proba	Make probability estimates for labels.
save	Saves component at file path

evalml.pipelines.components.LinearRegressor.__init__

LinearRegressor.**__init__**(*fit_intercept=True*, *normalize=False*, *n_jobs=- 1*, *random_seed=0*, ***kwargs*)

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

evalml.pipelines.components.LinearRegressor.clone

LinearRegressor.**clone**()

Constructs a new component with the same parameters and random state.

Returns A new instance of this component with identical parameters and random state.

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

Returns self

evalml.pipelines.components.LinearRegressor.load

static LinearRegressor.**load**(*file_path*)
Loads component at file path

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

Returns ComponentBase object

evalml.pipelines.components.LinearRegressor.predict

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

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

Returns Predicted values

Return type pd.Series

evalml.pipelines.components.LinearRegressor.predict_proba

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

Parameters **X** (*pd.DataFrame, or np.ndarray*) – Features

Returns Probability estimates

Return type pd.Series

evalml.pipelines.components.LinearRegressor.save

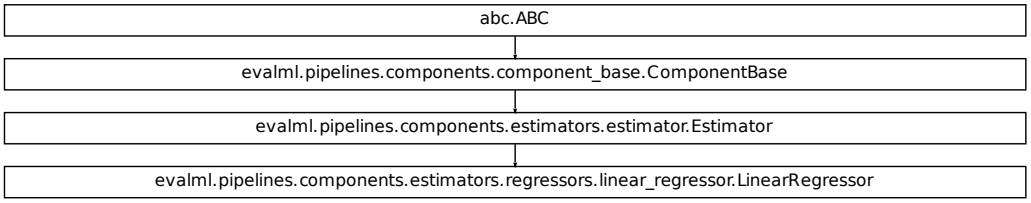
LinearRegressor.**save**(*file_path*, *pickle_protocol=5*)
Saves component at file path

Parameters

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

Returns None

Class Inheritance



evalml.pipelines.components.ExtraTreesRegressor

class evalml.pipelines.components.**ExtraTreesRegressor**(*n_estimators=100*, *max_features='auto'*, *max_depth=6*, *min_samples_split=2*, *min_weight_fraction_leaf=0.0*, *n_jobs=- 1*, *random_seed=0*, ***kwargs*)

Extra Trees Regressor.

name = 'Extra Trees Regressor'

model_family = 'extra_trees'

supported_problem_types = [<ProblemTypes.REGRESSION: 'regression'>, <ProblemTypes.TIME

hyperparameter_ranges = {'max_depth': Integer(low=4, high=10, prior='uniform', transfo

default_parameters = {'max_depth': 6, 'max_features': 'auto', 'min_samples_split': 2,

predict_uses_y = False

Instance attributes

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

Methods:

__init__	Initialize self.
clone	Constructs a new component with the same parameters and random state.
describe	Describe a component and its parameters
fit	Fits component to data
load	Loads component at file path
predict	Make predictions using selected features.
predict_proba	Make probability estimates for labels.
save	Saves component at file path

evalml.pipelines.components.ExtraTreesRegressor.__init__

ExtraTreesRegressor.**__init__**(*n_estimators=100*, *max_features='auto'*, *max_depth=6*, *min_samples_split=2*, *min_weight_fraction_leaf=0.0*, *n_jobs=- 1*, *random_seed=0*, ***kwargs*)

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

evalml.pipelines.components.ExtraTreesRegressor.clone

ExtraTreesRegressor.**clone**()

Constructs a new component with the same parameters and random state.

Returns A new instance of this component with identical parameters and random state.

evalml.pipelines.components.ExtraTreesRegressor.describe

ExtraTreesRegressor.**describe**(*print_name=False*, *return_dict=False*)

Describe a component and its parameters

Parameters

- **print_name** (*bool, optional*) – whether to print name of component
- **return_dict** (*bool, optional*) – whether to return description as dictionary in the format {"name": name, "parameters": parameters}

Returns prints and returns dictionary

Return type None or dict

evalml.pipelines.components.ExtraTreesRegressor.fit

ExtraTreesRegressor.**fit**(*X*, *y=None*)

Fits component to data

Parameters

- **X** (*list, pd.DataFrame or np.ndarray*) – The input training data of shape [n_samples, n_features]

- **y** (*list, pd.Series, np.ndarray, optional*) – The target training data of length [n_samples]

Returns self

evalml.pipelines.components.ExtraTreesRegressor.load

static ExtraTreesRegressor.**load**(*file_path*)

Loads component at file path

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

Returns ComponentBase object

evalml.pipelines.components.ExtraTreesRegressor.predict

ExtraTreesRegressor.**predict**(*X*)

Make predictions using selected features.

Parameters **X** (*pd.DataFrame, np.ndarray*) – Data of shape [n_samples, n_features]

Returns Predicted values

Return type pd.Series

evalml.pipelines.components.ExtraTreesRegressor.predict_proba

ExtraTreesRegressor.**predict_proba**(*X*)

Make probability estimates for labels.

Parameters **X** (*pd.DataFrame, or np.ndarray*) – Features

Returns Probability estimates

Return type pd.Series

evalml.pipelines.components.ExtraTreesRegressor.save

ExtraTreesRegressor.**save**(*file_path*, *pickle_protocol=5*)

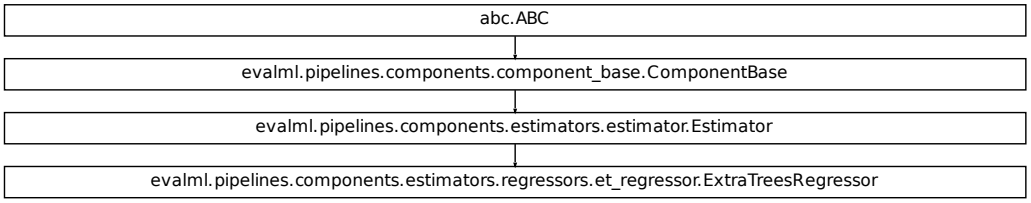
Saves component at file path

Parameters

- **file_path** (*str*) – Location to save file
- **pickle_protocol** (*int*) – The pickle data stream format.

Returns None

Class Inheritance



evalml.pipelines.components.RandomForestRegressor

class evalml.pipelines.components.**RandomForestRegressor**(*n_estimators=100*, *max_depth=6*, *n_jobs=-1*, *random_seed=0*, ***kwargs*)

Random Forest Regressor.

name = 'Random Forest Regressor'

model_family = 'random_forest'

supported_problem_types = [<ProblemTypes.REGRESSION: 'regression'>, <ProblemTypes.TIME

hyperparameter_ranges = {'max_depth': Integer(low=1, high=32, prior='uniform', transfo

default_parameters = {'max_depth': 6, 'n_estimators': 100, 'n_jobs': -1}

predict_uses_y = False

Instance attributes

feature_importance	Returns importance associated with each feature.
needs_fitting	
parameters	Returns the parameters which were used to initialize the component

Methods:

__init__	Initialize self.
clone	Constructs a new component with the same parameters and random state.
describe	Describe a component and its parameters
fit	Fits component to data
load	Loads component at file path
predict	Make predictions using selected features.
predict_proba	Make probability estimates for labels.
save	Saves component at file path

evalml.pipelines.components.RandomForestRegressor.__init__

RandomForestRegressor.**__init__**(*n_estimators=100*, *max_depth=6*, *n_jobs=-1*, *random_seed=0*, ***kwargs*)

Initialize self. See help(type(self)) for accurate signature.

evalml.pipelines.components.RandomForestRegressor.clone

RandomForestRegressor.**clone**()

Constructs a new component with the same parameters and random state.

Returns A new instance of this component with identical parameters and random state.

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** (*list, pd.DataFrame or np.ndarray*) – The input training data of shape [n_samples, n_features]
- **y** (*list, pd.Series, np.ndarray, optional*) – The target training data of length [n_samples]

Returns self

evalml.pipelines.components.RandomForestRegressor.load

static RandomForestRegressor.**load**(*file_path*)
Loads component at file path

Parameters **file_path** (*str*) – Location to load file

Returns ComponentBase object

evalml.pipelines.components.RandomForestRegressor.predict

RandomForestRegressor.**predict**(*X*)
Make predictions using selected features.

Parameters **X** (*pd.DataFrame, np.ndarray*) – Data of shape [n_samples, n_features]

Returns Predicted values

Return type pd.Series

evalml.pipelines.components.RandomForestRegressor.predict_proba

RandomForestRegressor.**predict_proba**(*X*)
Make probability estimates for labels.

Parameters **X** (*pd.DataFrame, or np.ndarray*) – Features

Returns Probability estimates

Return type pd.Series

evalml.pipelines.components.RandomForestRegressor.save

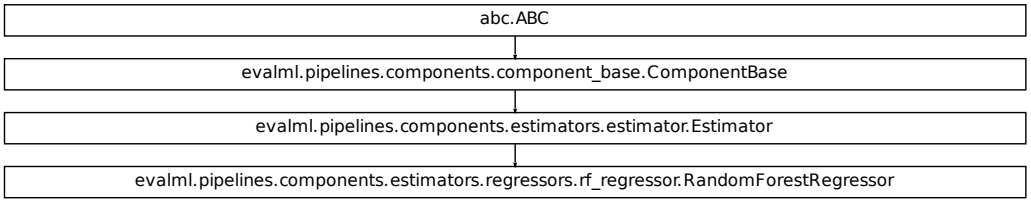
RandomForestRegressor.**save**(*file_path*, *pickle_protocol=5*)
Saves component at file path

Parameters

- **file_path** (*str*) – Location to save file
- **pickle_protocol** (*int*) – The pickle data stream format.

Returns None

Class Inheritance



evalml.pipelines.components.XGBoostRegressor

class evalml.pipelines.components.**XGBoostRegressor**(*eta=0.1*, *max_depth=6*, *min_child_weight=1*, *n_estimators=100*, *random_seed=0*, *n_jobs=-1*, ***kwargs*)

XGBoost Regressor.

name = 'XGBoost Regressor'

model_family = 'xgboost'

supported_problem_types = [<ProblemTypes.REGRESSION: 'regression'>, <ProblemTypes.TIME

hyperparameter_ranges = {'eta': Real(low=1e-06, high=1, prior='uniform', transform='id

default_parameters = {'eta': 0.1, 'max_depth': 6, 'min_child_weight': 1, 'n_estimators

predict_uses_y = False

Instance attributes

SEED_MAX	
SEED_MIN	
feature_importance	Return an attribute of instance, which is of type owner.
needs_fitting	
parameters	Returns the parameters which were used to initialize the component

Methods:

__init__	XGBoost Regressor.
clone	Constructs a new component with the same parameters and random state.
describe	Describe a component and its parameters
fit	Fits component to data
load	Loads component at file path
predict	Make predictions using selected features.
predict_proba	Make probability estimates for labels.
save	Saves component at file path

evalml.pipelines.components.XGBoostRegressor.__init__

XGBoostRegressor.**__init__**(*eta=0.1*, *max_depth=6*, *min_child_weight=1*, *n_estimators=100*, *random_seed=0*, *n_jobs=- 1*, ***kwargs*)

XGBoost Regressor.

Parameters

- **eta** (*float*) – Learning rate. Defaults to 0.1.
- **max_depth** (*int*) – Maximum tree depth for base learners. Defaults to 6.
- **min_child_weight** (*float*) – Minimum sum of instance weight(hessian) needed in a child. Defaults to 1.
- **n_estimators** (*int*) – Number of gradient boosted trees. Equivalent to number of boosting rounds. Defaults to 100.
- **random_seed** (*int*) – Seed for the random number generator. Defaults to 0.
- **n_jobs** (*int*) – Number of parallel threads used to run xgboost. Note that creating thread contention will significantly slow down the algorithm. Defaults to -1.

evalml.pipelines.components.XGBoostRegressor.clone

XGBoostRegressor.**clone**()

Constructs a new component with the same parameters and random state.

Returns A new instance of this component with identical parameters and random state.

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** (*list, pd.DataFrame or np.ndarray*) – The input training data of shape [n_samples, n_features]
- **y** (*list, pd.Series, np.ndarray, optional*) – The target training data of length [n_samples]

Returns self

evalml.pipelines.components.XGBoostRegressor.load

static XGBoostRegressor.**load**(*file_path*)

Loads component at file path

Parameters **file_path** (*str*) – Location to load file

Returns ComponentBase object

evalml.pipelines.components.XGBoostRegressor.predict

XGBoostRegressor.**predict**(*X*)

Make predictions using selected features.

Parameters **X** (*pd.DataFrame, np.ndarray*) – Data of shape [n_samples, n_features]

Returns Predicted values

Return type pd.Series

evalml.pipelines.components.XGBoostRegressor.predict_proba

XGBoostRegressor.**predict_proba**(*X*)

Make probability estimates for labels.

Parameters **X** (*pd.DataFrame, or np.ndarray*) – Features

Returns Probability estimates

Return type pd.Series

evalml.pipelines.components.XGBoostRegressor.save

XGBoostRegressor.**save**(*file_path*, *pickle_protocol=5*)

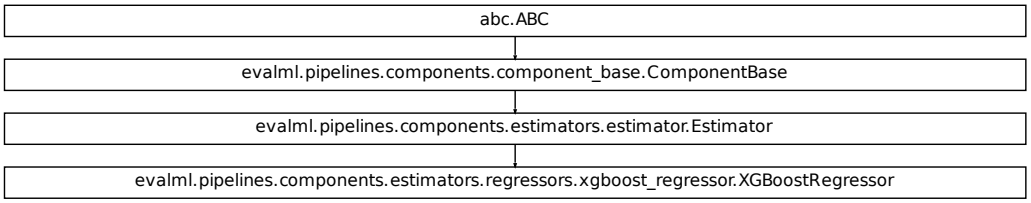
Saves component at file path

Parameters

- **file_path** (*str*) – Location to save file
- **pickle_protocol** (*int*) – The pickle data stream format.

Returns None

Class Inheritance



evalml.pipelines.components.BaselineRegressor

class evalml.pipelines.components.**BaselineRegressor**(*strategy='mean'*, *random_seed=0*, ***kwargs*)

Regressor that predicts using the specified strategy.

This is useful as a simple baseline regressor to compare with other regressors.

name = 'Baseline Regressor'

model_family = 'baseline'

supported_problem_types = [<ProblemTypes.REGRESSION: 'regression'>, <ProblemTypes.TIME

hyperparameter_ranges = {}

default_parameters = {'strategy': 'mean'}

predict_uses_y = False

Instance attributes

feature_importance	Returns importance associated with each feature.
needs_fitting	
parameters	Returns the parameters which were used to initialize the component

Methods:

__init__	Baseline regressor that uses a simple strategy to make predictions.
clone	Constructs a new component with the same parameters and random state.
describe	Describe a component and its parameters
fit	Fits component to data
load	Loads component at file path
predict	Make predictions using selected features.
predict_proba	Make probability estimates for labels.
save	Saves component at file path

evalml.pipelines.components.BaselineRegressor.__init__

BaselineRegressor.**__init__**(*strategy='mean'*, *random_seed=0*, ***kwargs*)

Baseline regressor that uses a simple strategy to make predictions.

Parameters

- **strategy** (*str*) – Method used to predict. Valid options are "mean", "median". Defaults to "mean".
- **random_seed** (*int*) – Seed for the random number generator. Defaults to 0.

evalml.pipelines.components.BaselineRegressor.clone

BaselineRegressor.**clone**()

Constructs a new component with the same parameters and random state.

Returns A new instance of this component with identical parameters and random state.

evalml.pipelines.components.BaselineRegressor.describe

BaselineRegressor.**describe**(*print_name=False*, *return_dict=False*)

Describe a component and its parameters

Parameters

- **print_name** (*bool, optional*) – whether to print name of component
- **return_dict** (*bool, optional*) – whether to return description as dictionary in the format {"name": name, "parameters": parameters}

Returns prints and returns dictionary

Return type None or dict

evalml.pipelines.components.BaselineRegressor.fit

BaselineRegressor.**fit**(*X*, *y=None*)

Fits component to data

Parameters

- **X** (*list, pd.DataFrame or np.ndarray*) – The input training data of shape [n_samples, n_features]
- **y** (*list, pd.Series, np.ndarray, optional*) – The target training data of length [n_samples]

Returns self

evalml.pipelines.components.BaselineRegressor.load

static BaselineRegressor.**load**(*file_path*)

Loads component at file path

Parameters **file_path** (*str*) – Location to load file

Returns ComponentBase object

evalml.pipelines.components.BaselineRegressor.predict

BaselineRegressor.**predict**(*X*)

Make predictions using selected features.

Parameters **X** (*pd.DataFrame, np.ndarray*) – Data of shape [n_samples, n_features]

Returns Predicted values

Return type pd.Series

evalml.pipelines.components.BaselineRegressor.predict_proba

BaselineRegressor.**predict_proba**(*X*)

Make probability estimates for labels.

Parameters **X** (*pd.DataFrame, or np.ndarray*) – Features

Returns Probability estimates

Return type pd.Series

evalml.pipelines.components.BaselineRegressor.save

BaselineRegressor.**save**(*file_path*, *pickle_protocol=5*)

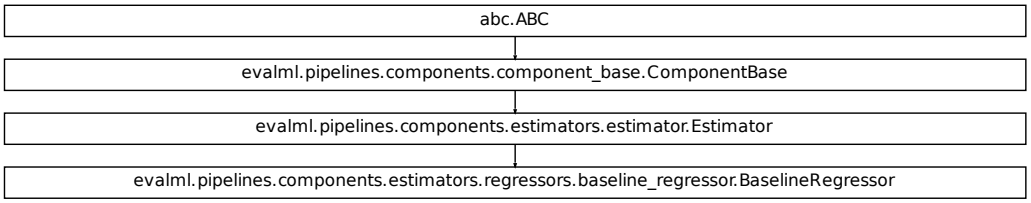
Saves component at file path

Parameters

- **file_path** (*str*) – Location to save file
- **pickle_protocol** (*int*) – The pickle data stream format.

Returns None

Class Inheritance



evalml.pipelines.components.TimeSeriesBaselineEstimator

class evalml.pipelines.components.**TimeSeriesBaselineEstimator**(*gap=1*, *random_seed=0*, ***kwargs*)

Time series estimator that predicts using the naive forecasting approach.

This is useful as a simple baseline estimator for time series problems

name = 'Time Series Baseline Estimator'

model_family = 'baseline'

supported_problem_types = [<ProblemTypes.TIME_SERIES_REGRESSION: 'time series regressi

hyperparameter_ranges = {}

default_parameters = {'gap': 1}

predict_uses_y = True

Instance attributes

feature_importance	Returns importance associated with each feature.
needs_fitting	
parameters	Returns the parameters which were used to initialize the component

Methods:

__init__	Baseline time series estimator that predicts using the naive forecasting approach.
clone	Constructs a new component with the same parameters and random state.
describe	Describe a component and its parameters
fit	Fits component to data
load	Loads component at file path
predict	Make predictions using selected features.
predict_proba	Make probability estimates for labels.
save	Saves component at file path

evalml.pipelines.components.TimeSeriesBaselineEstimator.__init__

TimeSeriesBaselineEstimator.**__init__**(*gap=1*, *random_seed=0*, ***kwargs*)

Baseline time series estimator that predicts using the naive forecasting approach.

Parameters

- **gap** (*int*) – Gap between prediction date and target date and must be a positive integer. If gap is 0, target date will be shifted ahead by 1 time period.
- **random_state** (*None, int*) – Deprecated - use random_seed instead.
- **random_seed** (*int*) – Seed for the random number generator. Defaults to 0.

evalml.pipelines.components.TimeSeriesBaselineEstimator.clone

TimeSeriesBaselineEstimator.**clone**()

Constructs a new component with the same parameters and random state.

Returns A new instance of this component with identical parameters and random state.

evalml.pipelines.components.TimeSeriesBaselineEstimator.describe

TimeSeriesBaselineEstimator.**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.TimeSeriesBaselineEstimator.fit

TimeSeriesBaselineEstimator.**fit**(*X*, *y=None*)

Fits component to data

Parameters

- **X** (*list, pd.DataFrame or np.ndarray*) – The input training data of shape [n_samples, n_features]
- **y** (*list, pd.Series, np.ndarray, optional*) – The target training data of length [n_samples]

Returns self

evalml.pipelines.components.TimeSeriesBaselineEstimator.load

static TimeSeriesBaselineEstimator.**load**(*file_path*)

Loads component at file path

Parameters **file_path** (*str*) – Location to load file

Returns ComponentBase object

evalml.pipelines.components.TimeSeriesBaselineEstimator.predict

TimeSeriesBaselineEstimator.**predict**(*X*, *y=None*)

Make predictions using selected features.

Parameters **X** (*pd.DataFrame, np.ndarray*) – Data of shape [n_samples, n_features]

Returns Predicted values

Return type pd.Series

evalml.pipelines.components.TimeSeriesBaselineEstimator.predict_proba

TimeSeriesBaselineEstimator.**predict_proba**(*X*, *y=None*)

Make probability estimates for labels.

Parameters **X** (*pd.DataFrame, or np.ndarray*) – Features

Returns Probability estimates

Return type pd.Series

evalml.pipelines.components.TimeSeriesBaselineEstimator.save

TimeSeriesBaselineEstimator.**save**(*file_path*, *pickle_protocol=5*)

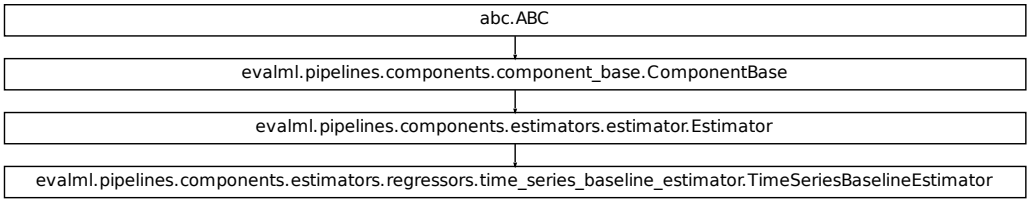
Saves component at file path

Parameters

- **file_path** (*str*) – Location to save file
- **pickle_protocol** (*int*) – The pickle data stream format.

Returns None

Class Inheritance



evalml.pipelines.components.StackedEnsembleRegressor

class evalml.pipelines.components.**StackedEnsembleRegressor**(*input_pipelines=None*, *final_estimator=None*, *cv=None*, *n_jobs=-1*, *random_seed=0*, ***kwargs*)

Stacked Ensemble Regressor.

name = 'Stacked Ensemble Regressor'

model_family = 'ensemble'

supported_problem_types = [<ProblemTypes.REGRESSION: 'regression'>, <ProblemTypes.TIME

hyperparameter_ranges = {}

default_parameters = {'cv': None, 'final_estimator': None, 'n_jobs': -1}

predict_uses_y = False

Instance attributes

feature_importance	Not implemented for StackedEnsembleClassifier and StackedEnsembleRegressor
needs_fitting	
parameters	Returns the parameters which were used to initialize the component

Methods:

__init__	Stacked ensemble regressor.
clone	Constructs a new component with the same parameters and random state.
describe	Describe a component and its parameters
fit	Fits component to data
load	Loads component at file path
predict	Make predictions using selected features.
predict_proba	Make probability estimates for labels.
save	Saves component at file path

evalml.pipelines.components.StackedEnsembleRegressor.__init__

StackedEnsembleRegressor.**__init__**(*input_pipelines=None*, *final_estimator=None*, *cv=None*, *n_jobs=- 1*, *random_seed=0*, ***kwargs*)

Stacked ensemble regressor.

Parameters

- **input_pipelines** (*list(PipelineBase or subclass obj)*) – List of pipeline instances to use as the base estimators. This must not be None or an empty list or else EnsembleMissingPipelinesError will be raised.
- **final_estimator** (*Estimator or subclass*) – The regressor used to combine the base estimators. If None, uses LinearRegressor.
- **cv** (*int, cross-validation generator or an iterable*) – Determines the cross-validation splitting strategy used to train final_estimator. For int/None inputs, KFold is used. Defaults to None. Possible inputs for cv are:
  - None: 3-fold cross validation
  - int: the number of folds in a (Stratified) KFold
  - An scikit-learn cross-validation generator object
  - An iterable yielding (train, test) splits
- **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. Defaults to None. - Note: there could be some multi-process errors thrown for values of *n_jobs != 1*. If this is the case, please use *n_jobs = 1*.
- **random_seed** (*int*) – Seed for the random number generator. Defaults to 0.

evalml.pipelines.components.StackedEnsembleRegressor.clone

StackedEnsembleRegressor.**clone**()

Constructs a new component with the same parameters and random state.

Returns A new instance of this component with identical parameters and random state.

evalml.pipelines.components.StackedEnsembleRegressor.describe

StackedEnsembleRegressor.**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.StackedEnsembleRegressor.fit

StackedEnsembleRegressor.**fit**(*X*, *y=None*)

Fits component to data

Parameters

- **X** (*list, pd.DataFrame or np.ndarray*) – The input training data of shape [n_samples, n_features]
- **y** (*list, pd.Series, np.ndarray, optional*) – The target training data of length [n_samples]

Returns self

evalml.pipelines.components.StackedEnsembleRegressor.load

static StackedEnsembleRegressor.**load**(*file_path*)

Loads component at file path

Parameters **file_path** (*str*) – Location to load file

Returns ComponentBase object

evalml.pipelines.components.StackedEnsembleRegressor.predict

StackedEnsembleRegressor.**predict**(*X*)

Make predictions using selected features.

Parameters **X** (*pd.DataFrame, np.ndarray*) – Data of shape [n_samples, n_features]

Returns Predicted values

Return type pd.Series

evalml.pipelines.components.StackedEnsembleRegressor.predict_proba

StackedEnsembleRegressor.**predict_proba**(*X*)

Make probability estimates for labels.

Parameters **X** (*pd.DataFrame, or np.ndarray*) – Features

Returns Probability estimates

Return type pd.Series

evalml.pipelines.components.StackedEnsembleRegressor.save

StackedEnsembleRegressor.**save**(*file_path*, *pickle_protocol=5*)

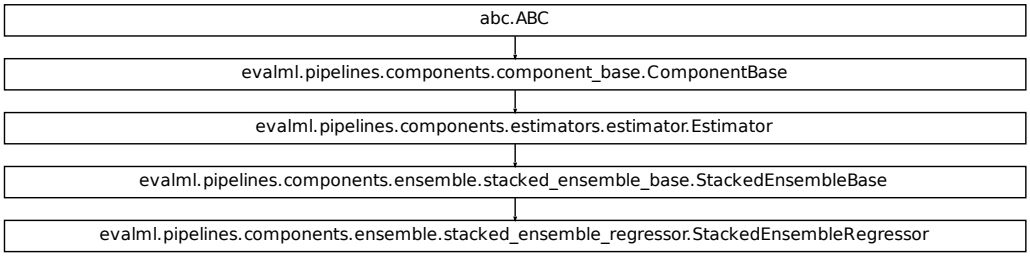
Saves component at file path

Parameters

- **file_path** (*str*) – Location to save file
- **pickle_protocol** (*int*) – The pickle data stream format.

Returns None

Class Inheritance



evalml.pipelines.components.DecisionTreeRegressor

class evalml.pipelines.components.**DecisionTreeRegressor**(*criterion='mse'*, *max_features='auto'*, *max_depth=6*, *min_samples_split=2*, *min_weight_fraction_leaf=0.0*, *random_seed=0*, ***kwargs*)

Decision Tree Regressor.

name = 'Decision Tree Regressor'

model_family = 'decision_tree'

supported_problem_types = [<ProblemTypes.REGRESSION: 'regression'>, <ProblemTypes.TIME

hyperparameter_ranges = {'criterion': ['mse', 'friedman_mse', 'mae'], 'max_depth': Int

default_parameters = {'criterion': 'mse', 'max_depth': 6, 'max_features': 'auto', 'min

predict_uses_y = False

Instance attributes

feature_importance	Returns importance associated with each feature.
needs_fitting	
parameters	Returns the parameters which were used to initialize the component

Methods:

__init__	Initialize self.
clone	Constructs a new component with the same parameters and random state.
describe	Describe a component and its parameters
fit	Fits component to data
load	Loads component at file path
predict	Make predictions using selected features.
predict_proba	Make probability estimates for labels.
save	Saves component at file path

evalml.pipelines.components.DecisionTreeRegressor.__init__

DecisionTreeRegressor.**__init__**(*criterion='mse'*, *max_features='auto'*, *max_depth=6*, *min_samples_split=2*, *min_weight_fraction_leaf=0.0*, *random_seed=0*, ***kwargs*)

Initialize self. See help(type(self)) for accurate signature.

evalml.pipelines.components.DecisionTreeRegressor.clone

DecisionTreeRegressor.**clone**()

Constructs a new component with the same parameters and random state.

Returns A new instance of this component with identical parameters and random state.

evalml.pipelines.components.DecisionTreeRegressor.describe

DecisionTreeRegressor.**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.DecisionTreeRegressor.fit

DecisionTreeRegressor.**fit**(*X*, *y=None*)

Fits component to data

Parameters

- **X** (*list, pd.DataFrame or np.ndarray*) – The input training data of shape [n_samples, n_features]

- **y** (*list, pd.Series, np.ndarray, optional*) – The target training data of length [n_samples]

Returns self

evalml.pipelines.components.DecisionTreeRegressor.load

static DecisionTreeRegressor.**load**(*file_path*)

Loads component at file path

Parameters **file_path** (*str*) – Location to load file

Returns ComponentBase object

evalml.pipelines.components.DecisionTreeRegressor.predict

DecisionTreeRegressor.**predict**(*X*)

Make predictions using selected features.

Parameters **X** (*pd.DataFrame, np.ndarray*) – Data of shape [n_samples, n_features]

Returns Predicted values

Return type pd.Series

evalml.pipelines.components.DecisionTreeRegressor.predict_proba

DecisionTreeRegressor.**predict_proba**(*X*)

Make probability estimates for labels.

Parameters **X** (*pd.DataFrame, or np.ndarray*) – Features

Returns Probability estimates

Return type pd.Series

evalml.pipelines.components.DecisionTreeRegressor.save

DecisionTreeRegressor.**save**(*file_path*, *pickle_protocol=5*)

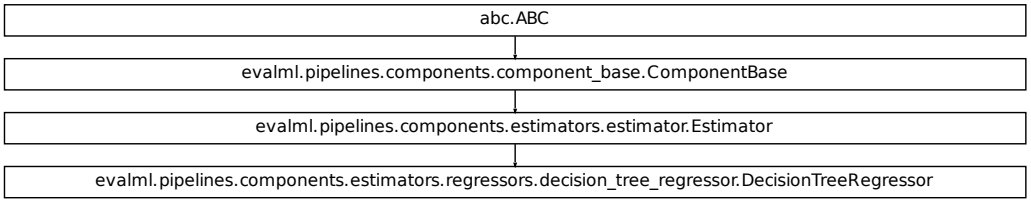
Saves component at file path

Parameters

- **file_path** (*str*) – Location to save file
- **pickle_protocol** (*int*) – The pickle data stream format.

Returns None

Class Inheritance



evalml.pipelines.components.LightGBMRegressor

class evalml.pipelines.components.**LightGBMRegressor**(*boosting_type='gbdt'*, *learning_rate=0.1*, *n_estimators=20*, *max_depth=0*, *num_leaves=31*, *min_child_samples=20*, *n_jobs=-1*, *random_seed=0*, *bagging_fraction=0.9*, *bagging_freq=0*, ***kwargs*)

LightGBM Regressor

name = 'LightGBM Regressor'

model_family = 'lightgbm'

supported_problem_types = [<ProblemTypes.REGRESSION: 'regression'>]

hyperparameter_ranges = {'bagging_fraction': Real(low=1e-06, high=1, prior='uniform',

default_parameters = {'bagging_fraction': 0.9, 'bagging_freq': 0, 'boosting_type': 'gb

predict_uses_y = False

Instance attributes

SEED_MAX	
SEED_MIN	
feature_importance	Returns importance associated with each feature.
needs_fitting	
parameters	Returns the parameters which were used to initialize the component

Methods:

__init__	Initialize self.
clone	Constructs a new component with the same parameters and random state.
describe	Describe a component and its parameters
fit	Fits component to data
load	Loads component at file path
predict	Make predictions using selected features.
predict_proba	Make probability estimates for labels.
save	Saves component at file path

evalml.pipelines.components.LightGBMRegressor.__init__

LightGBMRegressor.**__init__**(*boosting_type='gbdt'*, *learning_rate=0.1*, *n_estimators=20*, *max_depth=0*, *num_leaves=31*, *min_child_samples=20*, *n_jobs=- 1*, *random_seed=0*, *bagging_fraction=0.9*, *bagging_freq=0*, ***kwargs*)

Initialize self. See help(type(self)) for accurate signature.

evalml.pipelines.components.LightGBMRegressor.clone

LightGBMRegressor.**clone**()

Constructs a new component with the same parameters and random state.

Returns A new instance of this component with identical parameters and random state.

evalml.pipelines.components.LightGBMRegressor.describe

LightGBMRegressor.**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.LightGBMRegressor.fit

LightGBMRegressor.**fit**(*X*, *y=None*)

Fits component to data

Parameters

- **X** (*list, pd.DataFrame or np.ndarray*) – The input training data of shape [n_samples, n_features]
- **y** (*list, pd.Series, np.ndarray, optional*) – The target training data of length [n_samples]

Returns self

evalml.pipelines.components.LightGBMRegressor.load

static LightGBMRegressor.**load**(*file_path*)

Loads component at file path

Parameters **file_path** (*str*) – Location to load file

Returns ComponentBase object

evalml.pipelines.components.LightGBMRegressor.predict

LightGBMRegressor.**predict**(*X*)

Make predictions using selected features.

Parameters **X** (*pd.DataFrame, np.ndarray*) – Data of shape [n_samples, n_features]

Returns Predicted values

Return type pd.Series

evalml.pipelines.components.LightGBMRegressor.predict_proba

LightGBMRegressor.**predict_proba**(*X*)

Make probability estimates for labels.

Parameters **X** (*pd.DataFrame, or np.ndarray*) – Features

Returns Probability estimates

Return type pd.Series

evalml.pipelines.components.LightGBMRegressor.save

LightGBMRegressor.**save**(*file_path*, *pickle_protocol=5*)

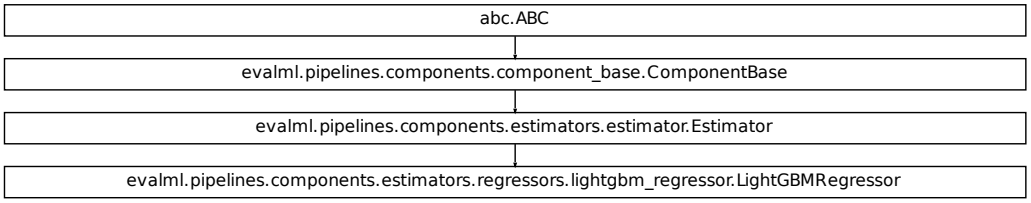
Saves component at file path

Parameters

- **file_path** (*str*) – Location to save file
- **pickle_protocol** (*int*) – The pickle data stream format.

Returns None

Class Inheritance



evalml.pipelines.components.SVMRegressor

class evalml.pipelines.components.**SVMRegressor**(*C=1.0*, *kernel='rbf'*, *gamma='scale'*, *random_seed=0*, ***kwargs*)

Support Vector Machine Regressor.

name = 'SVM Regressor'

model_family = 'svm'

supported_problem_types = [<ProblemTypes.REGRESSION: 'regression'>, <ProblemTypes.TIME

hyperparameter_ranges = {'C': Real(low=0, high=10, prior='uniform', transform='identit

default_parameters = {'C': 1.0, 'gamma': 'scale', 'kernel': 'rbf'}

predict_uses_y = False

Instance attributes

feature_importance	Feature importance only works with linear kernels.
needs_fitting	
parameters	Returns the parameters which were used to initialize the component

Methods:

__init__	Initialize self.
clone	Constructs a new component with the same parameters and random state.
describe	Describe a component and its parameters
fit	Fits component to data
load	Loads component at file path
predict	Make predictions using selected features.

continues on next page

Table 131 – continued from previous page

predict_proba	Make probability estimates for labels.
save	Saves component at file path

evalml.pipelines.components.SVMRegressor.__init__

SVMRegressor.**__init__**(*C=1.0*, *kernel='rbf'*, *gamma='scale'*, *random_seed=0*, ***kwargs*)

Initialize self. See help(type(self)) for accurate signature.

evalml.pipelines.components.SVMRegressor.clone

SVMRegressor.**clone**()

Constructs a new component with the same parameters and random state.

Returns A new instance of this component with identical parameters and random state.

evalml.pipelines.components.SVMRegressor.describe

SVMRegressor.**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.SVMRegressor.fit

SVMRegressor.**fit**(*X*, *y=None*)

Fits component to data

Parameters

- **X** (*list, pd.DataFrame or np.ndarray*) – The input training data of shape [n_samples, n_features]
- **y** (*list, pd.Series, np.ndarray, optional*) – The target training data of length [n_samples]

Returns self

evalml.pipelines.components.SVMRegressor.load

static SVMRegressor.**load**(*file_path*)
Loads component at file path

Parameters **file_path** (*str*) – Location to load file

Returns ComponentBase object

evalml.pipelines.components.SVMRegressor.predict

SVMRegressor.**predict**(*X*)
Make predictions using selected features.

Parameters **X** (*pd.DataFrame, np.ndarray*) – Data of shape [n_samples, n_features]

Returns Predicted values

Return type pd.Series

evalml.pipelines.components.SVMRegressor.predict_proba

SVMRegressor.**predict_proba**(*X*)
Make probability estimates for labels.

Parameters **X** (*pd.DataFrame, or np.ndarray*) – Features

Returns Probability estimates

Return type pd.Series

evalml.pipelines.components.SVMRegressor.save

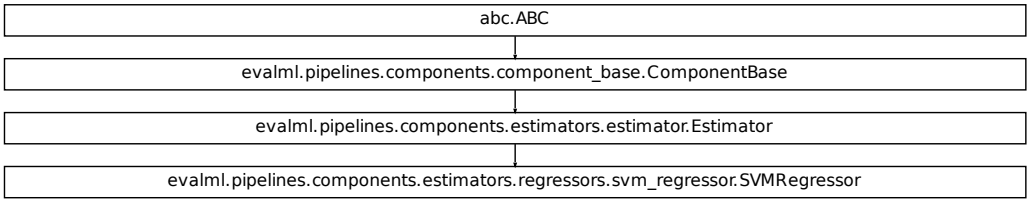
SVMRegressor.**save**(*file_path*, *pickle_protocol=5*)
Saves component at file path

Parameters

- **file_path** (*str*) – Location to save file
- **pickle_protocol** (*int*) – The pickle data stream format.

Returns None

Class Inheritance



5.7 Model Understanding

5.7.1 Utility Methods

confusion_matrix	Confusion matrix for binary and multiclass classification.
normalize_confusion_matrix	Normalizes a confusion matrix.
precision_recall_curve	Given labels and binary classifier predicted probabilities, compute and return the data representing a precision-recall curve.
roc_curve	Given labels and classifier predicted probabilities, compute and return the data representing a Receiver Operating Characteristic (ROC) curve.
calculate_permutation_importance	Calculates permutation importance for features.
calculate_permutation_importance_one_col	Calculates permutation importance for one column in the original dataframe.
binary_objective_vs_threshold	Computes objective score as a function of potential binary classification
get_prediction_vs_actual_over_time_data	Get the data needed for the prediction_vs_actual_over_time plot.
partial_dependence	Calculates one or two-way partial dependence.
get_prediction_vs_actual_data	Combines y_true and y_pred into a single dataframe and adds a column for outliers.
get_linear_coefficients	Returns a dataframe showing the features with the greatest predictive power for a linear model.
t_sne	Get the transformed output after fitting X to the embedded space using t-SNE.

evalml.model_understanding.confusion_matrix

evalml.model_understanding.**confusion_matrix**(*y_true*, *y_predicted*, *normalize_method='true'*)

Confusion matrix for binary and multiclass classification.

Parameters

- **y_true** (*pd.Series or np.ndarray*) – True binary labels.
- **y_pred** (*pd.Series or np.ndarray*) – Predictions from a binary classifier.
- **normalize_method** (*{'true', 'pred', 'all', None}*) – Normalization method to use, if not None. Supported options are: 'true' to normalize by row, 'pred' to normalize by column, or 'all' to normalize by all values. Defaults to 'true'.

Returns Confusion matrix. The column header represents the predicted labels while row header represents the actual labels.

Return type pd.DataFrame

evalml.model_understanding.normalize_confusion_matrix

evalml.model_understanding.**normalize_confusion_matrix**(*conf_mat*, *normalize_method='true'*)

Normalizes a confusion matrix.

Parameters

- **conf_mat** (*pd.DataFrame or np.ndarray*) – Confusion matrix to normalize.
- **normalize_method** (*{'true', 'pred', 'all'}*) – Normalization method. Supported options are: 'true' to normalize by row, 'pred' to normalize by column, or 'all' to normalize by all values. Defaults to 'true'.

Returns normalized version of the input confusion matrix. The column header represents the predicted labels while row header represents the actual labels.

Return type pd.DataFrame

evalml.model_understanding.precision_recall_curve

evalml.model_understanding.**precision_recall_curve**(*y_true*, *y_pred_proba*, *pos_label_idx=- 1*)

Given labels and binary classifier predicted probabilities, compute and return the data representing a precision-recall curve.

Parameters

- **y_true** (*pd.Series or np.ndarray*) – True binary labels.
- **y_pred_proba** (*pd.Series or np.ndarray*) – Predictions from a binary classifier, before thresholding has been applied. Note this should be the predicted probability for the "true" label.
- **pos_label_idx** (*int*) – the column index corresponding to the positive class. If predicted probabilities are two-dimensional, this will be used to access the probabilities for the positive class.

Returns

Dictionary containing metrics used to generate a precision-recall plot, with the following keys:

- *precision*: Precision values.
- *recall*: Recall values.
- *thresholds*: Threshold values used to produce the precision and recall.
- *auc_score*: The area under the ROC curve.

Return type list

evalml.model_understanding.roc_curve

evalml.model_understanding.**roc_curve**(*y_true*, *y_pred_proba*)

Given labels and classifier predicted probabilities, compute and return the data representing a Receiver Operating Characteristic (ROC) curve. Works with binary or multiclass problems.

Parameters

- **y_true** (*pd.Series or np.ndarray*) – True labels.
- **y_pred_proba** (*pd.Series or np.ndarray*) – Predictions from a classifier, before thresholding has been applied.

Returns

A list of dictionaries (with one for each class) is returned. Binary classification problems return a list with one di

Each dictionary contains metrics used to generate an ROC plot with the following keys:

- *fpr_rate*: False positive rate.
- *tpr_rate*: True positive rate.
- *threshold*: Threshold values used to produce each pair of true/false positive rates.
- *auc_score*: The area under the ROC curve.

Return type list(dict)

evalml.model_understanding.calculate_permutation_importance

evalml.model_understanding.**calculate_permutation_importance**(*pipeline*, *X*, *y*, *objective*, *n_repeats=5*, *n_jobs=None*, *random_seed=0*)

Calculates permutation importance for features.

Parameters

- **pipeline** (*PipelineBase or subclass*) – Fitted pipeline.
- **X** (*pd.DataFrame*) – The input data used to score and compute permutation importance.
- **y** (*pd.Series*) – The target data.
- **objective** (*str, ObjectiveBase*) – Objective to score on.
- **n_repeats** (*int*) – Number of times to permute a feature. Defaults to 5.
- **n_jobs** (*int or None*) – Non-negative integer describing level of parallelism used for pipelines. None and 1 are equivalent. If set to -1, all CPUs are used. For n_jobs below -1, (n_cpus + 1 + n_jobs) are used. Defaults to None.

- **random_seed** (*int*) – Seed for the random number generator. Defaults to 0.

Returns Mean feature importance scores over a number of shuffles.

Return type pd.DataFrame

evalml.model_understanding.calculate_permutation_importance_one_column

evalml.model_understanding.**calculate_permutation_importance_one_column**(*pipeline*, *X*, *y*, *col_name*, *objective*, *n_repeats=5*, *fast=True*, *precomputed_features=None*, *random_seed=0*)

Calculates permutation importance for one column in the original dataframe.

Parameters

- **pipeline** (*PipelineBase or subclass*) – Fitted pipeline.
- **X** (*pd.DataFrame*) – The input data used to score and compute permutation importance.
- **y** (*pd.Series*) – The target data.
- **col_name** (*str, int*) – The column in X to calculate permutation importance for.
- **objective** (*str, ObjectiveBase*) – Objective to score on.
- **n_repeats** (*int*) – Number of times to permute a feature. Defaults to 5.
- **fast** (*bool*) – Whether to use the fast method of calculating the permutation importance or not. Defaults to True.
- **precomputed_features** (*pd.DataFrame*) – Precomputed features necessary to calculate permutation importance using the fast method. Defaults to None.
- **random_seed** (*int*) – Seed for the random number generator. Defaults to 0.

Returns Mean feature importance scores over a number of shuffles.

Return type float

evalml.model_understanding.binary_objective_vs_threshold

evalml.model_understanding.**binary_objective_vs_threshold**(*pipeline*, *X*, *y*, *objective*, *steps=100*)

Computes objective score as a function of potential binary classification decision thresholds for a fitted binary classification pipeline.

Parameters

- **pipeline** (*BinaryClassificationPipeline obj*) – Fitted binary classification pipeline

- **X** (*pd.DataFrame*) – The input data used to compute objective score
- **y** (*pd.Series*) – The target labels
- **objective** (*ObjectiveBase obj, str*) – Objective used to score
- **steps** (*int*) – Number of intervals to divide and calculate objective score at

Returns DataFrame with thresholds and the corresponding objective score calculated at each threshold

Return type pd.DataFrame

evalml.model_understanding.get_prediction_vs_actual_over_time_data

evalml.model_understanding.**get_prediction_vs_actual_over_time_data**(*pipeline*, *X*, *y*, *dates*)

Get the data needed for the prediction_vs_actual_over_time plot.

Parameters

- **pipeline** (TimeSeriesRegressionPipeline) – Fitted time series regression pipeline.
- **X** (*pd.DataFrame*) – Features used to generate new predictions.
- **y** (*pd.Series*) – Target values to compare predictions against.
- **dates** (*pd.Series*) – Dates corresponding to target values and predictions.

Returns pd.DataFrame

evalml.model_understanding.partial_dependence

evalml.model_understanding.**partial_dependence**(*pipeline*, *X*, *features*, *percentiles=(0.05, 0.95)*, *grid_resolution=100*, *kind='average'*)

Calculates one or two-way partial dependence. If a single integer or string is given for features, one-way partial dependence is calculated. If a tuple of two integers or strings is given, two-way partial dependence is calculated with the first feature in the y-axis and second feature in the x-axis.

Parameters

- **pipeline** (PipelineBase *or subclass*) – Fitted pipeline
- **X** (*pd.DataFrame, np.ndarray*) – The input data used to generate a grid of values for feature where partial dependence will be calculated at
- **features** (*int, string, tuple[int or string]*) – The target feature for which to create the partial dependence plot for. If features is an int, it must be the index of the feature to use. If features is a string, it must be a valid column name in X. If features is a tuple of int/strings, it must contain valid column integers/names in X.
- **percentiles** (*tuple[float]*) – The lower and upper percentile used to create the extreme values for the grid. Must be in [0, 1]. Defaults to (0.05, 0.95).
- **grid_resolution** (*int*) – Number of samples of feature(s) for partial dependence plot. If this value is less than the maximum number of categories present in categorical data within X, it will be set to the max number of categories + 1. Defaults to 100.

- **{'average'** (*kind*) – The type of predictions to return. 'individual' will return the predictions for all of the points in the grid for each sample in X. 'average' will return the predictions for all of the points in the grid but averaged over all of the samples in X.
- **'individual'** – The type of predictions to return. 'individual' will return the predictions for all of the points in the grid for each sample in X. 'average' will return the predictions for all of the points in the grid but averaged over all of the samples in X.
- **'both'}** – The type of predictions to return. 'individual' will return the predictions for all of the points in the grid for each sample in X. 'average' will return the predictions for all of the points in the grid but averaged over all of the samples in X.

Returns

When *kind='average'*: DataFrame with averaged predictions for all points in the grid averaged over all samples of X and the values used to calculate those predictions.

When *kind='individual'*: DataFrame with individual predictions for all points in the grid for each sample of X and the values used to calculate those predictions. If a two-way partial dependence is calculated, then the result is a list of DataFrames with each DataFrame representing one sample's predictions.

When *kind='both'*: A tuple consisting of the averaged predictions (in a DataFrame) over all samples of X and the individual predictions (in a list of DataFrames) for each sample of X.

In the one-way case: The dataframe will contain two columns, "feature_values" (grid points at which the partial dependence was calculated) and "partial_dependence" (the partial dependence at that feature value). For classification problems, there will be a third column called "class_label" (the class label for which the partial dependence was calculated). For binary classification, the partial dependence is only calculated for the "positive" class.

In the two-way case: The data frame will contain grid_resolution number of columns and rows where the index and column headers are the sampled values of the first and second features, respectively, used to make the partial dependence contour. The values of the data frame contain the partial dependence data for each feature value pair.

Return type pd.DataFrame, list(pd.DataFrame), or tuple(pd.DataFrame, list(pd.DataFrame))

Raises

- **ValueError** – if the user provides a tuple of not exactly two features.
- **ValueError** – if the provided pipeline isn't fitted.
- **ValueError** – if the provided pipeline is a Baseline pipeline.
- **ValueError** – if any of the features passed in are completely NaN
- **ValueError** – if any of the features are low-variance. Defined as having one value occurring more than the upper percentile passed by the user. By default 95%.

evalml.model_understanding.get_prediction_vs_actual_data

evalml.model_understanding.**get_prediction_vs_actual_data**(*y_true*, *y_pred*, *outlier_threshold=None*)

Combines y_true and y_pred into a single dataframe and adds a column for outliers. Used in *graph_prediction_vs_actual()*.

Parameters

- **y_true** (*pd.Series, or np.ndarray*) – The real target values of the data
- **y_pred** (*pd.Series, or np.ndarray*) – The predicted values outputted by the regression model.
- **outlier_threshold** (*int, float*) – A positive threshold for what is considered an outlier value. This value is compared to the absolute difference between each value of y_true and y_pred. Values within this threshold will be blue, otherwise they will be yellow. Defaults to None

Returns

- *prediction*: Predicted values from regression model.
- *actual*: Real target values.
- *outlier*: Colors indicating which values are in the threshold for what is considered an outlier value.

Return type pd.DataFrame with the following columns

evalml.model_understanding.get_linear_coefficients

evalml.model_understanding.**get_linear_coefficients**(*estimator*, *features=None*)

Returns a dataframe showing the features with the greatest predictive power for a linear model.

Parameters

- **estimator** (Estimator) – Fitted linear model family estimator.
- **features** (*list[str]*) – List of feature names associated with the underlying data.

Returns Displaying the features by importance.

Return type pd.DataFrame

evalml.model_understanding.t_sne

evalml.model_understanding.**t_sne**(*X*, *n_components=2*, *perplexity=30.0*, *learning_rate=200.0*, *metric='euclidean'*, ***kwargs*)

Get the transformed output after fitting X to the embedded space using t-SNE.

Arguments: X (np.ndarray, pd.DataFrame): Data to be transformed. Must be numeric. n_components (int, optional): Dimension of the embedded space. perplexity (float, optional): Related to the number of nearest neighbors that is used in other manifold learning algorithms. Larger datasets usually require a larger perplexity. Consider selecting a value between 5 and 50. learning_rate (float, optional): Usually in the range [10.0, 1000.0]. If the cost function gets stuck in a bad local minimum, increasing the learning rate may help. metric (str, optional): The metric to use when calculating distance between instances in a feature array.

Returns np.ndarray (n_samples, n_components)

5.7.2 Graph Utility Methods

graph_precision_recall_curve	Generate and display a precision-recall plot.
graph_roc_curve	Generate and display a Receiver Operating Characteristic (ROC) plot for binary and multiclass classification problems.
graph_confusion_matrix	Generate and display a confusion matrix plot.
graph_permutation_importance	Generate a bar graph of the pipeline's permutation importance.
graph_binary_objective_vs_threshold	Generates a plot graphing objective score vs.
graph_prediction_vs_actual	Generate a scatter plot comparing the true and predicted values.
graph_prediction_vs_actual_over_time	Plot the target values and predictions against time on the x-axis.
graph_partial_dependence	Create an one-way or two-way partial dependence plot.
graph_t_sne	Plot high dimensional data into lower dimensional space using t-SNE .

evalml.model_understanding.graph_precision_recall_curve

evalml.model_understanding.**graph_precision_recall_curve**(*y_true*, *y_pred_proba*, *title_addition=None*)

Generate and display a precision-recall plot.

Parameters

- **y_true** (*pd.Series or np.ndarray*) – True binary labels.
- **y_pred_proba** (*pd.Series or np.ndarray*) – Predictions from a binary classifier, before thresholding has been applied. Note this should be the predicted probability for the "true" label.
- **title_addition** (*str or None*) – If not None, append to plot title. Default None.

Returns plotly.Figure representing the precision-recall plot generated

evalml.model_understanding.graph_roc_curve

evalml.model_understanding.**graph_roc_curve**(*y_true*, *y_pred_proba*, *custom_class_names=None*, *title_addition=None*)

Generate and display a Receiver Operating Characteristic (ROC) plot for binary and multiclass classification problems.

Parameters

- **y_true** (*pd.Series or np.ndarray*) – True labels.
- **y_pred_proba** (*pd.Series or np.ndarray*) – Predictions from a classifier, before thresholding has been applied. Note this should a one dimensional array with the predicted probability for the "true" label in the binary case.
- **custom_class_labels** (*list or None*) – If not None, custom labels for classes. Default None.
- **title_addition** (*str or None*) – if not None, append to plot title. Default None.

Returns plotly.Figure representing the ROC plot generated

evalml.model_understanding.graph_confusion_matrix

evalml.model_understanding.**graph_confusion_matrix**(*y_true*, *y_pred*, *normalize_method='true'*, *title_addition=None*)

Generate and display a confusion matrix plot.

If *normalize_method* is set, hover text will show raw count, otherwise hover text will show count normalized with method 'true'.

Parameters

- **y_true** (*pd.Series or np.ndarray*) – True binary labels.
- **y_pred** (*pd.Series or np.ndarray*) – Predictions from a binary classifier.
- **normalize_method** (*{'true', 'pred', 'all', None}*) – Normalization method to use, if not None. Supported options are: 'true' to normalize by row, 'pred' to normalize by column, or 'all' to normalize by all values. Defaults to 'true'.
- **title_addition** (*str or None*) – if not None, append to plot title. Defaults to None.

Returns plotly.Figure representing the confusion matrix plot generated

evalml.model_understanding.graph_permutation_importance

evalml.model_understanding.**graph_permutation_importance**(*pipeline*, *X*, *y*, *objective*, *importance_threshold=0*)

Generate a bar graph of the pipeline's permutation importance.

Parameters

- **pipeline** (*PipelineBase or subclass*) – Fitted pipeline
- **X** (*pd.DataFrame*) – The input data used to score and compute permutation importance
- **y** (*pd.Series*) – The target data
- **objective** (*str, ObjectiveBase*) – Objective to score on
- **importance_threshold** (*float, optional*) – If provided, graph features with a permutation importance whose absolute value is larger than importance_threshold. Defaults to zero.

Returns plotly.Figure, a bar graph showing features and their respective permutation importance.

evalml.model_understanding.graph_binary_objective_vs_threshold

evalml.model_understanding.**graph_binary_objective_vs_threshold**(*pipeline*, *X*, *y*, *objective*, *steps=100*)

Generates a plot graphing objective score vs. decision thresholds for a fitted binary classification pipeline.

Parameters

- **pipeline** (*PipelineBase or subclass*) – Fitted pipeline
- **X** (*pd.DataFrame*) – The input data used to score and compute scores

- **y** (*pd.Series*) – The target labels
- **objective** (*ObjectiveBase obj, str*) – Objective used to score, shown on the y-axis of the graph
- **steps** (*int*) – Number of intervals to divide and calculate objective score at

Returns plotly.Figure representing the objective score vs. threshold graph generated

evalml.model_understanding.graph_prediction_vs_actual

evalml.model_understanding.**graph_prediction_vs_actual**(*y_true*, *y_pred*, *outlier_threshold=None*)

Generate a scatter plot comparing the true and predicted values. Used for regression plotting

Parameters

- **y_true** (*pd.Series*) – The real target values of the data
- **y_pred** (*pd.Series*) – The predicted values outputted by the regression model.
- **outlier_threshold** (*int, float*) – A positive threshold for what is considered an outlier value. This value is compared to the absolute difference between each value of y_true and y_pred. Values within this threshold will be blue, otherwise they will be yellow. Defaults to None

Returns plotly.Figure representing the predicted vs. actual values graph

evalml.model_understanding.graph_prediction_vs_actual_over_time

evalml.model_understanding.**graph_prediction_vs_actual_over_time**(*pipeline*, *X*, *y*, *dates*)

Plot the target values and predictions against time on the x-axis.

Parameters

- **pipeline** (TimeSeriesRegressionPipeline) – Fitted time series regression pipeline.
- **X** (*pd.DataFrame*) – Features used to generate new predictions.
- **y** (*pd.Series*) – Target values to compare predictions against.
- **dates** (*pd.Series*) – Dates corresponding to target values and predictions.

Returns Showing the prediction vs actual over time.

Return type plotly.Figure

evalml.model_understanding.graph_partial_dependence

evalml.model_understanding.**graph_partial_dependence**(*pipeline*, *X*, *features*, *class_label=None*, *grid_resolution=100*, *kind='average'*)

Create an one-way or two-way partial dependence plot. Passing a single integer or string as features will create a one-way partial dependence plot with the feature values plotted against the partial dependence. Passing features a tuple of int/strings will create a two-way partial dependence plot with a contour of feature[0] in the y-axis, feature[1] in the x-axis and the partial dependence in the z-axis.

Parameters

- **pipeline** (*PipelineBase or subclass*) – Fitted pipeline
- **X** (*pd.DataFrame, np.ndarray*) – The input data used to generate a grid of values for feature where partial dependence will be calculated at
- **features** (*int, string, tuple[int or string]*) – The target feature for which to create the partial dependence plot for. If features is an int, it must be the index of the feature to use. If features is a string, it must be a valid column name in X. If features is a tuple of strings, it must contain valid column int/names in X.
- **class_label** (*string, optional*) – Name of class to plot for multiclass problems. If None, will plot the partial dependence for each class. This argument does not change behavior for regression or binary classification pipelines. For binary classification, the partial dependence for the positive label will always be displayed. Defaults to None.
- **grid_resolution** (*int*) – Number of samples of feature(s) for partial dependence plot
- **{'average'** (*kind*) – Type of partial dependence to plot. 'average' creates a regular partial dependence (PD) graph, 'individual' creates an individual conditional expectation (ICE) plot, and 'both' creates a single-figure PD and ICE plot. ICE plots can only be shown for one-way partial dependence plots.
- **'individual'** – Type of partial dependence to plot. 'average' creates a regular partial dependence (PD) graph, 'individual' creates an individual conditional expectation (ICE) plot, and 'both' creates a single-figure PD and ICE plot. ICE plots can only be shown for one-way partial dependence plots.
- **'both'}** – Type of partial dependence to plot. 'average' creates a regular partial dependence (PD) graph, 'individual' creates an individual conditional expectation (ICE) plot, and 'both' creates a single-figure PD and ICE plot. ICE plots can only be shown for one-way partial dependence plots.

Returns figure object containing the partial dependence data for plotting

Return type plotly.graph_objects.Figure

Raises **ValueError** – if a graph is requested for a class name that isn't present in the pipeline

evalml.model_understanding.graph_t_sne

evalml.model_understanding.**graph_t_sne**(*X*, *n_components=2*, *perplexity=30.0*, *learning_rate=200.0*, *metric='euclidean'*, *marker_line_width=2*, *marker_size=7*, ***kwargs*)

Plot high dimensional data into lower dimensional space using t-SNE .

Parameters

- **X** (*np.ndarray, pd.DataFrame*) – Data to be transformed. Must be numeric.
- **n_components** (*int, optional*) – Dimension of the embedded space.
- **perplexity** (*float, optional*) – Related to the number of nearest neighbors that is used in other manifold learning
- **Larger datasets usually require a larger perplexity. Consider selecting a value between 5 and 50.** (*algorithms.*) –
- **learning_rate** (*float, optional*) – Usually in the range [10.0, 1000.0]. If the cost function gets stuck in a bad
- **minimum** (*local*) –

- **the learning rate may help.** (*increasing*) –
- **metric** (*str, optional*) – The metric to use when calculating distance between instances in a feature array.
- **marker_line_width** (*int, optional*) – Determines the line width of the marker boundary.
- **marker_size** (*int, optional*) – Determines the size of the marker.

Returns plotly.Figure representing the transformed data

5.7.3 Prediction Explanations

explain_predictions	Creates a report summarizing the top contributing features for each data point in the input features.
explain_predictions_best_worst	Creates a report summarizing the top contributing features for the best and worst points in the dataset as measured by error to true labels.

evalml.model_understanding.prediction_explanations.explain_predictions

evalml.model_understanding.prediction_explanations.**explain_predictions**(*pipeline*, *input_features*, *y*, *indices_to_explain*, *top_k_features=3*, *include_shap_values=False*, *include_expected_value=False*, *output_format='text'*)

Creates a report summarizing the top contributing features for each data point in the input features.

XGBoost and Stacked Ensemble models, as well as CatBoost multiclass classifiers, are not currently supported.

Parameters

- **pipeline** (PipelineBase) – Fitted pipeline whose predictions we want to explain with SHAP.
- **input_features** (*pd.DataFrame*) – Dataframe of input data to evaluate the pipeline on.
- **y** (*pd.Series*) – Labels for the input data.
- **indices_to_explain** (*list(int)*) – List of integer indices to explain.
- **top_k_features** (*int*) – How many of the highest/lowest contributing feature to include in the table for each data point. Default is 3.
- **include_shap_values** (*bool*) – Whether SHAP values should be included in the table. Default is False.

- **include_expected_value** (*bool*) – Whether the expected value should be included in the table. Default is False.
- **output_format** (*str*) – Either "text", "dict", or "dataframe". Default is "text".

Returns

str, dict, or pd.DataFrame - A report explaining the top contributing features to each prediction for each row of
The report will include the feature names, prediction contribution, and SHAP Value (optional).

Raises

- **ValueError** – if input_features is empty.
- **ValueError** – if an output_format outside of "text", "dict" or "dataframe is provided.
- **ValueError** – if the requested index falls outside the input_feature's boundaries.

evalml.model_understanding.prediction_explanations.explain_predictions_best_worst

evalml.model_understanding.prediction_explanations.**explain_predictions_best_worst**(*pipeline*, *input_features*, *y_true*, *num_to_expl*, *top_k_featur*, *include_shap_*, *metric=None*, *output_format=*, *callback=None*)

Creates a report summarizing the top contributing features for the best and worst points in the dataset as measured by error to true labels.

XGBoost and Stacked Ensemble models, as well as CatBoost multiclass classifiers, are not currently supported.

Parameters

- **pipeline** (PipelineBase) – Fitted pipeline whose predictions we want to explain with SHAP.
- **input_features** (*pd.DataFrame*) – Input data to evaluate the pipeline on.
- **y_true** (*pd.Series*) – True labels for the input data.
- **num_to_explain** (*int*) – How many of the best, worst, random data points to explain.
- **top_k_features** (*int*) – How many of the highest/lowest contributing feature to include in the table for each data point.
- **include_shap_values** (*bool*) – Whether SHAP values should be included in the table. Default is False.
- **metric** (*callable*) – The metric used to identify the best and worst points in the dataset. Function must accept the true labels and predicted value or probabilities as the only arguments and lower values must be better. By default, this will be the absolute error for regression problems and cross entropy loss for classification problems.

- **output_format** (*str*) – Either "text" or "dict". Default is "text".
- **callback** (*callable*) – Function to be called with incremental updates. Has the following parameters: - progress_stage: stage of computation - time_elapsed: total time in seconds that has elapsed since start of call

Returns

str, dict, or pd.DataFrame - A report explaining the top contributing features for the best/worst predictions in th
For each of the best/worst rows of input_features, the predicted values, true labels, metric value, feature names, prediction contribution, and SHAP Value (optional) will be listed.

Raises

- **ValueError** – if input_features does not have more than twice the requested features to explain.
- **ValueError** – if y_true and input_features have mismatched lengths.
- **ValueError** – if an output_format outside of "text", "dict" or "dataframe is provided.

5.8 Objective Functions

5.8.1 Objective Base Classes

ObjectiveBase	Base class for all objectives.
BinaryClassificationObjective	Base class for all binary classification objectives.
MulticlassClassificationObjective	Base class for all multiclass classification objectives.
RegressionObjective	Base class for all regression objectives.

evalml.objectives.ObjectiveBase

class evalml.objectives.**ObjectiveBase**
Base class for all objectives.

problem_types = None

Methods

__init__	Initialize self.
calculate_percent_difference	Calculate the percent difference between scores.
is_defined_for_problem_type	
objective_function	Computes the relative value of the provided predictions compared to the actual labels, according a specified metric
score	Returns a numerical score indicating performance based on the differences between the predicted and actual values.
validate_inputs	Validates the input based on a few simple checks.

evalml.objectives.ObjectiveBase.__init__

ObjectiveBase.**__init__**()
Initialize self. See help(type(self)) for accurate signature.

evalml.objectives.ObjectiveBase.calculate_percent_difference

classmethod ObjectiveBase.**calculate_percent_difference**(*score*, *baseline_score*)
Calculate the percent difference between scores.

Parameters

- **score** (*float*) – A score. Output of the score method of this objective.
- **baseline_score** (*float*) – A score. Output of the score method of this objective. In practice, this is the score achieved on this objective with a baseline estimator.

Returns

The percent difference between the scores. Note that for objectives that can be interpreted as percentages, this will be the difference between the reference score and score. For all other objectives, the difference will be normalized by the reference score.

Return type float

evalml.objectives.ObjectiveBase.is_defined_for_problem_type

classmethod ObjectiveBase.**is_defined_for_problem_type**(*problem_type*)

evalml.objectives.ObjectiveBase.objective_function

abstract classmethod ObjectiveBase.**objective_function**(*y_true*, *y_predicted*, *X=None*, *sample_weight=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.ndarray): Extra data of shape [n_samples, n_features] necessary to calculate score sample_weight (pd.DataFrame or np.ndarray): Sample weights used in computing objective value result

Returns Numerical value used to calculate score

evalml.objectives.ObjectiveBase.score

ObjectiveBase.**score**(*y_true*, *y_predicted*, *X=None*, *sample_weight=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.ndarray*) – Extra data of shape [n_samples, n_features] necessary to calculate score
- **sample_weight** (*pd.DataFrame or np.ndarray*) – Sample weights used in computing objective value result

Returns score

evalml.objectives.ObjectiveBase.validate_inputs

ObjectiveBase.**validate_inputs**(*y_true*, *y_predicted*)

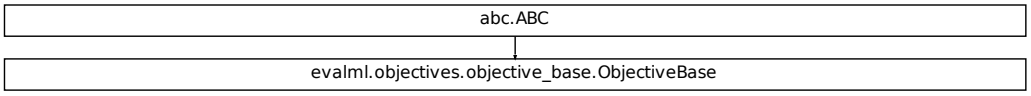
Validates the input based on a few simple checks.

Parameters

- **y_predicted** (*pd.Series, or pd.DataFrame*) – Predicted values of length [n_samples]
- **y_true** (*pd.Series*) – Actual class labels of length [n_samples]

Returns None

Class Inheritance



evalml.objectives.BinaryClassificationObjective

class evalml.objectives.**BinaryClassificationObjective**
Base class for all binary classification objectives.

problem_types = [<ProblemTypes.BINARY: 'binary'>, <ProblemTypes.TIME_SERIES_BINARY: 't

Methods

__init__	Initialize self.
calculate_percent_difference	Calculate the percent difference between scores.
decision_function	Apply a learned threshold to predicted probabilities to get predicted classes.
is_defined_for_problem_type	
objective_function	Computes the relative value of the provided predictions compared to the actual labels, according a specified metric
optimize_threshold	Learn a binary classification threshold which optimizes the current objective.
score	Returns a numerical score indicating performance based on the differences between the predicted and actual values.
validate_inputs	Validates the input based on a few simple checks.

evalml.objectives.BinaryClassificationObjective.__init__

BinaryClassificationObjective.**__init__**()
Initialize self. See help(type(self)) for accurate signature.

evalml.objectives.BinaryClassificationObjective.calculate_percent_difference

classmethod BinaryClassificationObjective.**calculate_percent_difference**(*score*, *baseline_score*)
Calculate the percent difference between scores.

Parameters

- **score** (*float*) – A score. Output of the score method of this objective.
- **baseline_score** (*float*) – A score. Output of the score method of this objective. In practice, this is the score achieved on this objective with a baseline estimator.

Returns

The percent difference between the scores. Note that for objectives that can be interpreted as percentages, this will be the difference between the reference score and score. For all other objectives, the difference will be normalized by the reference score.

Return type float

evalml.objectives.BinaryClassificationObjective.decision_function

BinaryClassificationObjective.**decision_function**(*ypred_proba*, *threshold=0.5*, *X=None*)

Apply a learned threshold to predicted probabilities to get predicted classes.

Parameters

- **ypred_proba** (*pd.Series, np.ndarray*) – The classifier's predicted probabilities
- **threshold** (*float, optional*) – Threshold used to make a prediction. Defaults to 0.5.
- **X** (*pd.DataFrame, optional*) – Any extra columns that are needed from training data.

Returns predictions

evalml.objectives.BinaryClassificationObjective.is_defined_for_problem_type

classmethod BinaryClassificationObjective.**is_defined_for_problem_type**(*problem_type*)

evalml.objectives.BinaryClassificationObjective.objective_function

abstract classmethod BinaryClassificationObjective.**objective_function**(*y_true*, *y_predicted*, *X=None*, *sample_weight=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.ndarray): Extra data of shape [n_samples, n_features] necessary to calculate score sample_weight (pd.DataFrame or np.ndarray): Sample weights used in computing objective value result

Returns Numerical value used to calculate score

evalml.objectives.BinaryClassificationObjective.optimize_threshold

BinaryClassificationObjective.**optimize_threshold**(*ypred_proba*, *y_true*, *X=None*)

Learn a binary classification threshold which optimizes the current objective.

Parameters

- **ypred_proba** (*pd.Series*) – The classifier's predicted probabilities
- **y_true** (*pd.Series*) – The ground truth for the predictions.
- **X** (*pd.DataFrame, optional*) – Any extra columns that are needed from training data.

Returns Optimal threshold for this objective

evalml.objectives.BinaryClassificationObjective.score

BinaryClassificationObjective.**score**(*y_true*, *y_predicted*, *X=None*, *sample_weight=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.ndarray*) – Extra data of shape [n_samples, n_features] necessary to calculate score
- **sample_weight** (*pd.DataFrame or np.ndarray*) – Sample weights used in computing objective value result

Returns score

evalml.objectives.BinaryClassificationObjective.validate_inputs

BinaryClassificationObjective.**validate_inputs**(*y_true*, *y_predicted*)

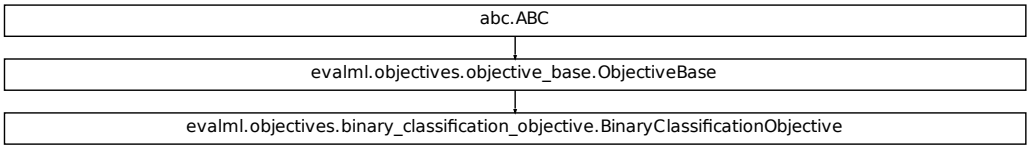
Validates the input based on a few simple checks.

Parameters

- **y_predicted** (*pd.Series, or pd.DataFrame*) – Predicted values of length [n_samples]
- **y_true** (*pd.Series*) – Actual class labels of length [n_samples]

Returns None

Class Inheritance



evalml.objectives.MulticlassClassificationObjective

class evalml.objectives.**MulticlassClassificationObjective**

Base class for all multiclass classification objectives.

problem_types = [<ProblemTypes.MULTICLASS: 'multiclass'>, <ProblemTypes.TIME_SERIES_MU

Methods

__init__	Initialize self.
calculate_percent_difference	Calculate the percent difference between scores.
is_defined_for_problem_type	
objective_function	Computes the relative value of the provided predictions compared to the actual labels, according a specified metric
score	Returns a numerical score indicating performance based on the differences between the predicted and actual values.
validate_inputs	Validates the input based on a few simple checks.

evalml.objectives.MulticlassClassificationObjective.__init__

MulticlassClassificationObjective.**__init__**()

Initialize self. See help(type(self)) for accurate signature.

evalml.objectives.MulticlassClassificationObjective.calculate_percent_difference

classmethod MulticlassClassificationObjective.**calculate_percent_difference**(*score*, *baseline_score*)

Calculate the percent difference between scores.

Parameters

- **score** (*float*) – A score. Output of the score method of this objective.
- **baseline_score** (*float*) – A score. Output of the score method of this objective. In practice, this is the score achieved on this objective with a baseline estimator.

Returns

The percent difference between the scores. Note that for objectives that can be interpreted as percentages, this will be the difference between the reference score and score. For all other objectives, the difference will be normalized by the reference score.

Return type float

evalml.objectives.MulticlassClassificationObjective.is_defined_for_problem_type

classmethod MulticlassClassificationObjective.**is_defined_for_problem_type**(*problem_type*)

evalml.objectives.MulticlassClassificationObjective.objective_function

abstract classmethod MulticlassClassificationObjective.**objective_function**(*y_true*, *y_predicted*, *X=None*, *sample_weight=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.ndarray): Extra data of shape [n_samples, n_features] necessary to calculate score sample_weight (pd.DataFrame or np.ndarray): Sample weights used in computing objective value result

Returns Numerical value used to calculate score

evalml.objectives.MulticlassClassificationObjective.score

MulticlassClassificationObjective.**score**(*y_true*, *y_predicted*, *X=None*, *sample_weight=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.ndarray*) – Extra data of shape [n_samples, n_features] necessary to calculate score
- **sample_weight** (*pd.DataFrame or np.ndarray*) – Sample weights used in computing objective value result

Returns score

evalml.objectives.MulticlassClassificationObjective.validate_inputs

MulticlassClassificationObjective.**validate_inputs**(*y_true*, *y_predicted*)

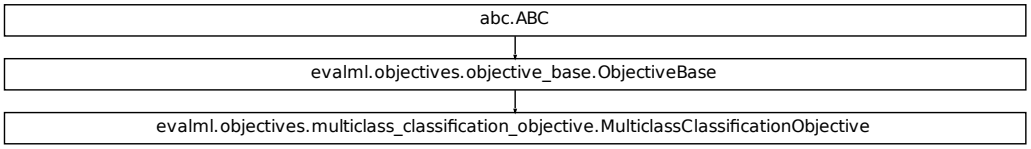
Validates the input based on a few simple checks.

Parameters

- **y_predicted** (*pd.Series, or pd.DataFrame*) – Predicted values of length [n_samples]
- **y_true** (*pd.Series*) – Actual class labels of length [n_samples]

Returns None

Class Inheritance



evalml.objectives.RegressionObjective

class evalml.objectives.**RegressionObjective**

Base class for all regression objectives.

problem_types = [<ProblemTypes.REGRESSION: 'regression'>, <ProblemTypes.TIME_SERIES_RE

Methods

__init__	Initialize self.
calculate_percent_difference	Calculate the percent difference between scores.
is_defined_for_problem_type	
objective_function	Computes the relative value of the provided predictions compared to the actual labels, according a specified metric
score	Returns a numerical score indicating performance based on the differences between the predicted and actual values.
validate_inputs	Validates the input based on a few simple checks.

evalml.objectives.RegressionObjective.__init__

RegressionObjective.**__init__**()

Initialize self. See help(type(self)) for accurate signature.

evalml.objectives.RegressionObjective.calculate_percent_difference

classmethod RegressionObjective.**calculate_percent_difference**(*score*, *baseline_score*)

Calculate the percent difference between scores.

Parameters

- **score** (*float*) – A score. Output of the score method of this objective.
- **baseline_score** (*float*) – A score. Output of the score method of this objective. In practice, this is the score achieved on this objective with a baseline estimator.

Returns

The percent difference between the scores. Note that for objectives that can be interpreted as percentages, this will be the difference between the reference score and score. For all other objectives, the difference will be normalized by the reference score.

Return type float

evalml.objectives.RegressionObjective.is_defined_for_problem_type

classmethod RegressionObjective.**is_defined_for_problem_type**(*problem_type*)

evalml.objectives.RegressionObjective.objective_function

abstract classmethod RegressionObjective.**objective_function**(*y_true*, *y_predicted*, *X=None*, *sample_weight=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.ndarray): Extra data of shape [n_samples, n_features] necessary to calculate score sample_weight (pd.DataFrame or np.ndarray): Sample weights used in computing objective value result

Returns Numerical value used to calculate score

evalml.objectives.RegressionObjective.score

RegressionObjective.**score**(*y_true*, *y_predicted*, *X=None*, *sample_weight=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.ndarray*) – Extra data of shape [n_samples, n_features] necessary to calculate score

- **sample_weight** (*pd.DataFrame or np.ndarray*) – Sample weights used in computing objective value result

Returns score

evalml.objectives.RegressionObjective.validate_inputs

RegressionObjective.**validate_inputs**(*y_true*, *y_predicted*)

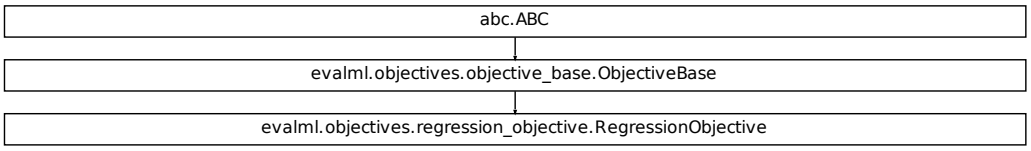
Validates the input based on a few simple checks.

Parameters

- **y_predicted** (*pd.Series, or pd.DataFrame*) – Predicted values of length [n_samples]
- **y_true** (*pd.Series*) – Actual class labels of length [n_samples]

Returns None

Class Inheritance



5.8.2 Domain-Specific Objectives

FraudCost	Score the percentage of money lost of the total transaction amount process due to fraud.
LeadScoring	Lead scoring.
CostBenefitMatrix	Score using a cost-benefit matrix.

evalml.objectives.FraudCost

class evalml.objectives.**FraudCost**(*retry_percentage=0.5*, *interchange_fee=0.02*, *fraud_payout_percentage=1.0*, *amount_col='amount'*)

Score the percentage of money lost of the total transaction amount process due to fraud.

name = 'Fraud Cost'

greater_is_better = False

perfect_score = 0.0

positive_only = False

problem_types = [<ProblemTypes.BINARY: 'binary'>, <ProblemTypes.TIME_SERIES_BINARY: 't

score_needs_proba = False

Methods

__init__	Create instance of FraudCost
calculate_percent_difference	Calculate the percent difference between scores.
decision_function	Determine if a transaction is fraud given predicted probabilities, threshold, and dataframe with transaction amount.
is_defined_for_problem_type	
objective_function	Calculate amount lost to fraud per transaction given predictions, true values, and dataframe with transaction amount.
optimize_threshold	Learn a binary classification threshold which optimizes the current objective.
score	Returns a numerical score indicating performance based on the differences between the predicted and actual values.
validate_inputs	Validates the input based on a few simple checks.

evalml.objectives.FraudCost.__init__

FraudCost.**__init__**(*retry_percentage=0.5*, *interchange_fee=0.02*, *fraud_payout_percentage=1.0*, *amount_col='amount'*)

Create instance of FraudCost

Parameters

- **retry_percentage** (*float*) – What percentage of customers that will retry a transaction if it is declined. Between 0 and 1. Defaults to .5
- **interchange_fee** (*float*) – How much of each successful transaction you can collect. Between 0 and 1. Defaults to .02
- **fraud_payout_percentage** (*float*) – Percentage of fraud you will not be able to collect. Between 0 and 1. Defaults to 1.0
- **amount_col** (*str*) – Name of column in data that contains the amount. Defaults to "amount"

evalml.objectives.FraudCost.calculate_percent_difference

classmethod FraudCost.**calculate_percent_difference**(*score*, *baseline_score*)

Calculate the percent difference between scores.

Parameters

- **score** (*float*) – A score. Output of the score method of this objective.
- **baseline_score** (*float*) – A score. Output of the score method of this objective. In practice, this is the score achieved on this objective with a baseline estimator.

Returns

The percent difference between the scores. Note that for objectives that can be interpreted as percentages, this will be the difference between the reference score and score. For all other objectives, the difference will be normalized by the reference score.

Return type float

evalml.objectives.FraudCost.decision_function

FraudCost.**decision_function**(*ypred_proba*, *threshold=0.0*, *X=None*)

Determine if a transaction is fraud given predicted probabilities, threshold, and dataframe with transaction amount.

Parameters

- **ypred_proba** (*pd.Series*) – Predicted probablities
- **threshold** (*float*) – Dollar threshold to determine if transaction is fraud
- **X** (*pd.DataFrame*) – Data containing transaction amounts

Returns pd.Series of predicted fraud labels using X and threshold

Return type pd.Series

evalml.objectives.FraudCost.is_defined_for_problem_type

classmethod FraudCost.**is_defined_for_problem_type**(*problem_type*)

evalml.objectives.FraudCost.objective_function

FraudCost.**objective_function**(*y_true*, *y_predicted*, *X*, *sample_weight=None*)

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*) – Data with transaction amounts
- **sample_weight** (*pd.DataFrame*) – Ignored.

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** (*pd.Series*) – The classifier's predicted probabilities
- **y_true** (*pd.Series*) – 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*, *sample_weight=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.ndarray*) – Extra data of shape [n_samples, n_features] necessary to calculate score
- **sample_weight** (*pd.DataFrame or np.ndarray*) – Sample weights used in computing objective value result

Returns score

evalml.objectives.FraudCost.validate_inputs

FraudCost.**validate_inputs**(*y_true*, *y_predicted*)

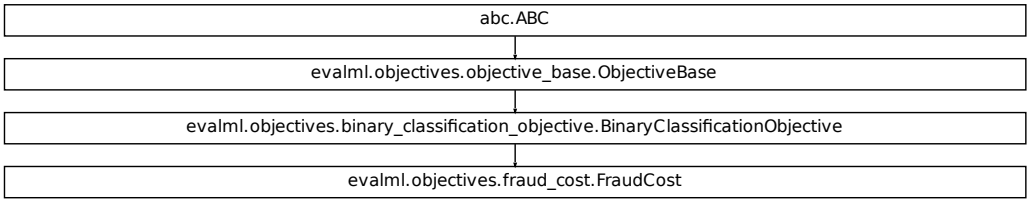
Validates the input based on a few simple checks.

Parameters

- **y_predicted** (*pd.Series, or pd.DataFrame*) – Predicted values of length [n_samples]
- **y_true** (*pd.Series*) – Actual class labels of length [n_samples]

Returns None

Class Inheritance



evalml.objectives.LeadScoring

class evalml.objectives.**LeadScoring**(*true_positives=1*, *false_positives=- 1*)

Lead scoring.

name = 'Lead Scoring'

greater_is_better = True

perfect_score = inf

positive_only = False

problem_types = [<ProblemTypes.BINARY: 'binary'>, <ProblemTypes.TIME_SERIES_BINARY: 't

score_needs_proba = False

Methods

__init__	Create instance.
calculate_percent_difference	Calculate the percent difference between scores.
decision_function	Apply a learned threshold to predicted probabilities to get predicted classes.
is_defined_for_problem_type	
objective_function	Calculate the profit per lead.
optimize_threshold	Learn a binary classification threshold which optimizes the current objective.
score	Returns a numerical score indicating performance based on the differences between the predicted and actual values.
validate_inputs	Validates the input based on a few simple checks.

evalml.objectives.LeadScoring.__init__

LeadScoring.**__init__**(*true_positives=1*, *false_positives=- 1*)

Create instance.

Parameters

- **true_positives** (*int*) – Reward for a true positive
- **false_positives** (*int*) – Cost for a false positive. Should be negative.

evalml.objectives.LeadScoring.calculate_percent_difference

classmethod LeadScoring.**calculate_percent_difference**(*score*, *baseline_score*)

Calculate the percent difference between scores.

Parameters

- **score** (*float*) – A score. Output of the score method of this objective.
- **baseline_score** (*float*) – A score. Output of the score method of this objective. In practice, this is the score achieved on this objective with a baseline estimator.

Returns

The percent difference between the scores. Note that for objectives that can be interpreted as percentages, this will be the difference between the reference score and score. For all other objectives, the difference will be normalized by the reference score.

Return type float

evalml.objectives.LeadScoring.decision_function

LeadScoring.**decision_function**(*ypred_proba*, *threshold=0.5*, *X=None*)

Apply a learned threshold to predicted probabilities to get predicted classes.

Parameters

- **ypred_proba** (*pd.Series, np.ndarray*) – 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.is_defined_for_problem_type

classmethod LeadScoring.**is_defined_for_problem_type**(*problem_type*)

evalml.objectives.LeadScoring.objective_function

LeadScoring.**objective_function**(*y_true*, *y_predicted*, *X=None*, *sample_weight=None*)

Calculate the profit per lead.

Parameters

- **y_predicted** (*pd.Series*) – Predicted labels
- **y_true** (*pd.Series*) – True labels
- **X** (*pd.DataFrame*) – Ignored.
- **sample_weight** (*pd.DataFrame*) – Ignored.

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** (*pd.Series*) – The classifier's predicted probabilities
- **y_true** (*pd.Series*) – 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*, *sample_weight=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.ndarray*) – Extra data of shape [n_samples, n_features] necessary to calculate score
- **sample_weight** (*pd.DataFrame or np.ndarray*) – Sample weights used in computing objective value result

Returns score

evalml.objectives.LeadScoring.validate_inputs

LeadScoring.**validate_inputs**(*y_true*, *y_predicted*)

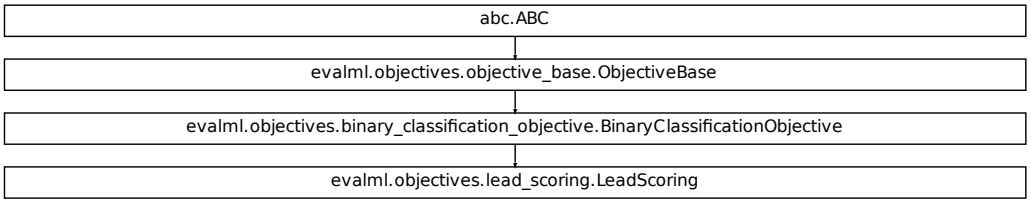
Validates the input based on a few simple checks.

Parameters

- **y_predicted** (*pd.Series, or pd.DataFrame*) – Predicted values of length [n_samples]
- **y_true** (*pd.Series*) – Actual class labels of length [n_samples]

Returns None

Class Inheritance



evalml.objectives.CostBenefitMatrix

class evalml.objectives.**CostBenefitMatrix**(*true_positive*, *true_negative*, *false_positive*, *false_negative*)

Score using a cost-benefit matrix. Scores quantify the benefits of a given value, so greater numeric scores represents a better score. Costs and scores can be negative, indicating that a value is not beneficial. For example, in the case of monetary profit, a negative cost and/or score represents loss of cash flow.

name = 'Cost Benefit Matrix'

greater_is_better = True

perfect_score = inf

positive_only = False

problem_types = [<ProblemTypes.BINARY: 'binary'>, <ProblemTypes.TIME_SERIES_BINARY: 't

score_needs_proba = False

Methods

__init__	Create instance of CostBenefitMatrix.
calculate_percent_difference	Calculate the percent difference between scores.
decision_function	Apply a learned threshold to predicted probabilities to get predicted classes.
is_defined_for_problem_type	
objective_function	Calculates cost-benefit of the using the predicted and true values.
optimize_threshold	Learn a binary classification threshold which optimizes the current objective.
score	Returns a numerical score indicating performance based on the differences between the predicted and actual values.
validate_inputs	Validates the input based on a few simple checks.

evalml.objectives.CostBenefitMatrix.__init__

CostBenefitMatrix.**__init__**(*true_positive*, *true_negative*, *false_positive*, *false_negative*)

Create instance of CostBenefitMatrix.

Parameters

- **true_positive** (*float*) – Cost associated with true positive predictions
- **true_negative** (*float*) – Cost associated with true negative predictions
- **false_positive** (*float*) – Cost associated with false positive predictions
- **false_negative** (*float*) – Cost associated with false negative predictions

evalml.objectives.CostBenefitMatrix.calculate_percent_difference

classmethod CostBenefitMatrix.**calculate_percent_difference**(*score*, *baseline_score*)

Calculate the percent difference between scores.

Parameters

- **score** (*float*) – A score. Output of the score method of this objective.
- **baseline_score** (*float*) – A score. Output of the score method of this objective. In practice, this is the score achieved on this objective with a baseline estimator.

Returns

The percent difference between the scores. Note that for objectives that can be interpreted as percentages, this will be the difference between the reference score and score. For all other objectives, the difference will be normalized by the reference score.

Return type float

evalml.objectives.CostBenefitMatrix.decision_function

CostBenefitMatrix.**decision_function**(*ypred_proba*, *threshold=0.5*, *X=None*)

Apply a learned threshold to predicted probabilities to get predicted classes.

Parameters

- **ypred_proba** (*pd.Series, np.ndarray*) – The classifier's predicted probabilities
- **threshold** (*float, optional*) – Threshold used to make a prediction. Defaults to 0.5.
- **X** (*pd.DataFrame, optional*) – Any extra columns that are needed from training data.

Returns predictions

evalml.objectives.CostBenefitMatrix.is_defined_for_problem_type

classmethod CostBenefitMatrix.**is_defined_for_problem_type**(*problem_type*)

evalml.objectives.CostBenefitMatrix.objective_function

CostBenefitMatrix.**objective_function**(*y_true*, *y_predicted*, *X=None*, *sample_weight=None*)

Calculates cost-benefit of the using the predicted and true values.

Parameters

- **y_predicted** (*pd.Series*) – Predicted labels
- **y_true** (*pd.Series*) – True labels
- **X** (*pd.DataFrame*) – Ignored.
- **sample_weight** (*pd.DataFrame*) – Ignored.

Returns Cost-benefit matrix score

Return type float

evalml.objectives.CostBenefitMatrix.optimize_threshold

CostBenefitMatrix.**optimize_threshold**(*ypred_proba*, *y_true*, *X=None*)

Learn a binary classification threshold which optimizes the current objective.

Parameters

- **ypred_proba** (*pd.Series*) – The classifier's predicted probabilities
- **y_true** (*pd.Series*) – The ground truth for the predictions.
- **X** (*pd.DataFrame, optional*) – Any extra columns that are needed from training data.

Returns Optimal threshold for this objective

evalml.objectives.CostBenefitMatrix.score

CostBenefitMatrix.**score**(*y_true*, *y_predicted*, *X=None*, *sample_weight=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.ndarray*) – Extra data of shape [n_samples, n_features] necessary to calculate score
- **sample_weight** (*pd.DataFrame or np.ndarray*) – Sample weights used in computing objective value result

Returns score

evalml.objectives.CostBenefitMatrix.validate_inputs

CostBenefitMatrix.**validate_inputs**(*y_true*, *y_predicted*)

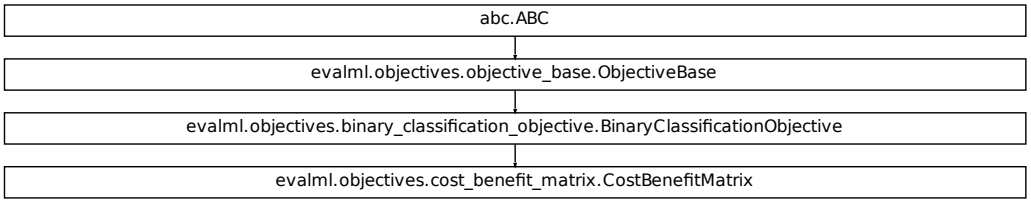
Validates the input based on a few simple checks.

Parameters

- **y_predicted** (*pd.Series, or pd.DataFrame*) – Predicted values of length [n_samples]
- **y_true** (*pd.Series*) – Actual class labels of length [n_samples]

Returns None

Class Inheritance



5.8.3 Classification Objectives

AccuracyBinary	Accuracy score for binary classification.
AccuracyMulticlass	Accuracy score for multiclass classification.
AUC	AUC score for binary classification.
AUCMacro	AUC score for multiclass classification using macro averaging.
AUCMicro	AUC score for multiclass classification using micro averaging.
AUCWeighted	AUC Score for multiclass classification using weighted averaging.
BalancedAccuracyBinary	Balanced accuracy score for binary classification.
BalancedAccuracyMulticlass	Balanced accuracy score for multiclass classification.
F1	F1 score for binary classification.
F1Micro	F1 score for multiclass classification using micro averaging.
F1Macro	F1 score for multiclass classification using macro averaging.
F1Weighted	F1 score for multiclass classification using weighted averaging.
LogLossBinary	Log Loss for binary classification.
LogLossMulticlass	Log Loss for multiclass classification.
MCCBinary	Matthews correlation coefficient for binary classification.
MCCMulticlass	Matthews correlation coefficient for multiclass classification.
Precision	Precision score for binary classification.
PrecisionMicro	Precision score for multiclass classification using micro averaging.
PrecisionMacro	Precision score for multiclass classification using macro averaging.
PrecisionWeighted	Precision score for multiclass classification using weighted averaging.
Recall	Recall score for binary classification.
RecallMicro	Recall score for multiclass classification using micro averaging.
RecallMacro	Recall score for multiclass classification using macro averaging.
RecallWeighted	Recall score for multiclass classification using weighted averaging.

evalml.objectives.AccuracyBinary

class evalml.objectives.**AccuracyBinary**

Accuracy score for binary classification.

name = 'Accuracy Binary'

greater_is_better = True

perfect_score = 1.0

positive_only = False

problem_types = [<ProblemTypes.BINARY: 'binary'>, <ProblemTypes.TIME_SERIES_BINARY: 't

score_needs_proba = False

Methods

__init__	Initialize self.
calculate_percent_difference	Calculate the percent difference between scores.
decision_function	Apply a learned threshold to predicted probabilities to get predicted classes.
is_defined_for_problem_type	
objective_function	Computes the relative value of the provided predictions compared to the actual labels, according a specified metric
optimize_threshold	Learn a binary classification threshold which optimizes the current objective.
score	Returns a numerical score indicating performance based on the differences between the predicted and actual values.
validate_inputs	Validates the input based on a few simple checks.

evalml.objectives.AccuracyBinary.__init__

AccuracyBinary.**__init__**()
Initialize self. See help(type(self)) for accurate signature.

evalml.objectives.AccuracyBinary.calculate_percent_difference

classmethod AccuracyBinary.**calculate_percent_difference**(*score*, *baseline_score*)
Calculate the percent difference between scores.

Parameters

- **score** (*float*) – A score. Output of the score method of this objective.
- **baseline_score** (*float*) – A score. Output of the score method of this objective. In practice, this is the score achieved on this objective with a baseline estimator.

Returns

The percent difference between the scores. Note that for objectives that can be interpreted as percentages, this will be the difference between the reference score and score. For all other objectives, the difference will be normalized by the reference score.

Return type float

evalml.objectives.AccuracyBinary.decision_function

AccuracyBinary.**decision_function**(*ypred_proba*, *threshold=0.5*, *X=None*)

Apply a learned threshold to predicted probabilities to get predicted classes.

Parameters

- **ypred_proba** (*pd.Series, np.ndarray*) – 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.is_defined_for_problem_type

classmethod AccuracyBinary.**is_defined_for_problem_type**(*problem_type*)

evalml.objectives.AccuracyBinary.objective_function

AccuracyBinary.**objective_function**(*y_true*, *y_predicted*, *X=None*, *sample_weight=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.ndarray): Extra data of shape [n_samples, n_features] necessary to calculate score sample_weight (pd.DataFrame or np.ndarray): Sample weights used in computing objective value result

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** (*pd.Series*) – The classifier's predicted probabilities
- **y_true** (*pd.Series*) – 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*, *sample_weight=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.ndarray*) – Extra data of shape [n_samples, n_features] necessary to calculate score
- **sample_weight** (*pd.DataFrame or np.ndarray*) – Sample weights used in computing objective value result

Returns score

evalml.objectives.AccuracyBinary.validate_inputs

AccuracyBinary.**validate_inputs**(*y_true*, *y_predicted*)

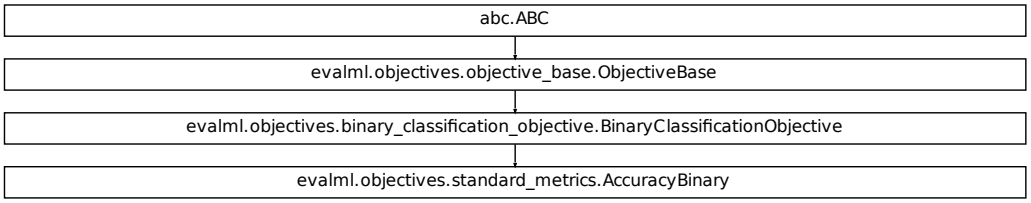
Validates the input based on a few simple checks.

Parameters

- **y_predicted** (*pd.Series, or pd.DataFrame*) – Predicted values of length [n_samples]
- **y_true** (*pd.Series*) – Actual class labels of length [n_samples]

Returns None

Class Inheritance



evalml.objectives.AccuracyMulticlass

class evalml.objectives.**AccuracyMulticlass**

Accuracy score for multiclass classification.

name = 'Accuracy Multiclass'

greater_is_better = True

perfect_score = 1.0

positive_only = False

problem_types = [<ProblemTypes.MULTICLASS: 'multiclass'>, <ProblemTypes.TIME_SERIES_MU

score_needs_proba = False

Methods

__init__	Initialize self.
calculate_percent_difference	Calculate the percent difference between scores.
is_defined_for_problem_type	
objective_function	Computes the relative value of the provided predictions compared to the actual labels, according a specified metric
score	Returns a numerical score indicating performance based on the differences between the predicted and actual values.
validate_inputs	Validates the input based on a few simple checks.

evalml.objectives.AccuracyMulticlass.__init__

AccuracyMulticlass.**__init__**()

Initialize self. See help(type(self)) for accurate signature.

evalml.objectives.AccuracyMulticlass.calculate_percent_difference

classmethod AccuracyMulticlass.**calculate_percent_difference**(*score*, *baseline_score*)

Calculate the percent difference between scores.

Parameters

- **score** (*float*) – A score. Output of the score method of this objective.
- **baseline_score** (*float*) – A score. Output of the score method of this objective. In practice, this is the score achieved on this objective with a baseline estimator.

Returns

The percent difference between the scores. Note that for objectives that can be interpreted as percentages, this will be the difference between the reference score and score. For all other objectives, the difference will be normalized by the reference score.

Return type float

evalml.objectives.AccuracyMulticlass.is_defined_for_problem_type

classmethod AccuracyMulticlass.**is_defined_for_problem_type**(*problem_type*)

evalml.objectives.AccuracyMulticlass.objective_function

AccuracyMulticlass.**objective_function**(*y_true*, *y_predicted*, *X=None*, *sample_weight=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.ndarray): Extra data of shape [n_samples, n_features] necessary to calculate score sample_weight (pd.DataFrame or np.ndarray): Sample weights used in computing objective value result

Returns Numerical value used to calculate score

evalml.objectives.AccuracyMulticlass.score

AccuracyMulticlass.**score**(*y_true*, *y_predicted*, *X=None*, *sample_weight=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.ndarray*) – Extra data of shape [n_samples, n_features] necessary to calculate score
- **sample_weight** (*pd.DataFrame or np.ndarray*) – Sample weights used in computing objective value result

Returns score

evalml.objectives.AccuracyMulticlass.validate_inputs

AccuracyMulticlass.**validate_inputs**(*y_true*, *y_predicted*)

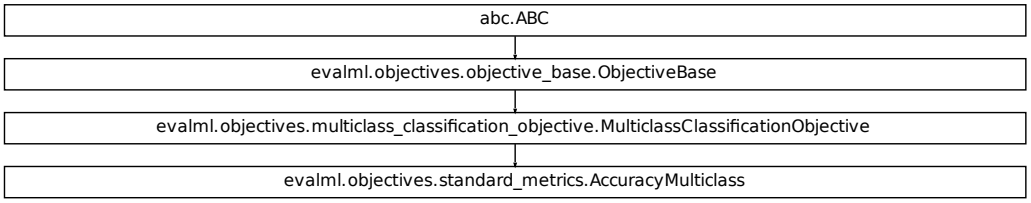
Validates the input based on a few simple checks.

Parameters

- **y_predicted** (*pd.Series, or pd.DataFrame*) – Predicted values of length [n_samples]
- **y_true** (*pd.Series*) – Actual class labels of length [n_samples]

Returns None

Class Inheritance



evalml.objectives.AUC

class evalml.objectives.**AUC**

AUC score for binary classification.

name = 'AUC'

greater_is_better = True

perfect_score = 1.0

positive_only = False

problem_types = [<ProblemTypes.BINARY: 'binary'>, <ProblemTypes.TIME_SERIES_BINARY: 't

score_needs_proba = True

Methods

__init__	Initialize self.
calculate_percent_difference	Calculate the percent difference between scores.
decision_function	Apply a learned threshold to predicted probabilities to get predicted classes.
is_defined_for_problem_type	
objective_function	Computes the relative value of the provided predictions compared to the actual labels, according a specified metric
optimize_threshold	Learn a binary classification threshold which optimizes the current objective.
score	Returns a numerical score indicating performance based on the differences between the predicted and actual values.
validate_inputs	Validates the input based on a few simple checks.

evalml.objectives.AUC.__init__

AUC.**__init__**()
Initialize self. See help(type(self)) for accurate signature.

evalml.objectives.AUC.calculate_percent_difference

classmethod AUC.**calculate_percent_difference**(*score*, *baseline_score*)
Calculate the percent difference between scores.

Parameters

- **score** (*float*) – A score. Output of the score method of this objective.
- **baseline_score** (*float*) – A score. Output of the score method of this objective. In practice, this is the score achieved on this objective with a baseline estimator.

Returns

The percent difference between the scores. Note that for objectives that can be interpreted as percentages, this will be the difference between the reference score and score. For all other objectives, the difference will be normalized by the reference score.

Return type float

evalml.objectives.AUC.decision_function

AUC.**decision_function**(*ypred_proba*, *threshold=0.5*, *X=None*)
Apply a learned threshold to predicted probabilities to get predicted classes.

Parameters

- **ypred_proba** (*pd.Series, np.ndarray*) – 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.is_defined_for_problem_type

classmethod AUC.**is_defined_for_problem_type**(*problem_type*)

evalml.objectives.AUC.objective_function

AUC.**objective_function**(*y_true*, *y_predicted*, *X=None*, *sample_weight=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.ndarray): Extra data of shape [n_samples, n_features] necessary to calculate score sample_weight (pd.DataFrame or np.ndarray): Sample weights used in computing objective value result

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** (*pd.Series*) – The classifier's predicted probabilities
- **y_true** (*pd.Series*) – 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*, *sample_weight=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.ndarray*) – Extra data of shape [n_samples, n_features] necessary to calculate score
- **sample_weight** (*pd.DataFrame or np.ndarray*) – Sample weights used in computing objective value result

Returns score

evalml.objectives.AUC.validate_inputs

AUC.**validate_inputs**(*y_true*, *y_predicted*)

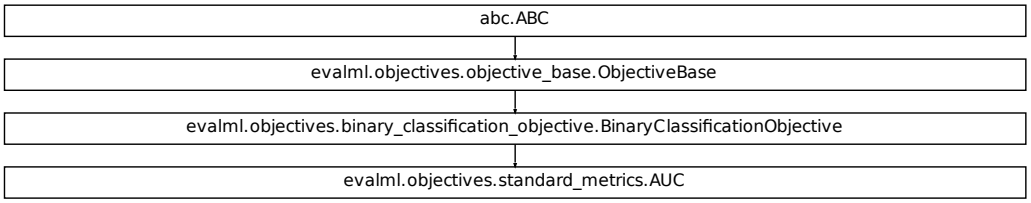
Validates the input based on a few simple checks.

Parameters

- **y_predicted** (*pd.Series, or pd.DataFrame*) – Predicted values of length [n_samples]
- **y_true** (*pd.Series*) – Actual class labels of length [n_samples]

Returns None

Class Inheritance



evalml.objectives.AUCMacro

class evalml.objectives.**AUCMacro**

AUC score for multiclass classification using macro averaging.

name = 'AUC Macro'

greater_is_better = True

perfect_score = 1.0

positive_only = False

problem_types = [<ProblemTypes.MULTICLASS: 'multiclass'>, <ProblemTypes.TIME_SERIES_MU

score_needs_proba = True

Methods

__init__	Initialize self.
calculate_percent_difference	Calculate the percent difference between scores.
is_defined_for_problem_type	
objective_function	Computes the relative value of the provided predictions compared to the actual labels, according a specified metric
score	Returns a numerical score indicating performance based on the differences between the predicted and actual values.
validate_inputs	Validates the input based on a few simple checks.

evalml.objectives.AUCMacro.__init__

AUCMacro.**__init__**()
Initialize self. See help(type(self)) for accurate signature.

evalml.objectives.AUCMacro.calculate_percent_difference

classmethod AUCMacro.**calculate_percent_difference**(*score*, *baseline_score*)
Calculate the percent difference between scores.

Parameters

- **score** (*float*) – A score. Output of the score method of this objective.
- **baseline_score** (*float*) – A score. Output of the score method of this objective. In practice, this is the score achieved on this objective with a baseline estimator.

Returns

The percent difference between the scores. Note that for objectives that can be interpreted as percentages, this will be the difference between the reference score and score. For all other objectives, the difference will be normalized by the reference score.

Return type float

evalml.objectives.AUCMacro.is_defined_for_problem_type

classmethod AUCMacro.**is_defined_for_problem_type**(*problem_type*)

evalml.objectives.AUCMacro.objective_function

AUCMacro.**objective_function**(*y_true*, *y_predicted*, *X=None*, *sample_weight=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.ndarray): Extra data of shape [n_samples, n_features] necessary to calculate score sample_weight (pd.DataFrame or np.ndarray): Sample weights used in computing objective value result

Returns Numerical value used to calculate score

evalml.objectives.AUCMacro.score

AUCMacro.**score**(*y_true*, *y_predicted*, *X=None*, *sample_weight=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.ndarray*) – Extra data of shape [n_samples, n_features] necessary to calculate score
- **sample_weight** (*pd.DataFrame or np.ndarray*) – Sample weights used in computing objective value result

Returns score

evalml.objectives.AUCMacro.validate_inputs

AUCMacro.**validate_inputs**(*y_true*, *y_predicted*)

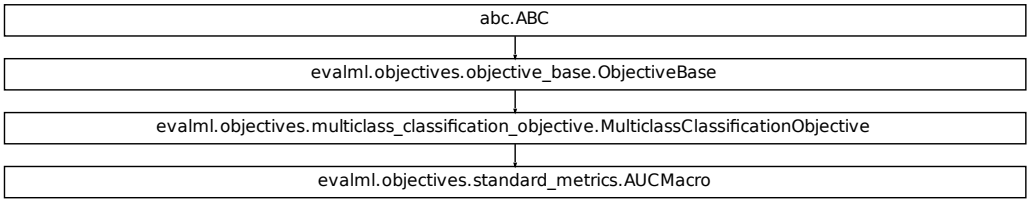
Validates the input based on a few simple checks.

Parameters

- **y_predicted** (*pd.Series, or pd.DataFrame*) – Predicted values of length [n_samples]
- **y_true** (*pd.Series*) – Actual class labels of length [n_samples]

Returns None

Class Inheritance



evalml.objectives.AUCMicro

class evalml.objectives.**AUCMicro**

AUC score for multiclass classification using micro averaging.

name = 'AUC Micro'

greater_is_better = True

perfect_score = 1.0

positive_only = False

problem_types = [<ProblemTypes.MULTICLASS: 'multiclass'>, <ProblemTypes.TIME_SERIES_MU

score_needs_proba = True

Methods

__init__	Initialize self.
calculate_percent_difference	Calculate the percent difference between scores.
is_defined_for_problem_type	
objective_function	Computes the relative value of the provided predictions compared to the actual labels, according a specified metric
score	Returns a numerical score indicating performance based on the differences between the predicted and actual values.
validate_inputs	Validates the input based on a few simple checks.

evalml.objectives.AUCMicro.__init__

AUCMicro.**__init__**()
Initialize self. See help(type(self)) for accurate signature.

evalml.objectives.AUCMicro.calculate_percent_difference

classmethod AUCMicro.**calculate_percent_difference**(*score*, *baseline_score*)
Calculate the percent difference between scores.

Parameters

- **score** (*float*) – A score. Output of the score method of this objective.
- **baseline_score** (*float*) – A score. Output of the score method of this objective. In practice, this is the score achieved on this objective with a baseline estimator.

Returns

The percent difference between the scores. Note that for objectives that can be interpreted as percentages, this will be the difference between the reference score and score. For all other objectives, the difference will be normalized by the reference score.

Return type float

evalml.objectives.AUCMicro.is_defined_for_problem_type

classmethod AUCMicro.**is_defined_for_problem_type**(*problem_type*)

evalml.objectives.AUCMicro.objective_function

AUCMicro.**objective_function**(*y_true*, *y_predicted*, *X=None*, *sample_weight=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.ndarray): Extra data of shape [n_samples, n_features] necessary to calculate score sample_weight (pd.DataFrame or np.ndarray): Sample weights used in computing objective value result

Returns Numerical value used to calculate score

evalml.objectives.AUCMicro.score

AUCMicro.**score**(*y_true*, *y_predicted*, *X=None*, *sample_weight=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.ndarray*) – Extra data of shape [n_samples, n_features] necessary to calculate score
- **sample_weight** (*pd.DataFrame or np.ndarray*) – Sample weights used in computing objective value result

Returns score

evalml.objectives.AUCMicro.validate_inputs

AUCMicro.**validate_inputs**(*y_true*, *y_predicted*)

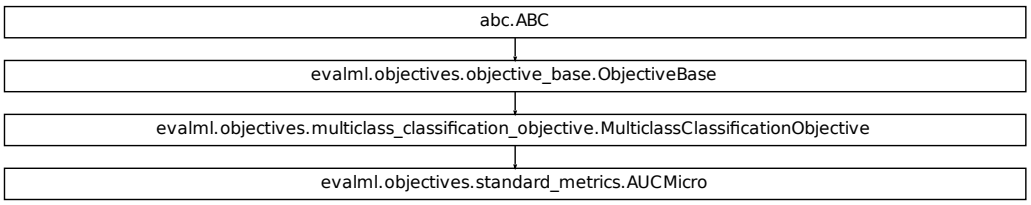
Validates the input based on a few simple checks.

Parameters

- **y_predicted** (*pd.Series, or pd.DataFrame*) – Predicted values of length [n_samples]
- **y_true** (*pd.Series*) – Actual class labels of length [n_samples]

Returns None

Class Inheritance



evalml.objectives.AUCWeighted

class evalml.objectives.**AUCWeighted**

AUC Score for multiclass classification using weighted averaging.

name = 'AUC Weighted'

greater_is_better = True

perfect_score = 1.0

positive_only = False

problem_types = [<ProblemTypes.MULTICLASS: 'multiclass'>, <ProblemTypes.TIME_SERIES_MU

score_needs_proba = True

Methods

__init__	Initialize self.
calculate_percent_difference	Calculate the percent difference between scores.
is_defined_for_problem_type	
objective_function	Computes the relative value of the provided predictions compared to the actual labels, according a specified metric
score	Returns a numerical score indicating performance based on the differences between the predicted and actual values.
validate_inputs	Validates the input based on a few simple checks.

evalml.objectives.AUCWeighted.__init__

AUCWeighted.**__init__**()
Initialize self. See help(type(self)) for accurate signature.

evalml.objectives.AUCWeighted.calculate_percent_difference

classmethod AUCWeighted.**calculate_percent_difference**(*score*, *baseline_score*)
Calculate the percent difference between scores.

Parameters

- **score** (*float*) – A score. Output of the score method of this objective.
- **baseline_score** (*float*) – A score. Output of the score method of this objective. In practice, this is the score achieved on this objective with a baseline estimator.

Returns

The percent difference between the scores. Note that for objectives that can be interpreted as percentages, this will be the difference between the reference score and score. For all other objectives, the difference will be normalized by the reference score.

Return type float

evalml.objectives.AUCWeighted.is_defined_for_problem_type

classmethod AUCWeighted.**is_defined_for_problem_type**(*problem_type*)

evalml.objectives.AUCWeighted.objective_function

AUCWeighted.**objective_function**(*y_true*, *y_predicted*, *X=None*, *sample_weight=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.ndarray): Extra data of shape [n_samples, n_features] necessary to calculate score sample_weight (pd.DataFrame or np.ndarray): Sample weights used in computing objective value result

Returns Numerical value used to calculate score

evalml.objectives.AUCWeighted.score

AUCWeighted.**score**(*y_true*, *y_predicted*, *X=None*, *sample_weight=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.ndarray*) – Extra data of shape [n_samples, n_features] necessary to calculate score
- **sample_weight** (*pd.DataFrame or np.ndarray*) – Sample weights used in computing objective value result

Returns score

evalml.objectives.AUCWeighted.validate_inputs

AUCWeighted.**validate_inputs**(*y_true*, *y_predicted*)

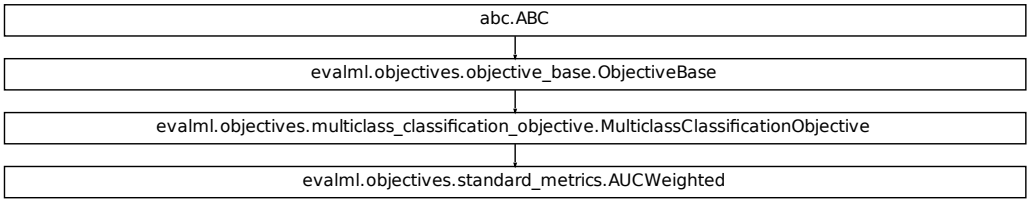
Validates the input based on a few simple checks.

Parameters

- **y_predicted** (*pd.Series, or pd.DataFrame*) – Predicted values of length [n_samples]
- **y_true** (*pd.Series*) – Actual class labels of length [n_samples]

Returns None

Class Inheritance



evalml.objectives.BalancedAccuracyBinary

class evalml.objectives.**BalancedAccuracyBinary**

Balanced accuracy score for binary classification.

name = 'Balanced Accuracy Binary'

greater_is_better = True

perfect_score = 1.0

positive_only = False

problem_types = [<ProblemTypes.BINARY: 'binary'>, <ProblemTypes.TIME_SERIES_BINARY: 't

score_needs_proba = False

Methods

__init__	Initialize self.
calculate_percent_difference	Calculate the percent difference between scores.
decision_function	Apply a learned threshold to predicted probabilities to get predicted classes.
is_defined_for_problem_type	
objective_function	Computes the relative value of the provided predictions compared to the actual labels, according a specified metric
optimize_threshold	Learn a binary classification threshold which optimizes the current objective.
score	Returns a numerical score indicating performance based on the differences between the predicted and actual values.
validate_inputs	Validates the input based on a few simple checks.

evalml.objectives.BalancedAccuracyBinary.__init__

BalancedAccuracyBinary.**__init__**()
Initialize self. See help(type(self)) for accurate signature.

evalml.objectives.BalancedAccuracyBinary.calculate_percent_difference

classmethod BalancedAccuracyBinary.**calculate_percent_difference**(*score*, *baseline_score*)
Calculate the percent difference between scores.

Parameters

- **score** (*float*) – A score. Output of the score method of this objective.
- **baseline_score** (*float*) – A score. Output of the score method of this objective. In practice, this is the score achieved on this objective with a baseline estimator.

Returns

The percent difference between the scores. Note that for objectives that can be interpreted as percentages, this will be the difference between the reference score and score. For all other objectives, the difference will be normalized by the reference score.

Return type float

evalml.objectives.BalancedAccuracyBinary.decision_function

BalancedAccuracyBinary.**decision_function**(*ypred_proba*, *threshold=0.5*, *X=None*)
Apply a learned threshold to predicted probabilities to get predicted classes.

Parameters

- **ypred_proba** (*pd.Series, np.ndarray*) – 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.is_defined_for_problem_type

classmethod BalancedAccuracyBinary.**is_defined_for_problem_type**(*problem_type*)

evalml.objectives.BalancedAccuracyBinary.objective_function

BalancedAccuracyBinary.**objective_function**(*y_true*, *y_predicted*, *X=None*, *sample_weight=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.ndarray): Extra data of shape [n_samples, n_features] necessary to calculate score sample_weight (pd.DataFrame or np.ndarray): Sample weights used in computing objective value result

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** (*pd.Series*) – The classifier's predicted probabilities
- **y_true** (*pd.Series*) – 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*, *sample_weight=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.ndarray*) – Extra data of shape [n_samples, n_features] necessary to calculate score
- **sample_weight** (*pd.DataFrame or np.ndarray*) – Sample weights used in computing objective value result

Returns score

evalml.objectives.BalancedAccuracyBinary.validate_inputs

BalancedAccuracyBinary.**validate_inputs**(*y_true*, *y_predicted*)

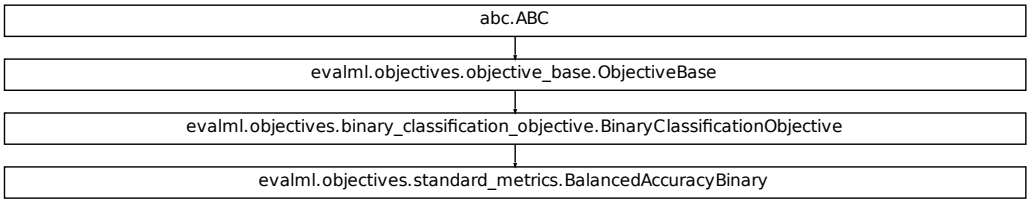
Validates the input based on a few simple checks.

Parameters

- **y_predicted** (*pd.Series, or pd.DataFrame*) – Predicted values of length [n_samples]
- **y_true** (*pd.Series*) – Actual class labels of length [n_samples]

Returns None

Class Inheritance



evalml.objectives.BalancedAccuracyMulticlass

class evalml.objectives.**BalancedAccuracyMulticlass**

Balanced accuracy score for multiclass classification.

name = 'Balanced Accuracy Multiclass'

greater_is_better = True

perfect_score = 1.0

positive_only = False

problem_types = [<ProblemTypes.MULTICLASS: 'multiclass'>, <ProblemTypes.TIME_SERIES_MU

score_needs_proba = False

Methods

__init__	Initialize self.
calculate_percent_difference	Calculate the percent difference between scores.
is_defined_for_problem_type	
objective_function	Computes the relative value of the provided predictions compared to the actual labels, according a specified metric
score	Returns a numerical score indicating performance based on the differences between the predicted and actual values.
validate_inputs	Validates the input based on a few simple checks.

evalml.objectives.BalancedAccuracyMulticlass.__init__

BalancedAccuracyMulticlass.**__init__**()

Initialize self. See help(type(self)) for accurate signature.

evalml.objectives.BalancedAccuracyMulticlass.calculate_percent_difference

classmethod BalancedAccuracyMulticlass.**calculate_percent_difference**(*score*, *baseline_score*)

Calculate the percent difference between scores.

Parameters

- **score** (*float*) – A score. Output of the score method of this objective.
- **baseline_score** (*float*) – A score. Output of the score method of this objective. In practice, this is the score achieved on this objective with a baseline estimator.

Returns

The percent difference between the scores. Note that for objectives that can be interpreted as percentages, this will be the difference between the reference score and score. For all other objectives, the difference will be normalized by the reference score.

Return type float

evalml.objectives.BalancedAccuracyMulticlass.is_defined_for_problem_type

classmethod BalancedAccuracyMulticlass.**is_defined_for_problem_type**(*problem_type*)

evalml.objectives.BalancedAccuracyMulticlass.objective_function

BalancedAccuracyMulticlass.**objective_function**(*y_true*, *y_predicted*, *X=None*, *sample_weight=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.ndarray): Extra data of shape [n_samples, n_features] necessary to calculate score sample_weight (pd.DataFrame or np.ndarray): Sample weights used in computing objective value result

Returns Numerical value used to calculate score

evalml.objectives.BalancedAccuracyMulticlass.score

BalancedAccuracyMulticlass.**score**(*y_true*, *y_predicted*, *X=None*, *sample_weight=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.ndarray*) – Extra data of shape [n_samples, n_features] necessary to calculate score
- **sample_weight** (*pd.DataFrame or np.ndarray*) – Sample weights used in computing objective value result

Returns score

evalml.objectives.BalancedAccuracyMulticlass.validate_inputs

BalancedAccuracyMulticlass.**validate_inputs**(*y_true*, *y_predicted*)

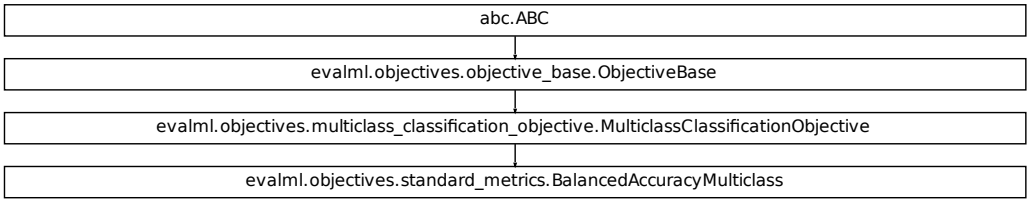
Validates the input based on a few simple checks.

Parameters

- **y_predicted** (*pd.Series, or pd.DataFrame*) – Predicted values of length [n_samples]
- **y_true** (*pd.Series*) – Actual class labels of length [n_samples]

Returns None

Class Inheritance



evalml.objectives.F1

class evalml.objectives.**F1**

F1 score for binary classification.

name = 'F1'

greater_is_better = True

perfect_score = 1.0

positive_only = False

problem_types = [<ProblemTypes.BINARY: 'binary'>, <ProblemTypes.TIME_SERIES_BINARY: 't

score_needs_proba = False

Methods

__init__	Initialize self.
calculate_percent_difference	Calculate the percent difference between scores.
decision_function	Apply a learned threshold to predicted probabilities to get predicted classes.
is_defined_for_problem_type	
objective_function	Computes the relative value of the provided predictions compared to the actual labels, according a specified metric
optimize_threshold	Learn a binary classification threshold which optimizes the current objective.
score	Returns a numerical score indicating performance based on the differences between the predicted and actual values.
validate_inputs	Validates the input based on a few simple checks.

evalml.objectives.F1.__init__

F1.**__init__**()
Initialize self. See help(type(self)) for accurate signature.

evalml.objectives.F1.calculate_percent_difference

classmethod F1.**calculate_percent_difference**(*score*, *baseline_score*)
Calculate the percent difference between scores.

Parameters

- **score** (*float*) – A score. Output of the score method of this objective.
- **baseline_score** (*float*) – A score. Output of the score method of this objective. In practice, this is the score achieved on this objective with a baseline estimator.

Returns

The percent difference between the scores. Note that for objectives that can be interpreted as percentages, this will be the difference between the reference score and score. For all other objectives, the difference will be normalized by the reference score.

Return type float

evalml.objectives.F1.decision_function

F1.**decision_function**(*ypred_proba*, *threshold=0.5*, *X=None*)
Apply a learned threshold to predicted probabilities to get predicted classes.

Parameters

- **ypred_proba** (*pd.Series, np.ndarray*) – 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.is_defined_for_problem_type

classmethod F1.**is_defined_for_problem_type**(*problem_type*)

evalml.objectives.F1.objective_function

F1.**objective_function**(*y_true*, *y_predicted*, *X=None*, *sample_weight=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.ndarray): Extra data of shape [n_samples, n_features] necessary to calculate score sample_weight (pd.DataFrame or np.ndarray): Sample weights used in computing objective value result

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** (*pd.Series*) – The classifier's predicted probabilities
- **y_true** (*pd.Series*) – 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*, *sample_weight=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.ndarray*) – Extra data of shape [n_samples, n_features] necessary to calculate score
- **sample_weight** (*pd.DataFrame or np.ndarray*) – Sample weights used in computing objective value result

Returns score

evalml.objectives.F1.validate_inputs

F1.**validate_inputs**(*y_true*, *y_predicted*)

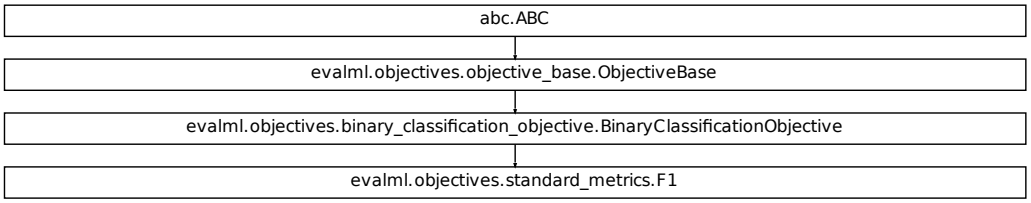
Validates the input based on a few simple checks.

Parameters

- **y_predicted** (*pd.Series, or pd.DataFrame*) – Predicted values of length [n_samples]
- **y_true** (*pd.Series*) – Actual class labels of length [n_samples]

Returns None

Class Inheritance



evalml.objectives.F1Micro

class evalml.objectives.**F1Micro**

F1 score for multiclass classification using micro averaging.

name = 'F1 Micro'

greater_is_better = True

perfect_score = 1.0

positive_only = False

problem_types = [<ProblemTypes.MULTICLASS: 'multiclass'>, <ProblemTypes.TIME_SERIES_MU

score_needs_proba = False

Methods

__init__	Initialize self.
calculate_percent_difference	Calculate the percent difference between scores.
is_defined_for_problem_type	
objective_function	Computes the relative value of the provided predictions compared to the actual labels, according a specified metric
score	Returns a numerical score indicating performance based on the differences between the predicted and actual values.
validate_inputs	Validates the input based on a few simple checks.

evalml.objectives.F1Micro.__init__

F1Micro.**__init__**()
Initialize self. See help(type(self)) for accurate signature.

evalml.objectives.F1Micro.calculate_percent_difference

classmethod F1Micro.**calculate_percent_difference**(*score*, *baseline_score*)
Calculate the percent difference between scores.

Parameters

- **score** (*float*) – A score. Output of the score method of this objective.
- **baseline_score** (*float*) – A score. Output of the score method of this objective. In practice, this is the score achieved on this objective with a baseline estimator.

Returns

The percent difference between the scores. Note that for objectives that can be interpreted as percentages, this will be the difference between the reference score and score. For all other objectives, the difference will be normalized by the reference score.

Return type float

evalml.objectives.F1Micro.is_defined_for_problem_type

classmethod F1Micro.**is_defined_for_problem_type**(*problem_type*)

evalml.objectives.F1Micro.objective_function

F1Micro.**objective_function**(*y_true*, *y_predicted*, *X=None*, *sample_weight=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.ndarray): Extra data of shape [n_samples, n_features] necessary to calculate score sample_weight (pd.DataFrame or np.ndarray): Sample weights used in computing objective value result

Returns Numerical value used to calculate score

evalml.objectives.F1Micro.score

F1Micro.**score**(*y_true*, *y_predicted*, *X=None*, *sample_weight=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.ndarray*) – Extra data of shape [n_samples, n_features] necessary to calculate score
- **sample_weight** (*pd.DataFrame or np.ndarray*) – Sample weights used in computing objective value result

Returns score

evalml.objectives.F1Micro.validate_inputs

F1Micro.**validate_inputs**(*y_true*, *y_predicted*)

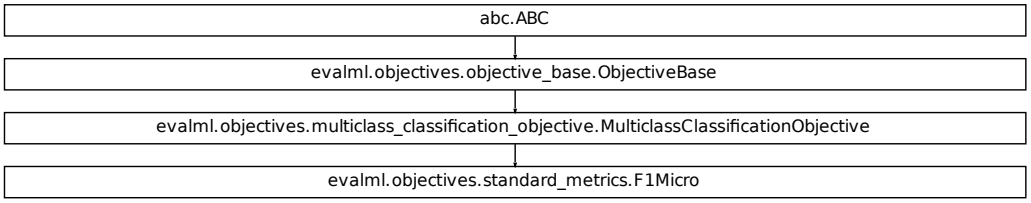
Validates the input based on a few simple checks.

Parameters

- **y_predicted** (*pd.Series, or pd.DataFrame*) – Predicted values of length [n_samples]
- **y_true** (*pd.Series*) – Actual class labels of length [n_samples]

Returns None

Class Inheritance



evalml.objectives.F1Macro

class evalml.objectives.**F1Macro**

F1 score for multiclass classification using macro averaging.

name = 'F1 Macro'

greater_is_better = True

perfect_score = 1.0

positive_only = False

problem_types = [<ProblemTypes.MULTICLASS: 'multiclass'>, <ProblemTypes.TIME_SERIES_MU

score_needs_proba = False

Methods

__init__	Initialize self.
calculate_percent_difference	Calculate the percent difference between scores.
is_defined_for_problem_type	
objective_function	Computes the relative value of the provided predictions compared to the actual labels, according a specified metric
score	Returns a numerical score indicating performance based on the differences between the predicted and actual values.
validate_inputs	Validates the input based on a few simple checks.

evalml.objectives.F1Macro.__init__

F1Macro.**__init__**()
Initialize self. See help(type(self)) for accurate signature.

evalml.objectives.F1Macro.calculate_percent_difference

classmethod F1Macro.**calculate_percent_difference**(*score*, *baseline_score*)
Calculate the percent difference between scores.

Parameters

- **score** (*float*) – A score. Output of the score method of this objective.
- **baseline_score** (*float*) – A score. Output of the score method of this objective. In practice, this is the score achieved on this objective with a baseline estimator.

Returns

The percent difference between the scores. Note that for objectives that can be interpreted as percentages, this will be the difference between the reference score and score. For all other objectives, the difference will be normalized by the reference score.

Return type float

evalml.objectives.F1Macro.is_defined_for_problem_type

classmethod F1Macro.**is_defined_for_problem_type**(*problem_type*)

evalml.objectives.F1Macro.objective_function

F1Macro.**objective_function**(*y_true*, *y_predicted*, *X=None*, *sample_weight=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.ndarray): Extra data of shape [n_samples, n_features] necessary to calculate score sample_weight (pd.DataFrame or np.ndarray): Sample weights used in computing objective value result

Returns Numerical value used to calculate score

evalml.objectives.F1Macro.score

F1Macro.**score**(*y_true*, *y_predicted*, *X=None*, *sample_weight=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.ndarray*) – Extra data of shape [n_samples, n_features] necessary to calculate score
- **sample_weight** (*pd.DataFrame or np.ndarray*) – Sample weights used in computing objective value result

Returns score

evalml.objectives.F1Macro.validate_inputs

F1Macro.**validate_inputs**(*y_true*, *y_predicted*)

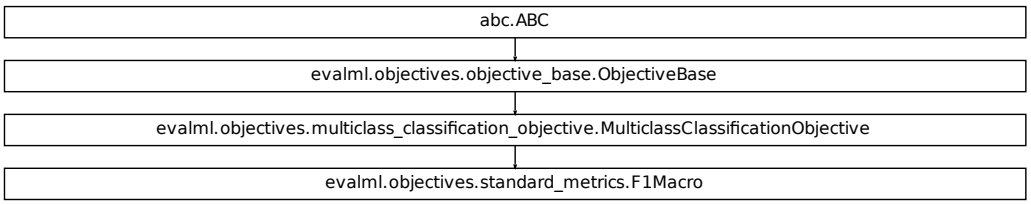
Validates the input based on a few simple checks.

Parameters

- **y_predicted** (*pd.Series, or pd.DataFrame*) – Predicted values of length [n_samples]
- **y_true** (*pd.Series*) – Actual class labels of length [n_samples]

Returns None

Class Inheritance



evalml.objectives.F1Weighted

class evalml.objectives.**F1Weighted**

F1 score for multiclass classification using weighted averaging.

name = 'F1 Weighted'

greater_is_better = True

perfect_score = 1.0

positive_only = False

problem_types = [<ProblemTypes.MULTICLASS: 'multiclass'>, <ProblemTypes.TIME_SERIES_MU

score_needs_proba = False

Methods

__init__	Initialize self.
calculate_percent_difference	Calculate the percent difference between scores.
is_defined_for_problem_type	
objective_function	Computes the relative value of the provided predictions compared to the actual labels, according a specified metric
score	Returns a numerical score indicating performance based on the differences between the predicted and actual values.
validate_inputs	Validates the input based on a few simple checks.

evalml.objectives.F1Weighted.__init__

F1Weighted.**__init__**()
Initialize self. See help(type(self)) for accurate signature.

evalml.objectives.F1Weighted.calculate_percent_difference

classmethod F1Weighted.**calculate_percent_difference**(*score*, *baseline_score*)
Calculate the percent difference between scores.

Parameters

- **score** (*float*) – A score. Output of the score method of this objective.
- **baseline_score** (*float*) – A score. Output of the score method of this objective. In practice, this is the score achieved on this objective with a baseline estimator.

Returns

The percent difference between the scores. Note that for objectives that can be interpreted as percentages, this will be the difference between the reference score and score. For all other objectives, the difference will be normalized by the reference score.

Return type float

evalml.objectives.F1Weighted.is_defined_for_problem_type

classmethod F1Weighted.**is_defined_for_problem_type**(*problem_type*)

evalml.objectives.F1Weighted.objective_function

F1Weighted.**objective_function**(*y_true*, *y_predicted*, *X=None*, *sample_weight=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.ndarray): Extra data of shape [n_samples, n_features] necessary to calculate score sample_weight (pd.DataFrame or np.ndarray): Sample weights used in computing objective value result

Returns Numerical value used to calculate score

evalml.objectives.F1Weighted.score

F1Weighted.**score**(*y_true*, *y_predicted*, *X=None*, *sample_weight=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.ndarray*) – Extra data of shape [n_samples, n_features] necessary to calculate score
- **sample_weight** (*pd.DataFrame or np.ndarray*) – Sample weights used in computing objective value result

Returns score

evalml.objectives.F1Weighted.validate_inputs

F1Weighted.**validate_inputs**(*y_true*, *y_predicted*)

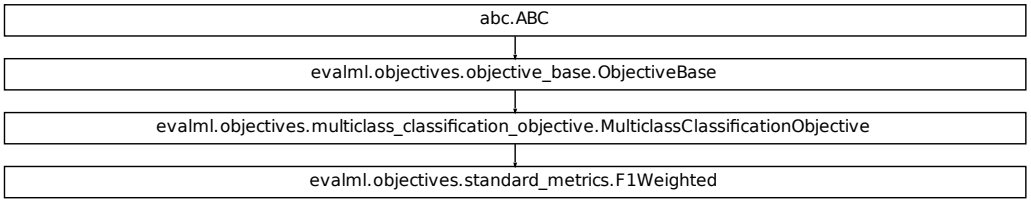
Validates the input based on a few simple checks.

Parameters

- **y_predicted** (*pd.Series, or pd.DataFrame*) – Predicted values of length [n_samples]
- **y_true** (*pd.Series*) – Actual class labels of length [n_samples]

Returns None

Class Inheritance



evalml.objectives.LogLossBinary

class evalml.objectives.**LogLossBinary**

Log Loss for binary classification.

name = 'Log Loss Binary'

greater_is_better = False

perfect_score = 0.0

positive_only = False

problem_types = [<ProblemTypes.BINARY: 'binary'>, <ProblemTypes.TIME_SERIES_BINARY: 't

score_needs_proba = True

Methods

__init__	Initialize self.
calculate_percent_difference	Calculate the percent difference between scores.
decision_function	Apply a learned threshold to predicted probabilities to get predicted classes.
is_defined_for_problem_type	
objective_function	Computes the relative value of the provided predictions compared to the actual labels, according a specified metric
optimize_threshold	Learn a binary classification threshold which optimizes the current objective.
score	Returns a numerical score indicating performance based on the differences between the predicted and actual values.
validate_inputs	Validates the input based on a few simple checks.

evalml.objectives.LogLossBinary.__init__

LogLossBinary.**__init__**()
Initialize self. See help(type(self)) for accurate signature.

evalml.objectives.LogLossBinary.calculate_percent_difference

classmethod LogLossBinary.**calculate_percent_difference**(*score*, *baseline_score*)
Calculate the percent difference between scores.

Parameters

- **score** (*float*) – A score. Output of the score method of this objective.
- **baseline_score** (*float*) – A score. Output of the score method of this objective. In practice, this is the score achieved on this objective with a baseline estimator.

Returns

The percent difference between the scores. Note that for objectives that can be interpreted as percentages, this will be the difference between the reference score and score. For all other objectives, the difference will be normalized by the reference score.

Return type float

evalml.objectives.LogLossBinary.decision_function

LogLossBinary.**decision_function**(*ypred_proba*, *threshold=0.5*, *X=None*)
Apply a learned threshold to predicted probabilities to get predicted classes.

Parameters

- **ypred_proba** (*pd.Series, np.ndarray*) – 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.is_defined_for_problem_type

classmethod LogLossBinary.**is_defined_for_problem_type**(*problem_type*)

evalml.objectives.LogLossBinary.objective_function

LogLossBinary.**objective_function**(*y_true*, *y_predicted*, *X=None*, *sample_weight=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.ndarray): Extra data of shape [n_samples, n_features] necessary to calculate score sample_weight (pd.DataFrame or np.ndarray): Sample weights used in computing objective value result

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** (*pd.Series*) – The classifier's predicted probabilities
- **y_true** (*pd.Series*) – 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*, *sample_weight=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.ndarray*) – Extra data of shape [n_samples, n_features] necessary to calculate score
- **sample_weight** (*pd.DataFrame or np.ndarray*) – Sample weights used in computing objective value result

Returns score

evalml.objectives.LogLossBinary.validate_inputs

LogLossBinary.**validate_inputs**(*y_true*, *y_predicted*)

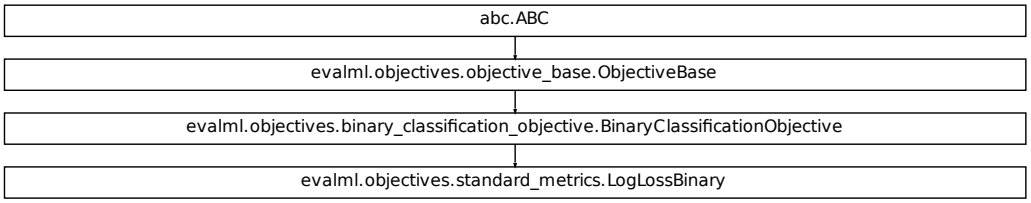
Validates the input based on a few simple checks.

Parameters

- **y_predicted** (*pd.Series, or pd.DataFrame*) – Predicted values of length [n_samples]
- **y_true** (*pd.Series*) – Actual class labels of length [n_samples]

Returns None

Class Inheritance



evalml.objectives.LogLossMulticlass

class evalml.objectives.**LogLossMulticlass**

Log Loss for multiclass classification.

name = 'Log Loss Multiclass'

greater_is_better = False

perfect_score = 0.0

positive_only = False

problem_types = [<ProblemTypes.MULTICLASS: 'multiclass'>, <ProblemTypes.TIME_SERIES_MU

score_needs_proba = True

Methods

__init__	Initialize self.
calculate_percent_difference	Calculate the percent difference between scores.
is_defined_for_problem_type	
objective_function	Computes the relative value of the provided predictions compared to the actual labels, according a specified metric
score	Returns a numerical score indicating performance based on the differences between the predicted and actual values.
validate_inputs	Validates the input based on a few simple checks.

evalml.objectives.LogLossMulticlass.__init__

LogLossMulticlass.**__init__**()
Initialize self. See help(type(self)) for accurate signature.

evalml.objectives.LogLossMulticlass.calculate_percent_difference

classmethod LogLossMulticlass.**calculate_percent_difference**(*score*, *baseline_score*)
Calculate the percent difference between scores.

Parameters

- **score** (*float*) – A score. Output of the score method of this objective.
- **baseline_score** (*float*) – A score. Output of the score method of this objective. In practice, this is the score achieved on this objective with a baseline estimator.

Returns

The percent difference between the scores. Note that for objectives that can be interpreted as percentages, this will be the difference between the reference score and score. For all other objectives, the difference will be normalized by the reference score.

Return type float

evalml.objectives.LogLossMulticlass.is_defined_for_problem_type

classmethod LogLossMulticlass.**is_defined_for_problem_type**(*problem_type*)

evalml.objectives.LogLossMulticlass.objective_function

LogLossMulticlass.**objective_function**(*y_true*, *y_predicted*, *X=None*, *sample_weight=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.ndarray): Extra data of shape [n_samples, n_features] necessary to calculate score sample_weight (pd.DataFrame or np.ndarray): Sample weights used in computing objective value result

Returns Numerical value used to calculate score

evalml.objectives.LogLossMulticlass.score

LogLossMulticlass.**score**(*y_true*, *y_predicted*, *X=None*, *sample_weight=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.ndarray*) – Extra data of shape [n_samples, n_features] necessary to calculate score
- **sample_weight** (*pd.DataFrame or np.ndarray*) – Sample weights used in computing objective value result

Returns score

evalml.objectives.LogLossMulticlass.validate_inputs

LogLossMulticlass.**validate_inputs**(*y_true*, *y_predicted*)

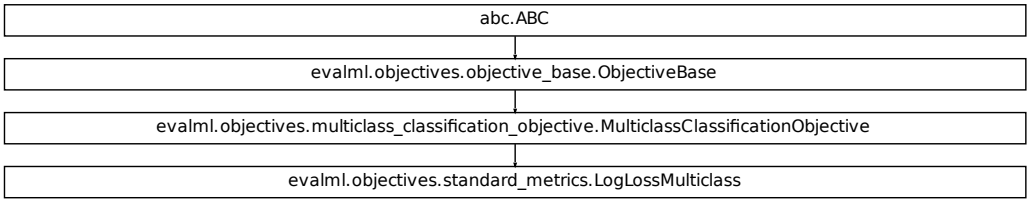
Validates the input based on a few simple checks.

Parameters

- **y_predicted** (*pd.Series, or pd.DataFrame*) – Predicted values of length [n_samples]
- **y_true** (*pd.Series*) – Actual class labels of length [n_samples]

Returns None

Class Inheritance



evalml.objectives.MCCBinary

class evalml.objectives.**MCCBinary**

Matthews correlation coefficient for binary classification.

name = 'MCC Binary'

greater_is_better = True

perfect_score = 1.0

positive_only = False

problem_types = [<ProblemTypes.BINARY: 'binary'>, <ProblemTypes.TIME_SERIES_BINARY: 't

score_needs_proba = False

Methods

__init__	Initialize self.
calculate_percent_difference	Calculate the percent difference between scores.
decision_function	Apply a learned threshold to predicted probabilities to get predicted classes.
is_defined_for_problem_type	
objective_function	Computes the relative value of the provided predictions compared to the actual labels, according a specified metric
optimize_threshold	Learn a binary classification threshold which optimizes the current objective.
score	Returns a numerical score indicating performance based on the differences between the predicted and actual values.
validate_inputs	Validates the input based on a few simple checks.

evalml.objectives.MCCBinary.__init__

MCCBinary.**__init__**()
Initialize self. See help(type(self)) for accurate signature.

evalml.objectives.MCCBinary.calculate_percent_difference

classmethod MCCBinary.**calculate_percent_difference**(*score*, *baseline_score*)
Calculate the percent difference between scores.

Parameters

- **score** (*float*) – A score. Output of the score method of this objective.
- **baseline_score** (*float*) – A score. Output of the score method of this objective. In practice, this is the score achieved on this objective with a baseline estimator.

Returns

The percent difference between the scores. Note that for objectives that can be interpreted as percentages, this will be the difference between the reference score and score. For all other objectives, the difference will be normalized by the reference score.

Return type float

evalml.objectives.MCCBinary.decision_function

MCCBinary.**decision_function**(*ypred_proba*, *threshold=0.5*, *X=None*)
Apply a learned threshold to predicted probabilities to get predicted classes.

Parameters

- **ypred_proba** (*pd.Series, np.ndarray*) – 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.is_defined_for_problem_type

classmethod MCCBinary.**is_defined_for_problem_type**(*problem_type*)

evalml.objectives.MCCBinary.objective_function

MCCBinary.**objective_function**(*y_true*, *y_predicted*, *X=None*, *sample_weight=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.ndarray): Extra data of shape [n_samples, n_features] necessary to calculate score sample_weight (pd.DataFrame or np.ndarray): Sample weights used in computing objective value result

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** (*pd.Series*) – The classifier's predicted probabilities
- **y_true** (*pd.Series*) – 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*, *sample_weight=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.ndarray*) – Extra data of shape [n_samples, n_features] necessary to calculate score
- **sample_weight** (*pd.DataFrame or np.ndarray*) – Sample weights used in computing objective value result

Returns score

evalml.objectives.MCCBinary.validate_inputs

MCCBinary.**validate_inputs**(*y_true*, *y_predicted*)

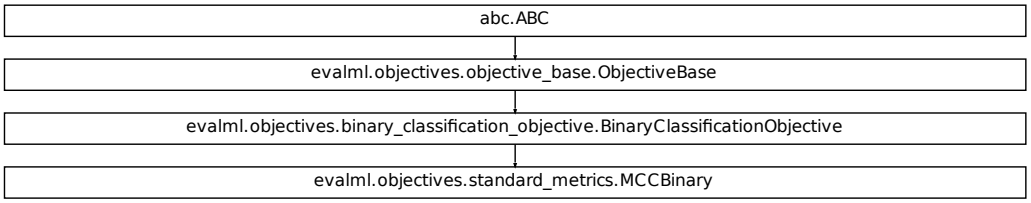
Validates the input based on a few simple checks.

Parameters

- **y_predicted** (*pd.Series, or pd.DataFrame*) – Predicted values of length [n_samples]
- **y_true** (*pd.Series*) – Actual class labels of length [n_samples]

Returns None

Class Inheritance



evalml.objectives.MCCMulticlass

class evalml.objectives.**MCCMulticlass**

Matthews correlation coefficient for multiclass classification.

name = 'MCC Multiclass'

greater_is_better = True

perfect_score = 1.0

positive_only = False

problem_types = [<ProblemTypes.MULTICLASS: 'multiclass'>, <ProblemTypes.TIME_SERIES_MU

score_needs_proba = False

Methods

__init__	Initialize self.
calculate_percent_difference	Calculate the percent difference between scores.
is_defined_for_problem_type	
objective_function	Computes the relative value of the provided predictions compared to the actual labels, according a specified metric
score	Returns a numerical score indicating performance based on the differences between the predicted and actual values.
validate_inputs	Validates the input based on a few simple checks.

evalml.objectives.MCCMulticlass.__init__

MCCMulticlass.**__init__**()
Initialize self. See help(type(self)) for accurate signature.

evalml.objectives.MCCMulticlass.calculate_percent_difference

classmethod MCCMulticlass.**calculate_percent_difference**(*score*, *baseline_score*)
Calculate the percent difference between scores.

Parameters

- **score** (*float*) – A score. Output of the score method of this objective.
- **baseline_score** (*float*) – A score. Output of the score method of this objective. In practice, this is the score achieved on this objective with a baseline estimator.

Returns

The percent difference between the scores. Note that for objectives that can be interpreted as percentages, this will be the difference between the reference score and score. For all other objectives, the difference will be normalized by the reference score.

Return type float

evalml.objectives.MCCMulticlass.is_defined_for_problem_type

classmethod MCCMulticlass.**is_defined_for_problem_type**(*problem_type*)

evalml.objectives.MCCMulticlass.objective_function

MCCMulticlass.**objective_function**(*y_true*, *y_predicted*, *X=None*, *sample_weight=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.ndarray): Extra data of shape [n_samples, n_features] necessary to calculate score sample_weight (pd.DataFrame or np.ndarray): Sample weights used in computing objective value result

Returns Numerical value used to calculate score

evalml.objectives.MCCMulticlass.score

MCCMulticlass.**score**(*y_true*, *y_predicted*, *X=None*, *sample_weight=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.ndarray*) – Extra data of shape [n_samples, n_features] necessary to calculate score
- **sample_weight** (*pd.DataFrame or np.ndarray*) – Sample weights used in computing objective value result

Returns score

evalml.objectives.MCCMulticlass.validate_inputs

MCCMulticlass.**validate_inputs**(*y_true*, *y_predicted*)

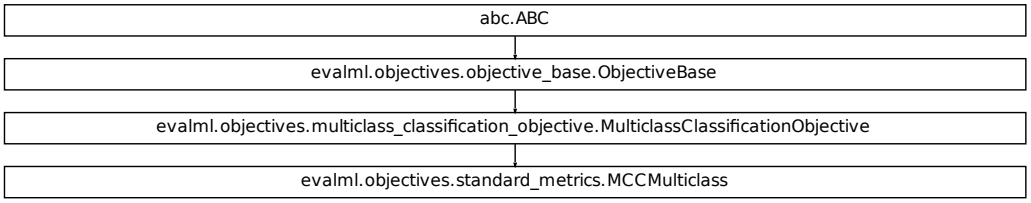
Validates the input based on a few simple checks.

Parameters

- **y_predicted** (*pd.Series, or pd.DataFrame*) – Predicted values of length [n_samples]
- **y_true** (*pd.Series*) – Actual class labels of length [n_samples]

Returns None

Class Inheritance



evalml.objectives.Precision

class evalml.objectives.**Precision**

Precision score for binary classification.

name = 'Precision'

greater_is_better = True

perfect_score = 1.0

positive_only = False

problem_types = [<ProblemTypes.BINARY: 'binary'>, <ProblemTypes.TIME_SERIES_BINARY: 't

score_needs_proba = False

Methods

__init__	Initialize self.
calculate_percent_difference	Calculate the percent difference between scores.
decision_function	Apply a learned threshold to predicted probabilities to get predicted classes.
is_defined_for_problem_type	
objective_function	Computes the relative value of the provided predictions compared to the actual labels, according a specified metric
optimize_threshold	Learn a binary classification threshold which optimizes the current objective.
score	Returns a numerical score indicating performance based on the differences between the predicted and actual values.
validate_inputs	Validates the input based on a few simple checks.

evalml.objectives.Precision.__init__

Precision.**__init__**()
Initialize self. See help(type(self)) for accurate signature.

evalml.objectives.Precision.calculate_percent_difference

classmethod Precision.**calculate_percent_difference**(*score*, *baseline_score*)
Calculate the percent difference between scores.

Parameters

- **score** (*float*) – A score. Output of the score method of this objective.
- **baseline_score** (*float*) – A score. Output of the score method of this objective. In practice, this is the score achieved on this objective with a baseline estimator.

Returns

The percent difference between the scores. Note that for objectives that can be interpreted as percentages, this will be the difference between the reference score and score. For all other objectives, the difference will be normalized by the reference score.

Return type float

evalml.objectives.Precision.decision_function

Precision.**decision_function**(*ypred_proba*, *threshold=0.5*, *X=None*)
Apply a learned threshold to predicted probabilities to get predicted classes.

Parameters

- **ypred_proba** (*pd.Series, np.ndarray*) – 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.is_defined_for_problem_type

classmethod Precision.**is_defined_for_problem_type**(*problem_type*)

evalml.objectives.Precision.objective_function

Precision.**objective_function**(*y_true*, *y_predicted*, *X=None*, *sample_weight=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.ndarray): Extra data of shape [n_samples, n_features] necessary to calculate score sample_weight (pd.DataFrame or np.ndarray): Sample weights used in computing objective value result

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** (*pd.Series*) – The classifier's predicted probabilities
- **y_true** (*pd.Series*) – 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*, *sample_weight=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.ndarray*) – Extra data of shape [n_samples, n_features] necessary to calculate score
- **sample_weight** (*pd.DataFrame or np.ndarray*) – Sample weights used in computing objective value result

Returns score

evalml.objectives.Precision.validate_inputs

Precision.**validate_inputs**(*y_true*, *y_predicted*)

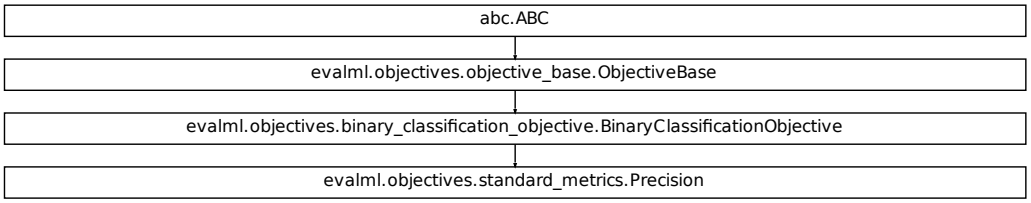
Validates the input based on a few simple checks.

Parameters

- **y_predicted** (*pd.Series, or pd.DataFrame*) – Predicted values of length [n_samples]
- **y_true** (*pd.Series*) – Actual class labels of length [n_samples]

Returns None

Class Inheritance



evalml.objectives.PrecisionMicro

class evalml.objectives.**PrecisionMicro**

Precision score for multiclass classification using micro averaging.

name = 'Precision Micro'

greater_is_better = True

perfect_score = 1.0

positive_only = False

problem_types = [<ProblemTypes.MULTICLASS: 'multiclass'>, <ProblemTypes.TIME_SERIES_MU

score_needs_proba = False

Methods

__init__	Initialize self.
calculate_percent_difference	Calculate the percent difference between scores.
is_defined_for_problem_type	
objective_function	Computes the relative value of the provided predictions compared to the actual labels, according a specified metric
score	Returns a numerical score indicating performance based on the differences between the predicted and actual values.
validate_inputs	Validates the input based on a few simple checks.

evalml.objectives.PrecisionMicro.__init__

PrecisionMicro.**__init__**()

Initialize self. See help(type(self)) for accurate signature.

evalml.objectives.PrecisionMicro.calculate_percent_difference

classmethod PrecisionMicro.**calculate_percent_difference**(*score*, *baseline_score*)

Calculate the percent difference between scores.

Parameters

- **score** (*float*) – A score. Output of the score method of this objective.
- **baseline_score** (*float*) – A score. Output of the score method of this objective. In practice, this is the score achieved on this objective with a baseline estimator.

Returns

The percent difference between the scores. Note that for objectives that can be interpreted as percentages, this will be the difference between the reference score and score. For all other objectives, the difference will be normalized by the reference score.

Return type float

evalml.objectives.PrecisionMicro.is_defined_for_problem_type

classmethod PrecisionMicro.**is_defined_for_problem_type**(*problem_type*)

evalml.objectives.PrecisionMicro.objective_function

PrecisionMicro.**objective_function**(*y_true*, *y_predicted*, *X=None*, *sample_weight=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.ndarray): Extra data of shape [n_samples, n_features] necessary to calculate score sample_weight (pd.DataFrame or np.ndarray): Sample weights used in computing objective value result

Returns Numerical value used to calculate score

evalml.objectives.PrecisionMicro.score

PrecisionMicro.**score**(*y_true*, *y_predicted*, *X=None*, *sample_weight=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.ndarray*) – Extra data of shape [n_samples, n_features] necessary to calculate score
- **sample_weight** (*pd.DataFrame or np.ndarray*) – Sample weights used in computing objective value result

Returns score

evalml.objectives.PrecisionMicro.validate_inputs

PrecisionMicro.**validate_inputs**(*y_true*, *y_predicted*)

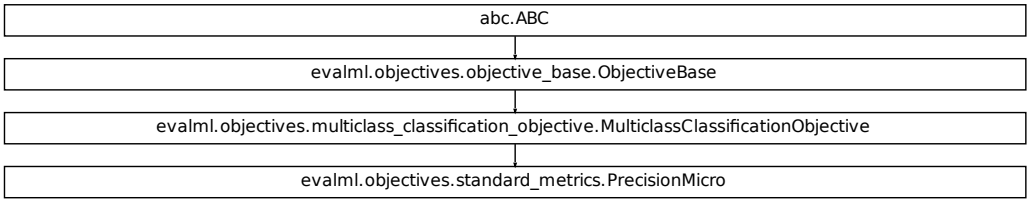
Validates the input based on a few simple checks.

Parameters

- **y_predicted** (*pd.Series, or pd.DataFrame*) – Predicted values of length [n_samples]
- **y_true** (*pd.Series*) – Actual class labels of length [n_samples]

Returns None

Class Inheritance



evalml.objectives.PrecisionMacro

class evalml.objectives.**PrecisionMacro**

Precision score for multiclass classification using macro averaging.

name = 'Precision Macro'

greater_is_better = True

perfect_score = 1.0

positive_only = False

problem_types = [<ProblemTypes.MULTICLASS: 'multiclass'>, <ProblemTypes.TIME_SERIES_MU

score_needs_proba = False

Methods

__init__	Initialize self.
calculate_percent_difference	Calculate the percent difference between scores.
is_defined_for_problem_type	
objective_function	Computes the relative value of the provided predictions compared to the actual labels, according a specified metric
score	Returns a numerical score indicating performance based on the differences between the predicted and actual values.
validate_inputs	Validates the input based on a few simple checks.

evalml.objectives.PrecisionMacro.__init__

PrecisionMacro.**__init__**()
Initialize self. See help(type(self)) for accurate signature.

evalml.objectives.PrecisionMacro.calculate_percent_difference

classmethod PrecisionMacro.**calculate_percent_difference**(*score*, *baseline_score*)
Calculate the percent difference between scores.

Parameters

- **score** (*float*) – A score. Output of the score method of this objective.
- **baseline_score** (*float*) – A score. Output of the score method of this objective. In practice, this is the score achieved on this objective with a baseline estimator.

Returns

The percent difference between the scores. Note that for objectives that can be interpreted as percentages, this will be the difference between the reference score and score. For all other objectives, the difference will be normalized by the reference score.

Return type float

evalml.objectives.PrecisionMacro.is_defined_for_problem_type

classmethod PrecisionMacro.**is_defined_for_problem_type**(*problem_type*)

evalml.objectives.PrecisionMacro.objective_function

PrecisionMacro.**objective_function**(*y_true*, *y_predicted*, *X=None*, *sample_weight=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.ndarray): Extra data of shape [n_samples, n_features] necessary to calculate score sample_weight (pd.DataFrame or np.ndarray): Sample weights used in computing objective value result

Returns Numerical value used to calculate score

evalml.objectives.PrecisionMacro.score

PrecisionMacro.**score**(*y_true*, *y_predicted*, *X=None*, *sample_weight=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.ndarray*) – Extra data of shape [n_samples, n_features] necessary to calculate score
- **sample_weight** (*pd.DataFrame or np.ndarray*) – Sample weights used in computing objective value result

Returns score

evalml.objectives.PrecisionMacro.validate_inputs

PrecisionMacro.**validate_inputs**(*y_true*, *y_predicted*)

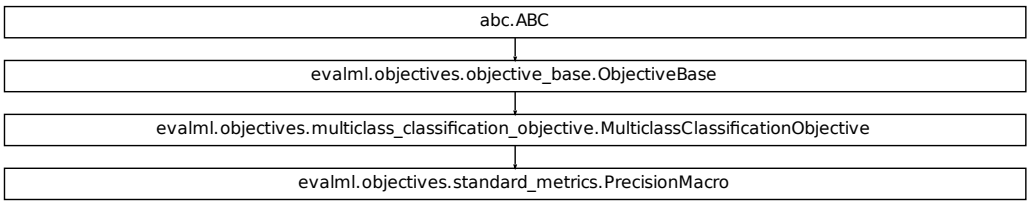
Validates the input based on a few simple checks.

Parameters

- **y_predicted** (*pd.Series, or pd.DataFrame*) – Predicted values of length [n_samples]
- **y_true** (*pd.Series*) – Actual class labels of length [n_samples]

Returns None

Class Inheritance



evalml.objectives.PrecisionWeighted

class evalml.objectives.**PrecisionWeighted**

Precision score for multiclass classification using weighted averaging.

name = 'Precision Weighted'

greater_is_better = True

perfect_score = 1.0

positive_only = False

problem_types = [<ProblemTypes.MULTICLASS: 'multiclass'>, <ProblemTypes.TIME_SERIES_MU

score_needs_proba = False

Methods

__init__	Initialize self.
calculate_percent_difference	Calculate the percent difference between scores.
is_defined_for_problem_type	
objective_function	Computes the relative value of the provided predictions compared to the actual labels, according a specified metric
score	Returns a numerical score indicating performance based on the differences between the predicted and actual values.
validate_inputs	Validates the input based on a few simple checks.

evalml.objectives.PrecisionWeighted.__init__

PrecisionWeighted.**__init__**()

Initialize self. See help(type(self)) for accurate signature.

evalml.objectives.PrecisionWeighted.calculate_percent_difference

classmethod PrecisionWeighted.**calculate_percent_difference**(*score*, *baseline_score*)

Calculate the percent difference between scores.

Parameters

- **score** (*float*) – A score. Output of the score method of this objective.
- **baseline_score** (*float*) – A score. Output of the score method of this objective. In practice, this is the score achieved on this objective with a baseline estimator.

Returns

The percent difference between the scores. Note that for objectives that can be interpreted as percentages, this will be the difference between the reference score and score. For all other objectives, the difference will be normalized by the reference score.

Return type float

evalml.objectives.PrecisionWeighted.is_defined_for_problem_type

classmethod PrecisionWeighted.**is_defined_for_problem_type**(*problem_type*)

evalml.objectives.PrecisionWeighted.objective_function

PrecisionWeighted.**objective_function**(*y_true*, *y_predicted*, *X=None*, *sample_weight=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.ndarray): Extra data of shape [n_samples, n_features] necessary to calculate score sample_weight (pd.DataFrame or np.ndarray): Sample weights used in computing objective value result

Returns Numerical value used to calculate score

evalml.objectives.PrecisionWeighted.score

PrecisionWeighted.**score**(*y_true*, *y_predicted*, *X=None*, *sample_weight=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.ndarray*) – Extra data of shape [n_samples, n_features] necessary to calculate score
- **sample_weight** (*pd.DataFrame or np.ndarray*) – Sample weights used in computing objective value result

Returns score

evalml.objectives.PrecisionWeighted.validate_inputs

PrecisionWeighted.**validate_inputs**(*y_true*, *y_predicted*)

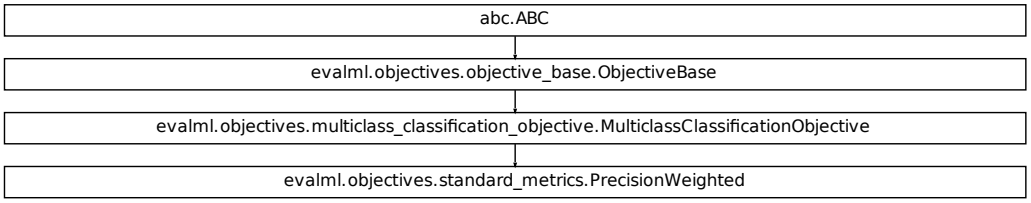
Validates the input based on a few simple checks.

Parameters

- **y_predicted** (*pd.Series, or pd.DataFrame*) – Predicted values of length [n_samples]
- **y_true** (*pd.Series*) – Actual class labels of length [n_samples]

Returns None

Class Inheritance



evalml.objectives.Recall

class evalml.objectives.**Recall**

Recall score for binary classification.

name = 'Recall'

greater_is_better = True

perfect_score = 1.0

positive_only = False

problem_types = [<ProblemTypes.BINARY: 'binary'>, <ProblemTypes.TIME_SERIES_BINARY: 't

score_needs_proba = False

Methods

__init__	Initialize self.
calculate_percent_difference	Calculate the percent difference between scores.
decision_function	Apply a learned threshold to predicted probabilities to get predicted classes.
is_defined_for_problem_type	
objective_function	Computes the relative value of the provided predictions compared to the actual labels, according a specified metric
optimize_threshold	Learn a binary classification threshold which optimizes the current objective.
score	Returns a numerical score indicating performance based on the differences between the predicted and actual values.
validate_inputs	Validates the input based on a few simple checks.

evalml.objectives.Recall.__init__

Recall.**__init__**()
Initialize self. See help(type(self)) for accurate signature.

evalml.objectives.Recall.calculate_percent_difference

classmethod Recall.**calculate_percent_difference**(*score*, *baseline_score*)
Calculate the percent difference between scores.

Parameters

- **score** (*float*) – A score. Output of the score method of this objective.
- **baseline_score** (*float*) – A score. Output of the score method of this objective. In practice, this is the score achieved on this objective with a baseline estimator.

Returns

The percent difference between the scores. Note that for objectives that can be interpreted as percentages, this will be the difference between the reference score and score. For all other objectives, the difference will be normalized by the reference score.

Return type float

evalml.objectives.Recall.decision_function

Recall.**decision_function**(*ypred_proba*, *threshold=0.5*, *X=None*)
Apply a learned threshold to predicted probabilities to get predicted classes.

Parameters

- **ypred_proba** (*pd.Series, np.ndarray*) – 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.is_defined_for_problem_type

classmethod Recall.**is_defined_for_problem_type**(*problem_type*)

evalml.objectives.Recall.objective_function

Recall.**objective_function**(*y_true*, *y_predicted*, *X=None*, *sample_weight=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.ndarray): Extra data of shape [n_samples, n_features] necessary to calculate score sample_weight (pd.DataFrame or np.ndarray): Sample weights used in computing objective value result

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** (*pd.Series*) – The classifier's predicted probabilities
- **y_true** (*pd.Series*) – 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*, *sample_weight=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.ndarray*) – Extra data of shape [n_samples, n_features] necessary to calculate score
- **sample_weight** (*pd.DataFrame or np.ndarray*) – Sample weights used in computing objective value result

Returns score

evalml.objectives.Recall.validate_inputs

Recall.**validate_inputs**(*y_true*, *y_predicted*)

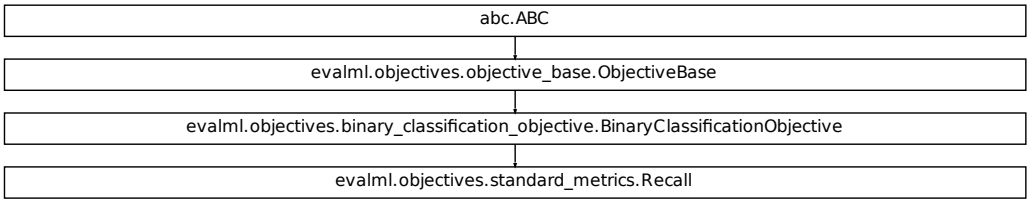
Validates the input based on a few simple checks.

Parameters

- **y_predicted** (*pd.Series, or pd.DataFrame*) – Predicted values of length [n_samples]
- **y_true** (*pd.Series*) – Actual class labels of length [n_samples]

Returns None

Class Inheritance



evalml.objectives.RecallMicro

class evalml.objectives.**RecallMicro**

Recall score for multiclass classification using micro averaging.

name = 'Recall Micro'

greater_is_better = True

perfect_score = 1.0

positive_only = False

problem_types = [<ProblemTypes.MULTICLASS: 'multiclass'>, <ProblemTypes.TIME_SERIES_MU

score_needs_proba = False

Methods

__init__	Initialize self.
calculate_percent_difference	Calculate the percent difference between scores.
is_defined_for_problem_type	
objective_function	Computes the relative value of the provided predictions compared to the actual labels, according a specified metric
score	Returns a numerical score indicating performance based on the differences between the predicted and actual values.
validate_inputs	Validates the input based on a few simple checks.

evalml.objectives.RecallMicro.__init__

RecallMicro.**__init__**()
Initialize self. See help(type(self)) for accurate signature.

evalml.objectives.RecallMicro.calculate_percent_difference

classmethod RecallMicro.**calculate_percent_difference**(*score*, *baseline_score*)
Calculate the percent difference between scores.

Parameters

- **score** (*float*) – A score. Output of the score method of this objective.
- **baseline_score** (*float*) – A score. Output of the score method of this objective. In practice, this is the score achieved on this objective with a baseline estimator.

Returns

The percent difference between the scores. Note that for objectives that can be interpreted as percentages, this will be the difference between the reference score and score. For all other objectives, the difference will be normalized by the reference score.

Return type float

evalml.objectives.RecallMicro.is_defined_for_problem_type

classmethod RecallMicro.**is_defined_for_problem_type**(*problem_type*)

evalml.objectives.RecallMicro.objective_function

RecallMicro.**objective_function**(*y_true*, *y_predicted*, *X=None*, *sample_weight=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.ndarray): Extra data of shape [n_samples, n_features] necessary to calculate score sample_weight (pd.DataFrame or np.ndarray): Sample weights used in computing objective value result

Returns Numerical value used to calculate score

evalml.objectives.RecallMicro.score

RecallMicro.**score**(*y_true*, *y_predicted*, *X=None*, *sample_weight=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.ndarray*) – Extra data of shape [n_samples, n_features] necessary to calculate score
- **sample_weight** (*pd.DataFrame or np.ndarray*) – Sample weights used in computing objective value result

Returns score

evalml.objectives.RecallMicro.validate_inputs

RecallMicro.**validate_inputs**(*y_true*, *y_predicted*)

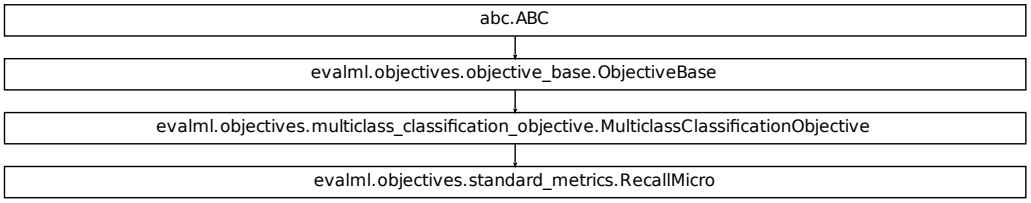
Validates the input based on a few simple checks.

Parameters

- **y_predicted** (*pd.Series, or pd.DataFrame*) – Predicted values of length [n_samples]
- **y_true** (*pd.Series*) – Actual class labels of length [n_samples]

Returns None

Class Inheritance



evalml.objectives.RecallMacro

class evalml.objectives.**RecallMacro**

Recall score for multiclass classification using macro averaging.

name = 'Recall Macro'

greater_is_better = True

perfect_score = 1.0

positive_only = False

problem_types = [<ProblemTypes.MULTICLASS: 'multiclass'>, <ProblemTypes.TIME_SERIES_MU

score_needs_proba = False

Methods

__init__	Initialize self.
calculate_percent_difference	Calculate the percent difference between scores.
is_defined_for_problem_type	
objective_function	Computes the relative value of the provided predictions compared to the actual labels, according a specified metric
score	Returns a numerical score indicating performance based on the differences between the predicted and actual values.
validate_inputs	Validates the input based on a few simple checks.

evalml.objectives.RecallMacro.__init__

RecallMacro.**__init__**()
Initialize self. See help(type(self)) for accurate signature.

evalml.objectives.RecallMacro.calculate_percent_difference

classmethod RecallMacro.**calculate_percent_difference**(*score*, *baseline_score*)
Calculate the percent difference between scores.

Parameters

- **score** (*float*) – A score. Output of the score method of this objective.
- **baseline_score** (*float*) – A score. Output of the score method of this objective. In practice, this is the score achieved on this objective with a baseline estimator.

Returns

The percent difference between the scores. Note that for objectives that can be interpreted as percentages, this will be the difference between the reference score and score. For all other objectives, the difference will be normalized by the reference score.

Return type float

evalml.objectives.RecallMacro.is_defined_for_problem_type

classmethod RecallMacro.**is_defined_for_problem_type**(*problem_type*)

evalml.objectives.RecallMacro.objective_function

RecallMacro.**objective_function**(*y_true*, *y_predicted*, *X=None*, *sample_weight=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.ndarray): Extra data of shape [n_samples, n_features] necessary to calculate score sample_weight (pd.DataFrame or np.ndarray): Sample weights used in computing objective value result

Returns Numerical value used to calculate score

evalml.objectives.RecallMacro.score

RecallMacro.**score**(*y_true*, *y_predicted*, *X=None*, *sample_weight=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.ndarray*) – Extra data of shape [n_samples, n_features] necessary to calculate score
- **sample_weight** (*pd.DataFrame or np.ndarray*) – Sample weights used in computing objective value result

Returns score

evalml.objectives.RecallMacro.validate_inputs

RecallMacro.**validate_inputs**(*y_true*, *y_predicted*)

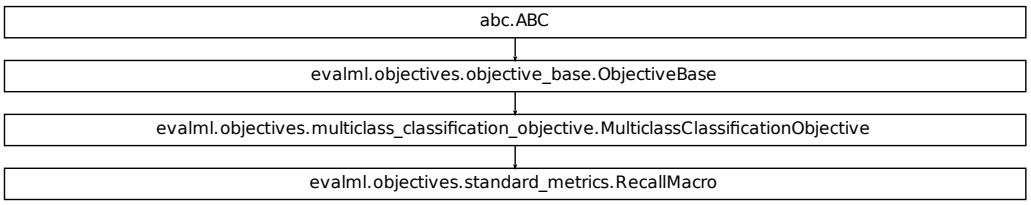
Validates the input based on a few simple checks.

Parameters

- **y_predicted** (*pd.Series, or pd.DataFrame*) – Predicted values of length [n_samples]
- **y_true** (*pd.Series*) – Actual class labels of length [n_samples]

Returns None

Class Inheritance



evalml.objectives.RecallWeighted

class evalml.objectives.**RecallWeighted**

Recall score for multiclass classification using weighted averaging.

name = 'Recall Weighted'

greater_is_better = True

perfect_score = 1.0

positive_only = False

problem_types = [<ProblemTypes.MULTICLASS: 'multiclass'>, <ProblemTypes.TIME_SERIES_MU

score_needs_proba = False

Methods

__init__	Initialize self.
calculate_percent_difference	Calculate the percent difference between scores.
is_defined_for_problem_type	
objective_function	Computes the relative value of the provided predictions compared to the actual labels, according a specified metric
score	Returns a numerical score indicating performance based on the differences between the predicted and actual values.
validate_inputs	Validates the input based on a few simple checks.

evalml.objectives.RecallWeighted.__init__

RecallWeighted.**__init__**()

Initialize self. See help(type(self)) for accurate signature.

evalml.objectives.RecallWeighted.calculate_percent_difference

classmethod RecallWeighted.**calculate_percent_difference**(*score*, *baseline_score*)

Calculate the percent difference between scores.

Parameters

- **score** (*float*) – A score. Output of the score method of this objective.
- **baseline_score** (*float*) – A score. Output of the score method of this objective. In practice, this is the score achieved on this objective with a baseline estimator.

Returns

The percent difference between the scores. Note that for objectives that can be interpreted as percentages, this will be the difference between the reference score and score. For all other objectives, the difference will be normalized by the reference score.

Return type float

evalml.objectives.RecallWeighted.is_defined_for_problem_type

classmethod RecallWeighted.**is_defined_for_problem_type**(*problem_type*)

evalml.objectives.RecallWeighted.objective_function

RecallWeighted.**objective_function**(*y_true*, *y_predicted*, *X=None*, *sample_weight=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.ndarray): Extra data of shape [n_samples, n_features] necessary to calculate score sample_weight (pd.DataFrame or np.ndarray): Sample weights used in computing objective value result

Returns Numerical value used to calculate score

evalml.objectives.RecallWeighted.score

RecallWeighted.**score**(*y_true*, *y_predicted*, *X=None*, *sample_weight=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.ndarray*) – Extra data of shape [n_samples, n_features] necessary to calculate score
- **sample_weight** (*pd.DataFrame or np.ndarray*) – Sample weights used in computing objective value result

Returns score

evalml.objectives.RecallWeighted.validate_inputs

RecallWeighted.**validate_inputs**(*y_true*, *y_predicted*)

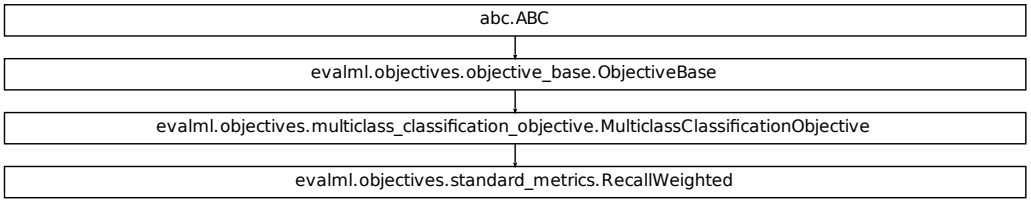
Validates the input based on a few simple checks.

Parameters

- **y_predicted** (*pd.Series, or pd.DataFrame*) – Predicted values of length [n_samples]
- **y_true** (*pd.Series*) – Actual class labels of length [n_samples]

Returns None

Class Inheritance



5.8.4 Regression Objectives

R2	Coefficient of determination for regression.
MAE	Mean absolute error for regression.
MAPE	Mean absolute percentage error for time series regression.
MSE	Mean squared error for regression.
MeanSquaredLogError	Mean squared log error for regression.
MedianAE	Median absolute error for regression.
MaxError	Maximum residual error for regression.
ExpVariance	Explained variance score for regression.
RootMeanSquaredError	Root mean squared error for regression.
RootMeanSquaredLogError	Root mean squared log error for regression.

evalml.objectives.R2

class evalml.objectives.**R2**

Coefficient of determination for regression.

name = 'R2'

greater_is_better = True

perfect_score = 1

positive_only = False

problem_types = [<ProblemTypes.REGRESSION: 'regression'>, <ProblemTypes.TIME_SERIES_RE

score_needs_proba = False

Methods

__init__	Initialize self.
calculate_percent_difference	Calculate the percent difference between scores.
is_defined_for_problem_type	
objective_function	Computes the relative value of the provided predictions compared to the actual labels, according a specified metric
score	Returns a numerical score indicating performance based on the differences between the predicted and actual values.
validate_inputs	Validates the input based on a few simple checks.

evalml.objectives.R2.__init__

R2.**__init__**()
Initialize self. See help(type(self)) for accurate signature.

evalml.objectives.R2.calculate_percent_difference

classmethod R2.**calculate_percent_difference**(*score*, *baseline_score*)
Calculate the percent difference between scores.

Parameters

- **score** (*float*) – A score. Output of the score method of this objective.
- **baseline_score** (*float*) – A score. Output of the score method of this objective. In practice, this is the score achieved on this objective with a baseline estimator.

Returns

The percent difference between the scores. Note that for objectives that can be interpreted as percentages, this will be the difference between the reference score and score. For all other objectives, the difference will be normalized by the reference score.

Return type float

evalml.objectives.R2.is_defined_for_problem_type

classmethod R2.**is_defined_for_problem_type**(*problem_type*)

evalml.objectives.R2.objective_function

R2.**objective_function**(*y_true*, *y_predicted*, *X=None*, *sample_weight=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.ndarray): Extra data of shape [n_samples, n_features] necessary to calculate score sample_weight (pd.DataFrame or np.ndarray): Sample weights used in computing objective value result

Returns Numerical value used to calculate score

evalml.objectives.R2.score

R2.**score**(*y_true*, *y_predicted*, *X=None*, *sample_weight=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.ndarray*) – Extra data of shape [n_samples, n_features] necessary to calculate score
- **sample_weight** (*pd.DataFrame or np.ndarray*) – Sample weights used in computing objective value result

Returns score

evalml.objectives.R2.validate_inputs

R2.**validate_inputs**(*y_true*, *y_predicted*)

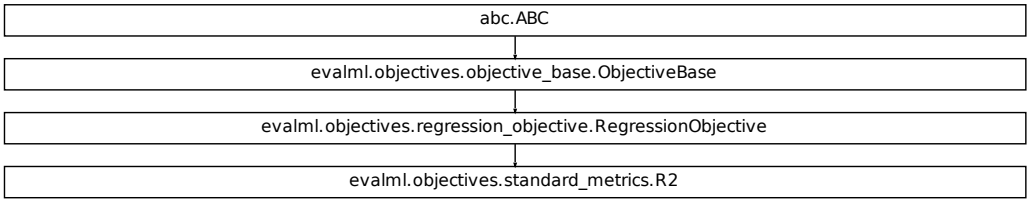
Validates the input based on a few simple checks.

Parameters

- **y_predicted** (*pd.Series, or pd.DataFrame*) – Predicted values of length [n_samples]
- **y_true** (*pd.Series*) – Actual class labels of length [n_samples]

Returns None

Class Inheritance



evalml.objectives.MAE

class evalml.objectives.**MAE**

Mean absolute error for regression.

name = 'MAE'

greater_is_better = False

perfect_score = 0.0

positive_only = False

problem_types = [<ProblemTypes.REGRESSION: 'regression'>, <ProblemTypes.TIME_SERIES_RE

score_needs_proba = False

Methods

__init__	Initialize self.
calculate_percent_difference	Calculate the percent difference between scores.
is_defined_for_problem_type	
objective_function	Computes the relative value of the provided predictions compared to the actual labels, according a specified metric
score	Returns a numerical score indicating performance based on the differences between the predicted and actual values.
validate_inputs	Validates the input based on a few simple checks.

evalml.objectives.MAE.__init__

MAE.**__init__**()
Initialize self. See help(type(self)) for accurate signature.

evalml.objectives.MAE.calculate_percent_difference

classmethod MAE.**calculate_percent_difference**(*score*, *baseline_score*)
Calculate the percent difference between scores.

Parameters

- **score** (*float*) – A score. Output of the score method of this objective.
- **baseline_score** (*float*) – A score. Output of the score method of this objective. In practice, this is the score achieved on this objective with a baseline estimator.

Returns

The percent difference between the scores. Note that for objectives that can be interpreted as percentages, this will be the difference between the reference score and score. For all other objectives, the difference will be normalized by the reference score.

Return type float

evalml.objectives.MAE.is_defined_for_problem_type

classmethod MAE.**is_defined_for_problem_type**(*problem_type*)

evalml.objectives.MAE.objective_function

MAE.**objective_function**(*y_true*, *y_predicted*, *X=None*, *sample_weight=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.ndarray): Extra data of shape [n_samples, n_features] necessary to calculate score sample_weight (pd.DataFrame or np.ndarray): Sample weights used in computing objective value result

Returns Numerical value used to calculate score

evalml.objectives.MAE.score

MAE.**score**(*y_true*, *y_predicted*, *X=None*, *sample_weight=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.ndarray*) – Extra data of shape [n_samples, n_features] necessary to calculate score
- **sample_weight** (*pd.DataFrame or np.ndarray*) – Sample weights used in computing objective value result

Returns score

evalml.objectives.MAE.validate_inputs

MAE.**validate_inputs**(*y_true*, *y_predicted*)

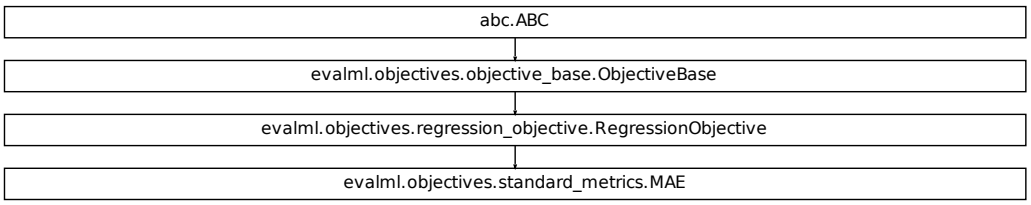
Validates the input based on a few simple checks.

Parameters

- **y_predicted** (*pd.Series, or pd.DataFrame*) – Predicted values of length [n_samples]
- **y_true** (*pd.Series*) – Actual class labels of length [n_samples]

Returns None

Class Inheritance



evalml.objectives.MAPE

class evalml.objectives.**MAPE**

Mean absolute percentage error for time series regression. Scaled by 100 to return a percentage.

Only valid for nonzero inputs. Otherwise, will throw a ValueError

name = 'Mean Absolute Percentage Error'

greater_is_better = False

perfect_score = 0.0

positive_only = True

problem_types = [<ProblemTypes.TIME_SERIES_REGRESSION: 'time series regression'>]

score_needs_proba = False

Methods

__init__	Initialize self.
calculate_percent_difference	Calculate the percent difference between scores.
is_defined_for_problem_type	
objective_function	Computes the relative value of the provided predictions compared to the actual labels, according a specified metric
score	Returns a numerical score indicating performance based on the differences between the predicted and actual values.
validate_inputs	Validates the input based on a few simple checks.

evalml.objectives.MAPE.__init__

MAPE.**__init__**()
Initialize self. See help(type(self)) for accurate signature.

evalml.objectives.MAPE.calculate_percent_difference

classmethod MAPE.**calculate_percent_difference**(*score*, *baseline_score*)
Calculate the percent difference between scores.

Parameters

- **score** (*float*) – A score. Output of the score method of this objective.
- **baseline_score** (*float*) – A score. Output of the score method of this objective. In practice, this is the score achieved on this objective with a baseline estimator.

Returns

The percent difference between the scores. Note that for objectives that can be interpreted as percentages, this will be the difference between the reference score and score. For all other objectives, the difference will be normalized by the reference score.

Return type float

evalml.objectives.MAPE.is_defined_for_problem_type

classmethod MAPE.**is_defined_for_problem_type**(*problem_type*)

evalml.objectives.MAPE.objective_function

MAPE.**objective_function**(*y_true*, *y_predicted*, *X=None*, *sample_weight=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.ndarray): Extra data of shape [n_samples, n_features] necessary to calculate score sample_weight (pd.DataFrame or np.ndarray): Sample weights used in computing objective value result

Returns Numerical value used to calculate score

evalml.objectives.MAPE.score

MAPE.**score**(*y_true*, *y_predicted*, *X=None*, *sample_weight=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.ndarray*) – Extra data of shape [n_samples, n_features] necessary to calculate score
- **sample_weight** (*pd.DataFrame or np.ndarray*) – Sample weights used in computing objective value result

Returns score

evalml.objectives.MAPE.validate_inputs

MAPE.**validate_inputs**(*y_true*, *y_predicted*)

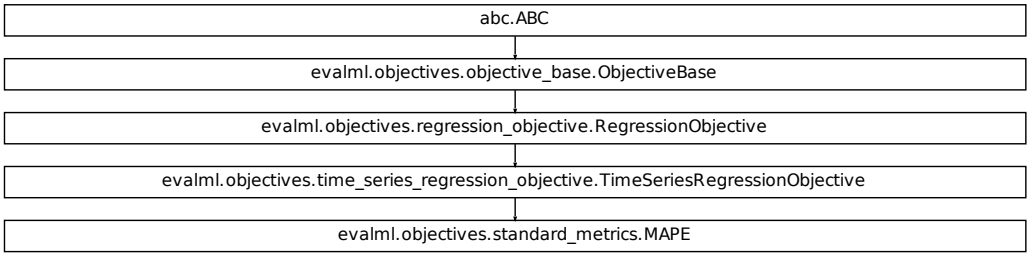
Validates the input based on a few simple checks.

Parameters

- **y_predicted** (*pd.Series, or pd.DataFrame*) – Predicted values of length [n_samples]
- **y_true** (*pd.Series*) – Actual class labels of length [n_samples]

Returns None

Class Inheritance



evalml.objectives.MSE

class evalml.objectives.**MSE**

Mean squared error for regression.

name = 'MSE'

greater_is_better = False

perfect_score = 0.0

positive_only = False

problem_types = [<ProblemTypes.REGRESSION: 'regression'>, <ProblemTypes.TIME_SERIES_RE

score_needs_proba = False

Methods

__init__	Initialize self.
calculate_percent_difference	Calculate the percent difference between scores.
is_defined_for_problem_type	
objective_function	Computes the relative value of the provided predictions compared to the actual labels, according a specified metric
score	Returns a numerical score indicating performance based on the differences between the predicted and actual values.
validate_inputs	Validates the input based on a few simple checks.

evalml.objectives.MSE.__init__

MSE.**__init__**()
Initialize self. See help(type(self)) for accurate signature.

evalml.objectives.MSE.calculate_percent_difference

classmethod MSE.**calculate_percent_difference**(*score*, *baseline_score*)
Calculate the percent difference between scores.

Parameters

- **score** (*float*) – A score. Output of the score method of this objective.
- **baseline_score** (*float*) – A score. Output of the score method of this objective. In practice, this is the score achieved on this objective with a baseline estimator.

Returns

The percent difference between the scores. Note that for objectives that can be interpreted as percentages, this will be the difference between the reference score and score. For all other objectives, the difference will be normalized by the reference score.

Return type float

evalml.objectives.MSE.is_defined_for_problem_type

classmethod MSE.**is_defined_for_problem_type**(*problem_type*)

evalml.objectives.MSE.objective_function

MSE.**objective_function**(*y_true*, *y_predicted*, *X=None*, *sample_weight=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.ndarray): Extra data of shape [n_samples, n_features] necessary to calculate score sample_weight (pd.DataFrame or np.ndarray): Sample weights used in computing objective value result

Returns Numerical value used to calculate score

evalml.objectives.MSE.score

MSE.**score**(*y_true*, *y_predicted*, *X=None*, *sample_weight=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.ndarray*) – Extra data of shape [n_samples, n_features] necessary to calculate score
- **sample_weight** (*pd.DataFrame or np.ndarray*) – Sample weights used in computing objective value result

Returns score

evalml.objectives.MSE.validate_inputs

MSE.**validate_inputs**(*y_true*, *y_predicted*)

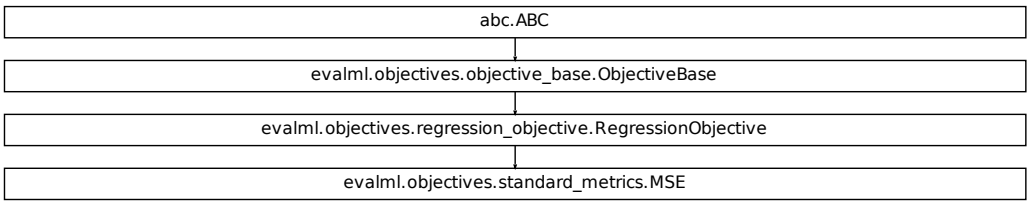
Validates the input based on a few simple checks.

Parameters

- **y_predicted** (*pd.Series, or pd.DataFrame*) – Predicted values of length [n_samples]
- **y_true** (*pd.Series*) – Actual class labels of length [n_samples]

Returns None

Class Inheritance



evalml.objectives.MeanSquaredLogError

class evalml.objectives.**MeanSquaredLogError**

Mean squared log error for regression.

Only valid for nonnegative inputs. Otherwise, will throw a ValueError

name = 'Mean Squared Log Error'

greater_is_better = False

perfect_score = 0.0

positive_only = True

problem_types = [<ProblemTypes.REGRESSION: 'regression'>, <ProblemTypes.TIME_SERIES_RE

score_needs_proba = False

Methods

__init__	Initialize self.
calculate_percent_difference	Calculate the percent difference between scores.
is_defined_for_problem_type	
objective_function	Computes the relative value of the provided predictions compared to the actual labels, according a specified metric
score	Returns a numerical score indicating performance based on the differences between the predicted and actual values.
validate_inputs	Validates the input based on a few simple checks.

evalml.objectives.MeanSquaredLogError.__init__

MeanSquaredLogError.**__init__**()
Initialize self. See help(type(self)) for accurate signature.

evalml.objectives.MeanSquaredLogError.calculate_percent_difference

classmethod MeanSquaredLogError.**calculate_percent_difference**(*score*, *baseline_score*)
Calculate the percent difference between scores.

Parameters

- **score** (*float*) – A score. Output of the score method of this objective.
- **baseline_score** (*float*) – A score. Output of the score method of this objective. In practice, this is the score achieved on this objective with a baseline estimator.

Returns

The percent difference between the scores. Note that for objectives that can be interpreted as percentages, this will be the difference between the reference score and score. For all other objectives, the difference will be normalized by the reference score.

Return type float

evalml.objectives.MeanSquaredLogError.is_defined_for_problem_type

classmethod MeanSquaredLogError.**is_defined_for_problem_type**(*problem_type*)

evalml.objectives.MeanSquaredLogError.objective_function

MeanSquaredLogError.**objective_function**(*y_true*, *y_predicted*, *X=None*, *sample_weight=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.ndarray): Extra data of shape [n_samples, n_features] necessary to calculate score sample_weight (pd.DataFrame or np.ndarray): Sample weights used in computing objective value result

Returns Numerical value used to calculate score

evalml.objectives.MeanSquaredLogError.score

MeanSquaredLogError.**score**(*y_true*, *y_predicted*, *X=None*, *sample_weight=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.ndarray*) – Extra data of shape [n_samples, n_features] necessary to calculate score
- **sample_weight** (*pd.DataFrame or np.ndarray*) – Sample weights used in computing objective value result

Returns score

evalml.objectives.MeanSquaredLogError.validate_inputs

MeanSquaredLogError.**validate_inputs**(*y_true*, *y_predicted*)

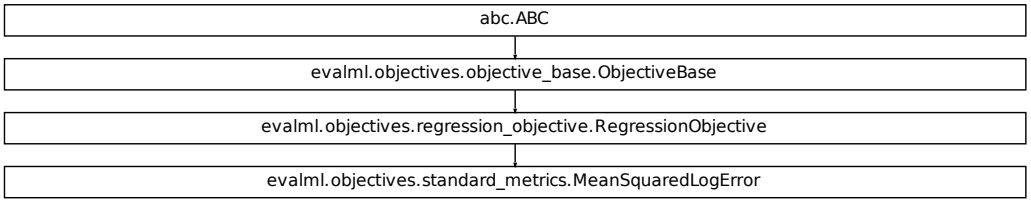
Validates the input based on a few simple checks.

Parameters

- **y_predicted** (*pd.Series, or pd.DataFrame*) – Predicted values of length [n_samples]
- **y_true** (*pd.Series*) – Actual class labels of length [n_samples]

Returns None

Class Inheritance



evalml.objectives.MedianAE

class evalml.objectives.**MedianAE**

Median absolute error for regression.

name = 'MedianAE'

greater_is_better = False

perfect_score = 0.0

positive_only = False

problem_types = [<ProblemTypes.REGRESSION: 'regression'>, <ProblemTypes.TIME_SERIES_RE

score_needs_proba = False

Methods

__init__	Initialize self.
calculate_percent_difference	Calculate the percent difference between scores.
is_defined_for_problem_type	
objective_function	Computes the relative value of the provided predictions compared to the actual labels, according a specified metric
score	Returns a numerical score indicating performance based on the differences between the predicted and actual values.
validate_inputs	Validates the input based on a few simple checks.

evalml.objectives.MedianAE.__init__

MedianAE.**__init__**()
Initialize self. See help(type(self)) for accurate signature.

evalml.objectives.MedianAE.calculate_percent_difference

classmethod MedianAE.**calculate_percent_difference**(*score*, *baseline_score*)
Calculate the percent difference between scores.

Parameters

- **score** (*float*) – A score. Output of the score method of this objective.
- **baseline_score** (*float*) – A score. Output of the score method of this objective. In practice, this is the score achieved on this objective with a baseline estimator.

Returns

The percent difference between the scores. Note that for objectives that can be interpreted as percentages, this will be the difference between the reference score and score. For all other objectives, the difference will be normalized by the reference score.

Return type float

evalml.objectives.MedianAE.is_defined_for_problem_type

classmethod MedianAE.**is_defined_for_problem_type**(*problem_type*)

evalml.objectives.MedianAE.objective_function

MedianAE.**objective_function**(*y_true*, *y_predicted*, *X=None*, *sample_weight=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.ndarray): Extra data of shape [n_samples, n_features] necessary to calculate score sample_weight (pd.DataFrame or np.ndarray): Sample weights used in computing objective value result

Returns Numerical value used to calculate score

evalml.objectives.MedianAE.score

MedianAE.**score**(*y_true*, *y_predicted*, *X=None*, *sample_weight=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.ndarray*) – Extra data of shape [n_samples, n_features] necessary to calculate score
- **sample_weight** (*pd.DataFrame or np.ndarray*) – Sample weights used in computing objective value result

Returns score

evalml.objectives.MedianAE.validate_inputs

MedianAE.**validate_inputs**(*y_true*, *y_predicted*)

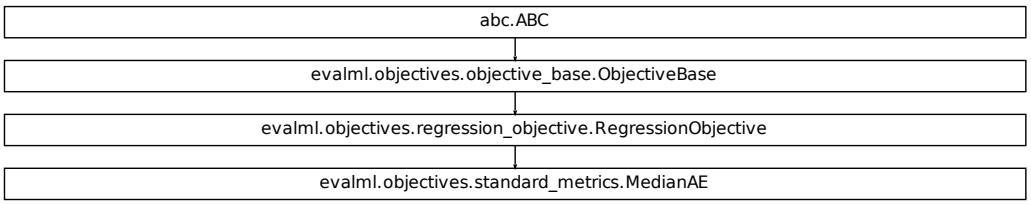
Validates the input based on a few simple checks.

Parameters

- **y_predicted** (*pd.Series, or pd.DataFrame*) – Predicted values of length [n_samples]
- **y_true** (*pd.Series*) – Actual class labels of length [n_samples]

Returns None

Class Inheritance



evalml.objectives.MaxError

class evalml.objectives.**MaxError**

Maximum residual error for regression.

name = 'MaxError'

greater_is_better = False

perfect_score = 0.0

positive_only = False

problem_types = [<ProblemTypes.REGRESSION: 'regression'>, <ProblemTypes.TIME_SERIES_RE

score_needs_proba = False

Methods

__init__	Initialize self.
calculate_percent_difference	Calculate the percent difference between scores.
is_defined_for_problem_type	
objective_function	Computes the relative value of the provided predictions compared to the actual labels, according a specified metric
score	Returns a numerical score indicating performance based on the differences between the predicted and actual values.
validate_inputs	Validates the input based on a few simple checks.

evalml.objectives.MaxError.__init__

MaxError.**__init__**()
Initialize self. See help(type(self)) for accurate signature.

evalml.objectives.MaxError.calculate_percent_difference

classmethod MaxError.**calculate_percent_difference**(*score*, *baseline_score*)
Calculate the percent difference between scores.

Parameters

- **score** (*float*) – A score. Output of the score method of this objective.
- **baseline_score** (*float*) – A score. Output of the score method of this objective. In practice, this is the score achieved on this objective with a baseline estimator.

Returns

The percent difference between the scores. Note that for objectives that can be interpreted as percentages, this will be the difference between the reference score and score. For all other objectives, the difference will be normalized by the reference score.

Return type float

evalml.objectives.MaxError.is_defined_for_problem_type

classmethod MaxError.**is_defined_for_problem_type**(*problem_type*)

evalml.objectives.MaxError.objective_function

MaxError.**objective_function**(*y_true*, *y_predicted*, *X=None*, *sample_weight=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.ndarray): Extra data of shape [n_samples, n_features] necessary to calculate score sample_weight (pd.DataFrame or np.ndarray): Sample weights used in computing objective value result

Returns Numerical value used to calculate score

evalml.objectives.MaxError.score

MaxError.**score**(*y_true*, *y_predicted*, *X=None*, *sample_weight=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.ndarray*) – Extra data of shape [n_samples, n_features] necessary to calculate score
- **sample_weight** (*pd.DataFrame or np.ndarray*) – Sample weights used in computing objective value result

Returns score

evalml.objectives.MaxError.validate_inputs

MaxError.**validate_inputs**(*y_true*, *y_predicted*)

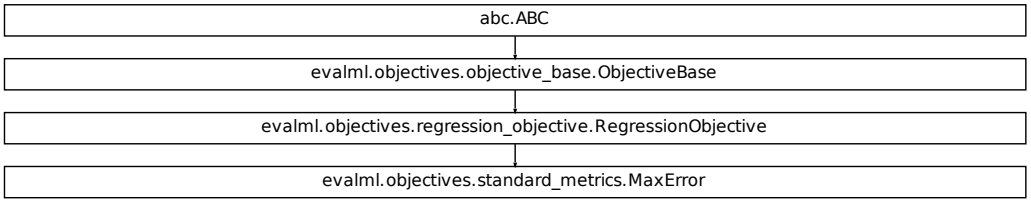
Validates the input based on a few simple checks.

Parameters

- **y_predicted** (*pd.Series, or pd.DataFrame*) – Predicted values of length [n_samples]
- **y_true** (*pd.Series*) – Actual class labels of length [n_samples]

Returns None

Class Inheritance



evalml.objectives.ExpVariance

class evalml.objectives.**ExpVariance**

Explained variance score for regression.

name = 'ExpVariance'

greater_is_better = True

perfect_score = 1.0

positive_only = False

problem_types = [<ProblemTypes.REGRESSION: 'regression'>, <ProblemTypes.TIME_SERIES_RE

score_needs_proba = False

Methods

__init__	Initialize self.
calculate_percent_difference	Calculate the percent difference between scores.
is_defined_for_problem_type	
objective_function	Computes the relative value of the provided predictions compared to the actual labels, according a specified metric
score	Returns a numerical score indicating performance based on the differences between the predicted and actual values.
validate_inputs	Validates the input based on a few simple checks.

evalml.objectives.ExpVariance.__init__

ExpVariance.**__init__**()
Initialize self. See help(type(self)) for accurate signature.

evalml.objectives.ExpVariance.calculate_percent_difference

classmethod ExpVariance.**calculate_percent_difference**(*score*, *baseline_score*)
Calculate the percent difference between scores.

Parameters

- **score** (*float*) – A score. Output of the score method of this objective.
- **baseline_score** (*float*) – A score. Output of the score method of this objective. In practice, this is the score achieved on this objective with a baseline estimator.

Returns

The percent difference between the scores. Note that for objectives that can be interpreted as percentages, this will be the difference between the reference score and score. For all other objectives, the difference will be normalized by the reference score.

Return type float

evalml.objectives.ExpVariance.is_defined_for_problem_type

classmethod ExpVariance.**is_defined_for_problem_type**(*problem_type*)

evalml.objectives.ExpVariance.objective_function

ExpVariance.**objective_function**(*y_true*, *y_predicted*, *X=None*, *sample_weight=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.ndarray): Extra data of shape [n_samples, n_features] necessary to calculate score sample_weight (pd.DataFrame or np.ndarray): Sample weights used in computing objective value result

Returns Numerical value used to calculate score

evalml.objectives.ExpVariance.score

ExpVariance.**score**(*y_true*, *y_predicted*, *X=None*, *sample_weight=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.ndarray*) – Extra data of shape [n_samples, n_features] necessary to calculate score
- **sample_weight** (*pd.DataFrame or np.ndarray*) – Sample weights used in computing objective value result

Returns score

evalml.objectives.ExpVariance.validate_inputs

ExpVariance.**validate_inputs**(*y_true*, *y_predicted*)

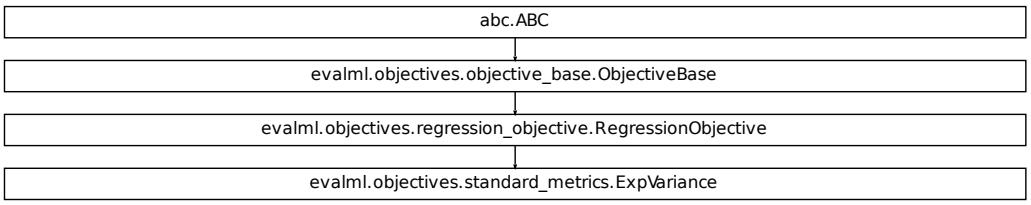
Validates the input based on a few simple checks.

Parameters

- **y_predicted** (*pd.Series, or pd.DataFrame*) – Predicted values of length [n_samples]
- **y_true** (*pd.Series*) – Actual class labels of length [n_samples]

Returns None

Class Inheritance



evalml.objectives.RootMeanSquaredError

class evalml.objectives.**RootMeanSquaredError**

Root mean squared error for regression.

name = 'Root Mean Squared Error'

greater_is_better = False

perfect_score = 0.0

positive_only = False

problem_types = [<ProblemTypes.REGRESSION: 'regression'>, <ProblemTypes.TIME_SERIES_RE

score_needs_proba = False

Methods

__init__	Initialize self.
calculate_percent_difference	Calculate the percent difference between scores.
is_defined_for_problem_type	
objective_function	Computes the relative value of the provided predictions compared to the actual labels, according a specified metric
score	Returns a numerical score indicating performance based on the differences between the predicted and actual values.
validate_inputs	Validates the input based on a few simple checks.

evalml.objectives.RootMeanSquaredError.__init__

RootMeanSquaredError.**__init__**()

Initialize self. See help(type(self)) for accurate signature.

evalml.objectives.RootMeanSquaredError.calculate_percent_difference

classmethod RootMeanSquaredError.**calculate_percent_difference**(*score*, *baseline_score*)

Calculate the percent difference between scores.

Parameters

- **score** (*float*) – A score. Output of the score method of this objective.
- **baseline_score** (*float*) – A score. Output of the score method of this objective. In practice, this is the score achieved on this objective with a baseline estimator.

Returns

The percent difference between the scores. Note that for objectives that can be interpreted as percentages, this will be the difference between the reference score and score. For all other objectives, the difference will be normalized by the reference score.

Return type float

evalml.objectives.RootMeanSquaredError.is_defined_for_problem_type

classmethod RootMeanSquaredError.**is_defined_for_problem_type**(*problem_type*)

evalml.objectives.RootMeanSquaredError.objective_function

RootMeanSquaredError.**objective_function**(*y_true*, *y_predicted*, *X=None*, *sample_weight=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.ndarray): Extra data of shape [n_samples, n_features] necessary to calculate score sample_weight (pd.DataFrame or np.ndarray): Sample weights used in computing objective value result

Returns Numerical value used to calculate score

evalml.objectives.RootMeanSquaredError.score

RootMeanSquaredError.**score**(*y_true*, *y_predicted*, *X=None*, *sample_weight=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.ndarray*) – Extra data of shape [n_samples, n_features] necessary to calculate score
- **sample_weight** (*pd.DataFrame or np.ndarray*) – Sample weights used in computing objective value result

Returns score

evalml.objectives.RootMeanSquaredError.validate_inputs

RootMeanSquaredError.**validate_inputs**(*y_true*, *y_predicted*)

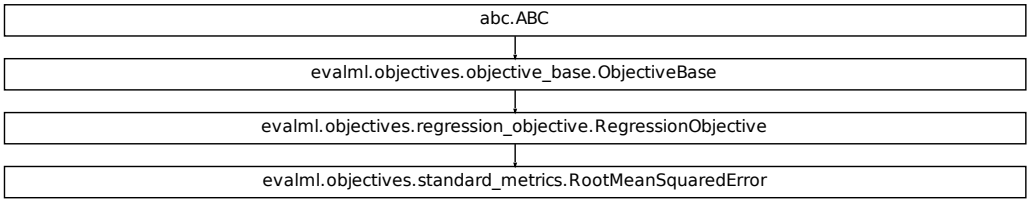
Validates the input based on a few simple checks.

Parameters

- **y_predicted** (*pd.Series, or pd.DataFrame*) – Predicted values of length [n_samples]
- **y_true** (*pd.Series*) – Actual class labels of length [n_samples]

Returns None

Class Inheritance



evalml.objectives.RootMeanSquaredLogError

class evalml.objectives.**RootMeanSquaredLogError**

Root mean squared log error for regression.

Only valid for nonnegative inputs.Otherwise, will throw a ValueError.

name = 'Root Mean Squared Log Error'

greater_is_better = False

perfect_score = 0.0

positive_only = True

problem_types = [<ProblemTypes.REGRESSION: 'regression'>, <ProblemTypes.TIME_SERIES_RE

score_needs_proba = False

Methods

__init__	Initialize self.
calculate_percent_difference	Calculate the percent difference between scores.
is_defined_for_problem_type	
objective_function	Computes the relative value of the provided predictions compared to the actual labels, according a specified metric
score	Returns a numerical score indicating performance based on the differences between the predicted and actual values.
validate_inputs	Validates the input based on a few simple checks.

evalml.objectives.RootMeanSquaredLogError.__init__

RootMeanSquaredLogError.**__init__**()
Initialize self. See help(type(self)) for accurate signature.

evalml.objectives.RootMeanSquaredLogError.calculate_percent_difference

classmethod RootMeanSquaredLogError.**calculate_percent_difference**(*score*, *baseline_score*)
Calculate the percent difference between scores.

Parameters

- **score** (*float*) – A score. Output of the score method of this objective.
- **baseline_score** (*float*) – A score. Output of the score method of this objective. In practice, this is the score achieved on this objective with a baseline estimator.

Returns

The percent difference between the scores. Note that for objectives that can be interpreted as percentages, this will be the difference between the reference score and score. For all other objectives, the difference will be normalized by the reference score.

Return type float

evalml.objectives.RootMeanSquaredLogError.is_defined_for_problem_type

classmethod RootMeanSquaredLogError.**is_defined_for_problem_type**(*problem_type*)

evalml.objectives.RootMeanSquaredLogError.objective_function

RootMeanSquaredLogError.**objective_function**(*y_true*, *y_predicted*, *X=None*, *sample_weight=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.ndarray): Extra data of shape [n_samples, n_features] necessary to calculate score sample_weight (pd.DataFrame or np.ndarray): Sample weights used in computing objective value result

Returns Numerical value used to calculate score

evalml.objectives.RootMeanSquaredLogError.score

RootMeanSquaredLogError.**score**(*y_true*, *y_predicted*, *X=None*, *sample_weight=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.ndarray*) – Extra data of shape [n_samples, n_features] necessary to calculate score
- **sample_weight** (*pd.DataFrame or np.ndarray*) – Sample weights used in computing objective value result

Returns score

evalml.objectives.RootMeanSquaredLogError.validate_inputs

RootMeanSquaredLogError.**validate_inputs**(*y_true*, *y_predicted*)

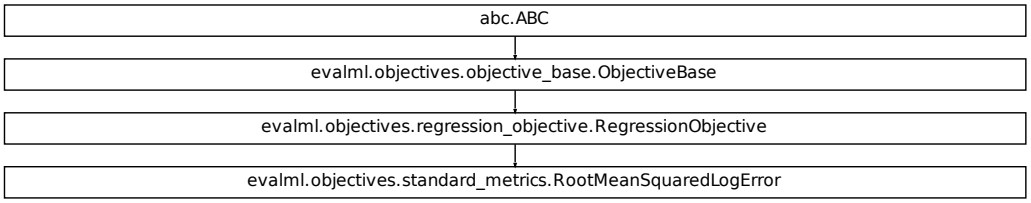
Validates the input based on a few simple checks.

Parameters

- **y_predicted** (*pd.Series, or pd.DataFrame*) – Predicted values of length [n_samples]
- **y_true** (*pd.Series*) – Actual class labels of length [n_samples]

Returns None

Class Inheritance



5.8.5 Objective Utils

get_all_objective_names	Get a list of the names of all objectives.
get_core_objectives	Returns all core objective instances associated with the given problem type.
get_core_objective_names	Get a list of all valid core objectives.
get_non_core_objectives	Get non-core objective classes.
get_objective	Returns the Objective class corresponding to a given objective name.

evalml.objectives.get_all_objective_names

evalml.objectives.**get_all_objective_names**()
Get a list of the names of all objectives.

Returns Objective names

Return type list (str)

evalml.objectives.get_core_objectives

evalml.objectives.**get_core_objectives**(*problem_type*)
Returns all core objective instances associated with the given problem type.

Core objectives are designed to work out-of-the-box for any dataset.

Parameters **problem_type** (*str/ProblemTypes*) – Type of problem

Returns List of ObjectiveBase instances

evalml.objectives.get_core_objective_names

evalml.objectives.**get_core_objective_names**()
Get a list of all valid core objectives.

Returns Objective names.

Return type list[str]

evalml.objectives.get_non_core_objectives

evalml.objectives.**get_non_core_objectives**()
Get non-core objective classes.

Non-core objectives are objectives that are domain-specific. Users typically need to configure these objectives before using them in AutoMLSearch.

Returns List of ObjectiveBase classes

evalml.objectives.get_objective

evalml.objectives.**get_objective**(*objective*, *return_instance=False*, ***kwargs*)

Returns the Objective class corresponding to a given objective name.

Parameters

- **objective** (*str or ObjectiveBase*) – Name or instance of the objective class.
- **return_instance** (*bool*) – Whether to return an instance of the objective. This only applies if objective is of type str. Note that the instance will be initialized with default arguments.
- **kwargs** (*Any*) – Any keyword arguments to pass into the objective. Only used when return_instance=True.

Returns ObjectiveBase if the parameter objective is of type ObjectiveBase. If objective is instead a valid objective name, function will return the class corresponding to that name. If return_instance is True, an instance of that objective will be returned.

5.9 Problem Types

handle_problem_types	Handles problem_type by either returning the ProblemTypes or converting from a str.
detect_problem_type	Determine the type of problem is being solved based on the targets (binary vs multiclass classification, regression)
ProblemTypes	Enum defining the supported types of machine learning problems.

5.9.1 evalml.problem_types.handle_problem_types

evalml.problem_types.**handle_problem_types**(*problem_type*)

Handles problem_type by either returning the ProblemTypes or converting from a str.

Parameters **problem_type** (*str or ProblemTypes*) – Problem type that needs to be handled

Returns ProblemTypes

5.9.2 evalml.problem_types.detect_problem_type

evalml.problem_types.**detect_problem_type**(*y*)

Determine the type of problem is being solved based on the targets (binary vs multiclass classification, regression)

Ignores missing and null data

Parameters **y** (*pd.Series*) – the target labels to predict

Returns ProblemType Enum

Return type ProblemType

Example

```
>>> y = pd.Series([0, 1, 0, 0, 0, 1, 1, 0, 0, 1, 1])
>>> problem_type = detect_problem_type(y)
>>> assert problem_type == ProblemTypes.BINARY
```

5.9.3 evalml.problem_types.ProblemTypes

class evalml.problem_types.**ProblemTypes**(*value*)

Enum defining the supported types of machine learning problems.

Attributes

BINARY	Binary classification problem.
MULTICLASS	Multiclass classification problem.
REGRESSION	Regression problem.
TIME_SERIES_BINARY	Time series binary classification problem.
TIME_SERIES_MULTICLASS	Time series multiclass classification problem.
TIME_SERIES_REGRESSION	Time series regression problem.

5.10 Model Family

handle_model_family	Handles model_family by either returning the ModelFamily or converting from a string
ModelFamily	Enum for family of machine learning models.

5.10.1 evalml.model_family.handle_model_family

evalml.model_family.**handle_model_family**(*model_family*)

Handles model_family by either returning the ModelFamily or converting from a string

Parameters **model_family** (*str or ModelFamily*) – Model type that needs to be handled

Returns ModelFamily

5.10.2 evalml.model_family.ModelFamily

class evalml.model_family.**ModelFamily**(*value*)
Enum for family of machine learning models.

Attributes

ARIMA	ARIMA model family.
BASELINE	Baseline model family.
CATBOOST	CatBoost model family.
DECISION_TREE	Decision Tree model family.
ENSEMBLE	Ensemble model family.
EXTRA_TREES	Extra Trees model family.
K_NEIGHBORS	K Nearest Neighbors model family.
LIGHTGBM	LightGBM model family.
LINEAR_MODEL	Linear model family.
NONE	None
RANDOM_FOREST	Random Forest model family.
SVM	SVM model family.
XGBOOST	XGBoost model family.

5.11 Tuners

Tuner	Defines API for Tuners.
SKOptTuner	Bayesian Optimizer.
GridSearchTuner	Grid Search Optimizer.
RandomSearchTuner	Random Search Optimizer.

5.11.1 evalml.tuners.Tuner

class evalml.tuners.**Tuner**(*pipeline_hyperparameter_ranges*, *random_seed=0*)
Defines API for Tuners.

Tuners implement different strategies for sampling from a search space. They're used in EvalML to search the space of pipeline hyperparameters.

Methods

__init__	Base Tuner class
add	Register a set of hyperparameters with the score obtained from training a pipeline with those hyperparameters.
is_search_space_exhausted	Optional.
propose	Returns a suggested set of parameters to train and score a pipeline with, based off the search space dimensions and prior samples.

evalml.tuners.Tuner.__init__

Tuner.**__init__**(*pipeline_hyperparameter_ranges*, *random_seed=0*)

Base Tuner class

Parameters

- **pipeline_hyperparameter_ranges** (*dict*) – a set of hyperparameter ranges corresponding to a pipeline's parameters
- **random_seed** (*int*) – The random state. Defaults to 0.

evalml.tuners.Tuner.add

abstract Tuner.**add**(*pipeline_parameters*, *score*)

Register a set of hyperparameters with the score obtained from training a pipeline with those hyperparameters.

Parameters

- **pipeline_parameters** (*dict*) – a dict of the parameters used to evaluate a pipeline
- **score** (*float*) – the score obtained by evaluating the pipeline with the provided parameters

Returns None

evalml.tuners.Tuner.is_search_space_exhausted

Tuner.**is_search_space_exhausted**()

Optional. If possible search space for tuner is finite, this method indicates whether or not all possible parameters have been scored.

Returns Returns true if all possible parameters in a search space has been scored.

Return type bool

evalml.tuners.Tuner.propose

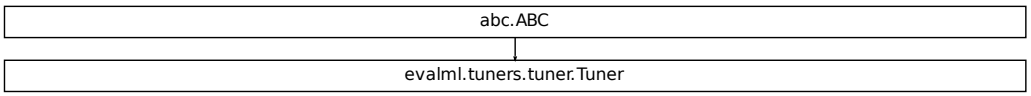
abstract Tuner.**propose**()

Returns a suggested set of parameters to train and score a pipeline with, based off the search space dimensions and prior samples.

Returns Proposed pipeline parameters

Return type dict

Class Inheritance



5.11.2 evalml.tuners.SKOptTuner

class evalml.tuners.**SKOptTuner**(*pipeline_hyperparameter_ranges*, *random_seed=0*)
Bayesian Optimizer.

Methods

__init__	Init SkOptTuner
add	Add score to sample
is_search_space_exhausted	Optional.
propose	Returns a suggested set of parameters to train and score a pipeline with, based off the search space dimensions and prior samples.

evalml.tuners.SKOptTuner.__init__

SKOptTuner.**__init__**(*pipeline_hyperparameter_ranges*, *random_seed=0*)
Init SkOptTuner

Parameters

- **pipeline_hyperparameter_ranges** (*dict*) – A set of hyperparameter ranges corresponding to a pipeline's parameters
- **random_seed** (*int*) – The seed for the random number generator. Defaults to 0.

evalml.tuners.SKOptTuner.add

SKOptTuner.**add**(*pipeline_parameters*, *score*)
Add score to sample

Parameters

- **pipeline_parameters** (*dict*) – A dict of the parameters used to evaluate a pipeline
- **score** (*float*) – The score obtained by evaluating the pipeline with the provided parameters

Returns None

evalml.tuners.SKOptTuner.is_search_space_exhausted

SKOptTuner.**is_search_space_exhausted**()

Optional. If possible search space for tuner is finite, this method indicates whether or not all possible parameters have been scored.

Returns Returns true if all possible parameters in a search space has been scored.

Return type bool

evalml.tuners.SKOptTuner.propose

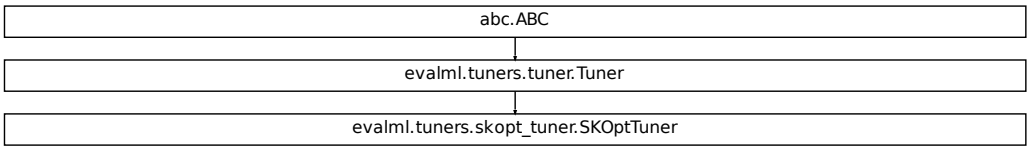
SKOptTuner.**propose**()

Returns a suggested set of parameters to train and score a pipeline with, based off the search space dimensions and prior samples.

Returns Proposed pipeline parameters

Return type dict

Class Inheritance



5.11.3 evalml.tuners.GridSearchTuner

class evalml.tuners.**GridSearchTuner**(*pipeline_hyperparameter_ranges*, *n_points=10*, *random_seed=0*)

Grid Search Optimizer.

Example

```
>>> tuner = GridSearchTuner({'My Component': {'param a': [0.0, 10.0], 'param b': [
↪'a', 'b', 'c']}}, n_points=5)
>>> proposal = tuner.propose()
>>> assert proposal.keys() == {'My Component'}
>>> assert proposal['My Component'] == {'param a': 0.0, 'param b': 'a'}
```

Methods

__init__	Generate all of the possible points to search for in the grid
add	Not applicable to grid search tuner as generated parameters are not dependent on scores of previous parameters.
is_search_space_exhausted	Checks if it is possible to generate a set of valid parameters.
propose	Returns parameters from _grid_points iterations

evalml.tuners.GridSearchTuner.__init__

GridSearchTuner.**__init__**(*pipeline_hyperparameter_ranges*, *n_points=10*, *random_seed=0*)

Generate all of the possible points to search for in the grid

Parameters

- **pipeline_hyperparameter_ranges** (*dict*) – a set of hyperparameter ranges corresponding to a pipeline's parameters
- **n_points** (*int*) – The number of points to sample from along each dimension defined in the `space` argument
- **random_seed** (*int*) – Seed for random number generator. Unused in this class, defaults to 0.

evalml.tuners.GridSearchTuner.add

GridSearchTuner.**add**(*pipeline_parameters*, *score*)

Not applicable to grid search tuner as generated parameters are not dependent on scores of previous parameters.

Parameters

- **pipeline_parameters** (*dict*) – a dict of the parameters used to evaluate a pipeline
- **score** (*float*) – the score obtained by evaluating the pipeline with the provided parameters

evalml.tuners.GridSearchTuner.is_search_space_exhausted

GridSearchTuner.**is_search_space_exhausted**()

Checks if it is possible to generate a set of valid parameters. Stores generated parameters in `self.curr_params` to be returned by `propose()`.

Raises **NoParamsException** – If a search space is exhausted, then this exception is thrown.

Returns If no more valid parameters exists in the search space, return false.

Return type bool

evalml.tuners.GridSearchTuner.propose

GridSearchTuner.**propose**()

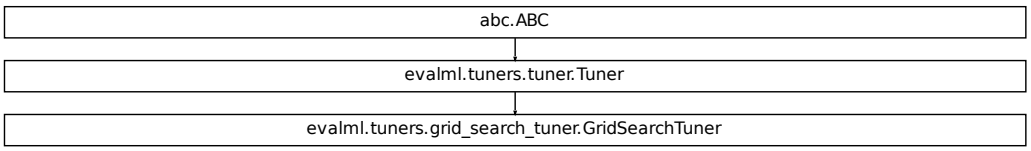
Returns parameters from _grid_points iterations

If all possible combinations of parameters have been scored, then NoParamsException is raised.

Returns proposed pipeline parameters

Return type dict

Class Inheritance



5.11.4 evalml.tuners.RandomSearchTuner

class evalml.tuners.**RandomSearchTuner**(*pipeline_hyperparameter_ranges*, *random_seed=0*, *with_replacement=False*, *replacement_max_attempts=10*)

Random Search Optimizer.

Example

```
>>> tuner = RandomSearchTuner({'My Component': {'param a': [0.0, 10.0], 'param b':
↪ ['a', 'b', 'c']}}, random_seed=42)
>>> proposal = tuner.propose()
>>> assert proposal.keys() == {'My Component'}
>>> assert proposal['My Component'] == {'param a': 3.7454011884736254, 'param b':
↪'c'}
```

Methods

__init__	Sets up check for duplication if needed.
add	Not applicable to random search tuner as generated parameters are not dependent on scores of previous parameters.
is_search_space_exhausted	Checks if it is possible to generate a set of valid parameters.
propose	Generate a unique set of parameters.

evalml.tuners.RandomSearchTuner.__init__

RandomSearchTuner.**__init__**(*pipeline_hyperparameter_ranges*, *random_seed=0*, *with_replacement=False*, *replacement_max_attempts=10*)

Sets up check for duplication if needed.

Parameters

- **pipeline_hyperparameter_ranges** (*dict*) – a set of hyperparameter ranges corresponding to a pipeline's parameters
- **random_state** (*int*) – Unused in this class. Defaults to 0.
- **with_replacement** (*bool*) – If false, only unique hyperparameters will be shown
- **replacement_max_attempts** (*int*) – The maximum number of tries to get a unique set of random parameters. Only used if tuner is initalized with with_replacement=True
- **random_seed** (*int*) – Seed for random number generator. Defaults to 0.

evalml.tuners.RandomSearchTuner.add

RandomSearchTuner.**add**(*pipeline_parameters*, *score*)

Not applicable to random search tuner as generated parameters are not dependent on scores of previous parameters.

Parameters

- **pipeline_parameters** (*dict*) – A dict of the parameters used to evaluate a pipeline
- **score** (*float*) – The score obtained by evaluating the pipeline with the provided parameters

evalml.tuners.RandomSearchTuner.is_search_space_exhausted

RandomSearchTuner.**is_search_space_exhausted**()

Checks if it is possible to generate a set of valid parameters. Stores generated parameters in `self.curr_params` to be returned by `propose()`.

Raises **NoParamsException** – If a search space is exhausted, then this exception is thrown.

Returns If no more valid parameters exists in the search space, return false.

Return type bool

evalml.tuners.RandomSearchTuner.propose

RandomSearchTuner.**propose**()

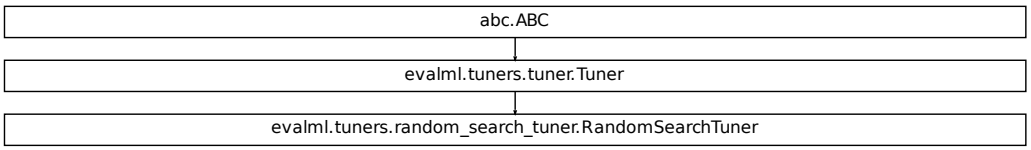
Generate a unique set of parameters.

If tuner was initialized with `with_replacement=True` and the tuner is unable to generate a unique set of parameters after `replacement_max_attempts` tries, then `NoParamsException` is raised.

Returns Proposed pipeline parameters

Return type dict

Class Inheritance



5.12 Data Checks

5.12.1 Data Check Classes

DataCheck	Base class for all data checks.
InvalidTargetDataCheck	Checks if the target data contains missing or invalid values.
HighlyNullDataCheck	Checks if there are any highly-null columns and rows in the input.
IDColumnsDataCheck	Check if any of the features are likely to be ID columns.
TargetLeakageDataCheck	Check if any of the features are highly correlated with the target by using mutual information or Pearson correlation.
OutliersDataCheck	Checks if there are any outliers in input data by using IQR to determine score anomalies.
NoVarianceDataCheck	Check if the target or any of the features have no variance.
ClassImbalanceDataCheck	Checks if any target labels are imbalanced beyond a threshold.
MulticollinearityDataCheck	Check if any set features are likely to be multicollinear.
DateTimeNaNDataCheck	Checks if datetime columns contain NaN values.
NaturalLanguageNaNDataCheck	Checks if natural language columns contain NaN values.

evalml.data_checks.DataCheck

class evalml.data_checks.**DataCheck**

Base class for all data checks. Data checks are a set of heuristics used to determine if there are problems with input data.

name = 'DataCheck'

Methods:

__init__	Initialize self.
validate	Inspects and validates the input data, runs any necessary calculations or algorithms, and returns a list of warnings and errors if applicable.

evalml.data_checks.DataCheck.__init__

DataCheck.**__init__**()

Initialize self. See help(type(self)) for accurate signature.

evalml.data_checks.DataCheck.validate

abstract DataCheck.**validate**(*X*, *y=None*)

Inspects and validates the input data, runs any necessary calculations or algorithms, and returns a list of warnings and errors if applicable.

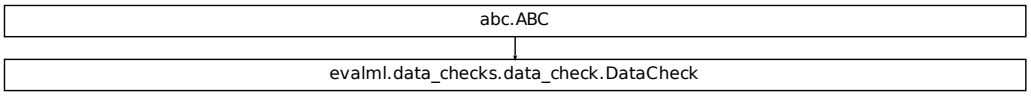
Parameters

- **X** (*pd.DataFrame*) – The input data of shape [n_samples, n_features]
- **y** (*pd.Series, optional*) – The target data of length [n_samples]

Returns Dictionary of DataCheckError and DataCheckWarning messages

Return type dict (*DataCheckMessage*)

Class Inheritance



evalml.data_checks.InvalidTargetDataCheck

class evalml.data_checks.**InvalidTargetDataCheck**(*problem_type*, *objective*, *n_unique=100*)

Checks if the target data contains missing or invalid values.

name = 'InvalidTargetDataCheck'

Methods:

__init__	Check if the target is invalid for the specified problem type.
validate	Checks if the target data contains missing or invalid values.

evalml.data_checks.InvalidTargetDataCheck.__init__

InvalidTargetDataCheck.**__init__**(*problem_type*, *objective*, *n_unique=100*)

Check if the target is invalid for the specified problem type.

Parameters

- **problem_type** (*str or ProblemTypes*) – The specific problem type to data check for. e.g. 'binary', 'multiclass', 'regression, 'time series regression'
- **objective** (*str or ObjectiveBase*) – Name or instance of the objective class.
- **n_unique** (*int*) – Number of unique target values to store when problem type is binary and target incorrectly has more than 2 unique values. Non-negative integer. Defaults to 100. If None, stores all unique values.

evalml.data_checks.InvalidTargetDataCheck.validate

InvalidTargetDataCheck.**validate**(*X*, *y*)

Checks if the target data contains missing or invalid values.

Parameters

- **X** (*pd.DataFrame, np.ndarray*) – Features. Ignored.
- **y** (*pd.Series, np.ndarray*) – Target data to check for invalid values.

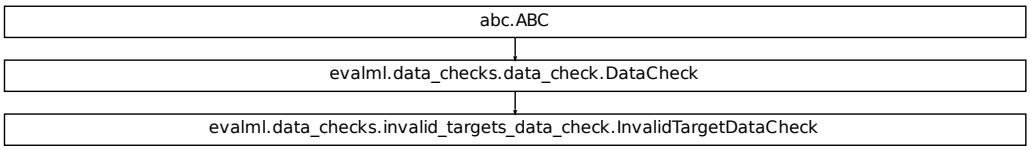
Returns List with DataCheckErrors if any invalid values are found in the target data.

Return type dict (*DataCheckError*)

Example

```
>>> import pandas as pd
>>> X = pd.DataFrame({"col": [1, 2, 3, 1]})
>>> y = pd.Series([0, 1, None, None])
>>> target_check = InvalidTargetDataCheck('binary', 'Log Loss Binary')
>>> assert target_check.validate(X, y) == {"errors": [{"message": "2 row(s)␣
↪(50.0%) of target values are null",
↪                           "data_check_name": "InvalidTargetDataCheck",␣
↪                                                "level": "error
↪",
↪ "TARGET_HAS_NULL",
↪           "details": {"num_null_rows": 2, "pct_null_rows": 50}}],
↪                                             "warnings": [],
↪                                          "actions": [{'code': 'IMPUTE_COL',
↪'metadata': {'column': None, 'impute_strategy': 'most_frequent', 'is_target
↪': True}}]}
```

Class Inheritance



evalml.data_checks.HighlyNullDataCheck

class evalml.data_checks.**HighlyNullDataCheck**(*pct_null_threshold=0.95*)

Checks if there are any highly-null columns and rows in the input.

name = 'HighlyNullDataCheck'

Methods:

__init__	Checks if there are any highly-null columns and rows in the input.
validate	Checks if there are any highly-null columns or rows in the input.

evalml.data_checks.HighlyNullDataCheck.__init__

HighlyNullDataCheck.**__init__**(*pct_null_threshold=0.95*)

Checks if there are any highly-null columns and rows in the input.

Parameters **pct_null_threshold** (*float*) – If the percentage of NaN values in an input feature exceeds this amount, that column/row will be considered highly-null. Defaults to 0.95.

evalml.data_checks.HighlyNullDataCheck.validate

HighlyNullDataCheck.**validate**(*X*, *y=None*)

Checks if there are any highly-null columns or rows in the input.

Parameters

- **X** (*pd.DataFrame, np.ndarray*) – Features.
- **y** (*pd.Series, np.ndarray*) – Ignored.

Returns dict with a DataCheckWarning if there are any highly-null columns or rows.

Return type dict

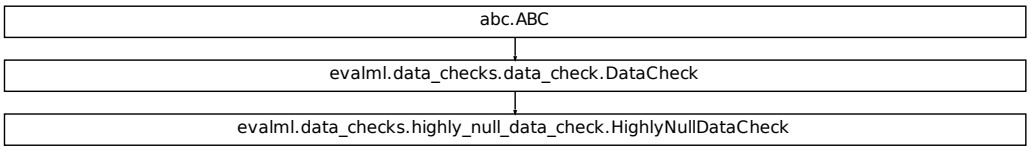
Example

```
>>> import pandas as pd
>>> class SeriesWrap():
...     def __init__(self, series):
...         self.series = series
...
...     def __eq__(self, series_2):
...         return all(self.series.eq(series_2.series))
...
>>> df = pd.DataFrame({
...     'lots_of_null': [None, None, None, None, 5],
...     'no_null': [1, 2, 3, 4, 5]
... })
>>> null_check = HighlyNullDataCheck(pct_null_threshold=0.50)
>>> validation_results = null_check.validate(df)
>>> validation_results['warnings'][0]['details']['pct_null_cols'] =␣
↪SeriesWrap(validation_results['warnings'][0]['details']['pct_null_cols'])
>>> highly_null_rows = SeriesWrap(pd.Series([0.5, 0.5, 0.5, 0.5]))
>>> assert validation_results== {"errors": [],␣
↪            "warnings": [{"message": "4 out of 5 rows are more than 50.0%␣
↪null",                                                    "data_
↪check_name": "HighlyNullDataCheck",
↪                    "level": "warning",
↪                       "code": "HIGHLY_NULL_ROWS",
↪                                  "details": {"pct_null_cols": highly_null_
↪rows}},                                                     {"message
↪": "Column 'lots_of_null' is 50.0% or more null",
↪                                  "data_check_name": "HighlyNullDataCheck",
↪                                                  "level": "warning
↪",                                                     "code":
↪"HIGHLY_NULL_COLS",
↪    "details": {"column": "lots_of_null", "pct_null_rows": 0.8}}],
↪                                  "actions": [{"code": "DROP_ROWS", "metadata
↪": {"rows": [0, 1, 2, 3]}},
↪           {"code": "DROP_COL",
↪              "metadata": {"column": "lots_of_null"}}]}
```

(continues on next page)

(continued from previous page)

Class Inheritance



evalml.data_checks.IDColumnsDataCheck

class evalml.data_checks.**IDColumnsDataCheck**(*id_threshold=1.0*)

Check if any of the features are likely to be ID columns.

name = 'IDColumnsDataCheck'

Methods:

__init__	Check if any of the features are likely to be ID columns.
validate	Check if any of the features are likely to be ID columns.

evalml.data_checks.IDColumnsDataCheck.__init__

IDColumnsDataCheck.**__init__**(*id_threshold=1.0*)

Check if any of the features are likely to be ID columns.

Parameters **id_threshold** (*float*) – The probability threshold to be considered an ID column. Defaults to 1.0.

evalml.data_checks.IDColumnsDataCheck.validate

IDColumnsDataCheck.**validate**(*X*, *y=None*)

Check if any of the features are likely to be ID columns. Currently performs these simple checks:

- column name is "id"
- column name ends in "_id"
- column contains all unique values (and is categorical / integer type)

Parameters **X** (*pd.DataFrame, np.ndarray*) – The input features to check

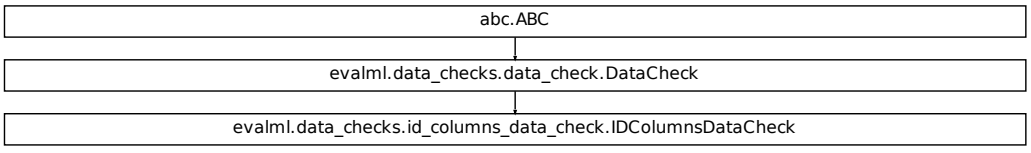
Returns A dictionary of features with column name or index and their probability of being ID columns

Return type dict

Example

```
>>> import pandas as pd
>>> df = pd.DataFrame({
...     'df_id': [0, 1, 2, 3, 4],
...     'x': [10, 42, 31, 51, 61],
...     'y': [42, 54, 12, 64, 12]
... })
>>> id_col_check = IDColumnsDataCheck()
>>> assert id_col_check.validate(df) == {"errors": [],
↪                                     "warnings": [{"message": "Column 'df_id' is␣
↪100.0% or more likely to be an ID column",
↪                                       "data_check_name": "IDColumnsDataCheck",
↪                                                          "level":
↪"warning",
↪ "code": "HAS_ID_COLUMN",
↪                    "details": {"column": "df_id"}}],
↪                              "actions": [{"code": "DROP_COL",
↪                                           "metadata": {"column": "df_id
↪"}}]}
```

Class Inheritance



evalml.data_checks.TargetLeakageDataCheck

class evalml.data_checks.**TargetLeakageDataCheck**(*pct_corr_threshold=0.95*, *method='mutual'*)

Check if any of the features are highly correlated with the target by using mutual information or Pearson correlation.

name = 'TargetLeakageDataCheck'

Methods:

__init__	Check if any of the features are highly correlated with the target by using mutual information or Pearson correlation.
validate	Check if any of the features are highly correlated with the target by using mutual information or Pearson correlation.

evalml.data_checks.TargetLeakageDataCheck.__init__

TargetLeakageDataCheck.**__init__**(*pct_corr_threshold=0.95*, *method='mutual'*)

Check if any of the features are highly correlated with the target by using mutual information or Pearson correlation.

If *method='mutual'*, this data check uses mutual information and supports all target and feature types. Otherwise, if *method='pearson'*, it uses Pearson correlation and only supports binary with numeric and boolean dtypes. Pearson correlation returns a value in [-1, 1], while mutual information returns a value in [0, 1].

Parameters

- **pct_corr_threshold** (*float*) – The correlation threshold to be considered leakage. Defaults to 0.95.
- **method** (*string*) – The method to determine correlation. Use 'mutual' for mutual information, otherwise 'pearson' for Pearson correlation. Defaults to 'mutual'.

evalml.data_checks.TargetLeakageDataCheck.validate

TargetLeakageDataCheck.**validate**(*X*, *y*)

Check if any of the features are highly correlated with the target by using mutual information or Pearson correlation.

If *method='mutual'*, supports all target and feature types. Otherwise, if *method='pearson'* only supports binary with numeric and boolean dtypes. Pearson correlation returns a value in [-1, 1], while mutual information returns a value in [0, 1].

Parameters

- **X** (*pd.DataFrame, np.ndarray*) – The input features to check
- **y** (*pd.Series, np.ndarray*) – The target data

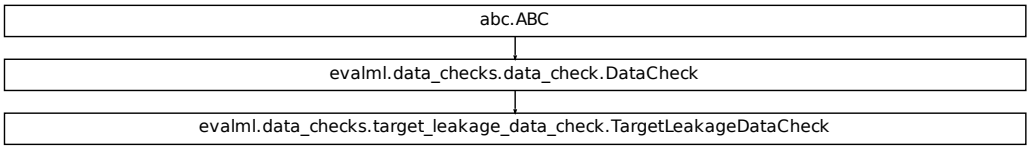
Returns dict with a DataCheckWarning if target leakage is detected.

Return type dict (*DataCheckWarning*)

Example

```
>>> import pandas as pd
>>> X = pd.DataFrame({
...     'leak': [10, 42, 31, 51, 61],
...     'x': [42, 54, 12, 64, 12],
...     'y': [13, 5, 13, 74, 24],
... })
>>> y = pd.Series([10, 42, 31, 51, 40])
>>> target_leakage_check = TargetLeakageDataCheck(pct_corr_threshold=0.95)
>>> assert target_leakage_check.validate(X, y) == {"warnings": [{"message":
↪"Column 'leak' is 95.0% or more correlated with the target",
↪                                                            "data_check_
↪name": "TargetLeakageDataCheck",
↪                                    "level": "warning",
↪                                                 "code": "TARGET_LEAKAGE
↪",
↪   "details": {"column": "leak"}}],
↪                       "errors": [],
↪                       "actions": [{"code": "DROP_COL",
↪                                                  "metadata": {"column":
↪ "leak"}}]}
```

Class Inheritance



evalml.data_checks.OutliersDataCheck

class evalml.data_checks.**OutliersDataCheck**

Checks if there are any outliers in input data by using IQR to determine score anomalies. Columns with score anomalies are considered to contain outliers.

name = 'OutliersDataCheck'

Methods:

__init__	Checks if there are any outliers in the input data.
validate	Checks if there are any outliers in a dataframe by using IQR to determine column anomalies.

evalml.data_checks.OutliersDataCheck.__init__

OutliersDataCheck.**__init__**()

Checks if there are any outliers in the input data.

evalml.data_checks.OutliersDataCheck.validate

OutliersDataCheck.**validate**(*X*, *y=None*)

Checks if there are any outliers in a dataframe by using IQR to determine column anomalies. Column with anomalies are considered to contain outliers.

Parameters

- **X** (*pd.DataFrame, np.ndarray*) – Features
- **y** (*pd.Series, np.ndarray*) – Ignored.

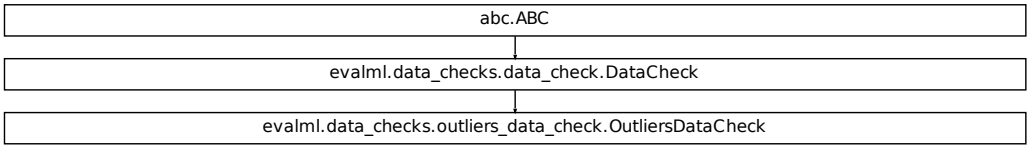
Returns A dictionary with warnings if any columns have outliers.

Return type dict

Example

```
>>> import pandas as pd
>>> df = pd.DataFrame({
...     'x': [1, 2, 3, 4, 5],
...     'y': [6, 7, 8, 9, 10],
...     'z': [-1, -2, -3, -1201, -4]
... })
>>> outliers_check = OutliersDataCheck()
>>> assert outliers_check.validate(df) == {"warnings": [{"message":
↪"Column(s) 'z' are likely to have outlier data.",
↪                                                    "data_check_name":
↪"OutliersDataCheck",
↪             "level": "warning",
↪                          "code": "HAS_OUTLIERS",
↪                                                  "details": {"columns": ["z"]}}],
↪                                                             "errors": [],
↪                                                    "actions": []}
```

Class Inheritance



evalml.data_checks.NoVarianceDataCheck

class evalml.data_checks.**NoVarianceDataCheck**(*count_nan_as_value=False*)

Check if the target or any of the features have no variance.

name = 'NoVarianceDataCheck'

Methods:

__init__	Check if the target or any of the features have no variance.
validate	Check if the target or any of the features have no variance (1 unique value).

evalml.data_checks.NoVarianceDataCheck.__init__

NoVarianceDataCheck.**__init__**(*count_nan_as_value=False*)

Check if the target or any of the features have no variance.

Parameters **count_nan_as_value** (*bool*) – If True, missing values will be counted as their own unique value. If set to True, a feature that has one unique value and all other data is missing, a DataCheckWarning will be returned instead of an error. Defaults to False.

evalml.data_checks.NoVarianceDataCheck.validate

NoVarianceDataCheck.**validate**(*X*, *y*)

Check if the target or any of the features have no variance (1 unique value).

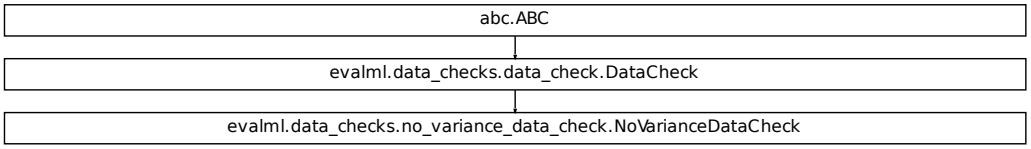
Parameters

- **X** (*pd.DataFrame, np.ndarray*) – The input features.
- **y** (*pd.Series, np.ndarray*) – The target data.

Returns dict of warnings/errors corresponding to features or target with no variance.

Return type dict

Class Inheritance



evalml.data_checks.ClassImbalanceDataCheck

class evalml.data_checks.**ClassImbalanceDataCheck**(*threshold=0.1*, *min_samples=100*, *num_cv_folds=3*)

Checks if any target labels are imbalanced beyond a threshold. Use for classification problems

name = 'ClassImbalanceDataCheck'

Methods:

__init__	Check if any of the target labels are imbalanced, or if the number of values for each target
validate	Checks if any target labels are imbalanced beyond a threshold for binary and multiclass problems

evalml.data_checks.ClassImbalanceDataCheck.__init__

ClassImbalanceDataCheck.**__init__**(*threshold=0.1*, *min_samples=100*, *num_cv_folds=3*)

Check if any of the target labels are imbalanced, or if the number of values for each target are below 2 times the number of cv folds

Parameters

- **threshold** (*float*) – The minimum threshold allowed for class imbalance before a warning is raised. This threshold is calculated by comparing the number of samples in each class to the sum of samples in that class and the majority class. For example, a multiclass case with [900, 900, 100] samples per classes 0, 1, and 2, respectively, would have a 0.10 threshold for class 2 (100 / (900 + 100)). Defaults to 0.10.
- **min_samples** (*int*) – The minimum number of samples per accepted class. If the minority class is both below the threshold and min_samples, then we consider this severely imbalanced. Must be greater than 0. Defaults to 100.
- **num_cv_folds** (*int*) – The number of cross-validation folds. Must be positive. Choose 0 to ignore this warning.

evalml.data_checks.ClassImbalanceDataCheck.validate

ClassImbalanceDataCheck.**validate**(*X*, *y*)

Checks if any target labels are imbalanced beyond a threshold for binary and multiclass problems
Ignores NaN values in target labels if they appear.

Parameters

- **X** (*pd.DataFrame, np.ndarray*) – Features. Ignored.
- **y** (*pd.Series, np.ndarray*) – Target labels to check for imbalanced data.

Returns

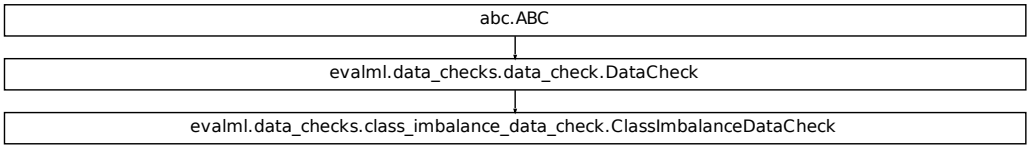
Dictionary with DataCheckWarnings if imbalance in classes is less than the threshold,
and DataCheckErrors if the number of values for each target is below 2 * num_cv_folds.

Return type dict

Example

```
>>> import pandas as pd
>>> X = pd.DataFrame()
>>> y = pd.Series([0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1])
>>> target_check = ClassImbalanceDataCheck(threshold=0.10)
>>> assert target_check.validate(X, y) == {"errors": [{"message": "The number␣
↪of instances of these targets is less than 2 * the number of cross folds =␣
↪6 instances: [0]",
↪         "data_check_name": "ClassImbalanceDataCheck",
↪                                            "level": "error",
↪                                                  "code": "CLASS_
↪IMBALANCE_BELOW_FOLDS",
↪              "details": {"target_values": [0]}}],
↪                "warnings": [{"message": "The following labels␣
↪fall below 10% of the target: [0]",
↪                          "data_check_name": "ClassImbalanceDataCheck",
↪                                                       "level":
↪"warning",
↪ "code": "CLASS_IMBALANCE_BELOW_THRESHOLD",
↪                                  "details": {"target_values": [0]}},
↪                                                     {"message":
↪"The following labels in the target have severe class imbalance because␣
↪they fall under 10% of the target and have less than 100 samples: [0]",
↪                                                       "data_check_
↪name": "ClassImbalanceDataCheck",
↪                        "level": "warning",
↪                                  "code": "CLASS_IMBALANCE_SEVERE",
↪                                                "details": {
↪"target_values": [0]}}],
↪ "actions": []}
```

Class Inheritance



evalml.data_checks.MulticollinearityDataCheck

class evalml.data_checks.**MulticollinearityDataCheck**(*threshold=0.9*)

Check if any set features are likely to be multicollinear.

name = 'MulticollinearityDataCheck'

Methods:

__init__	Check if any set of features are likely to be multicollinear.
validate	Check if any set of features are likely to be multicollinear.

evalml.data_checks.MulticollinearityDataCheck.__init__

MulticollinearityDataCheck.**__init__**(*threshold=0.9*)

Check if any set of features are likely to be multicollinear.

Parameters **threshold** (*float*) – The threshold to be considered. Defaults to 0.9.

evalml.data_checks.MulticollinearityDataCheck.validate

MulticollinearityDataCheck.**validate**(*X*, *y=None*)

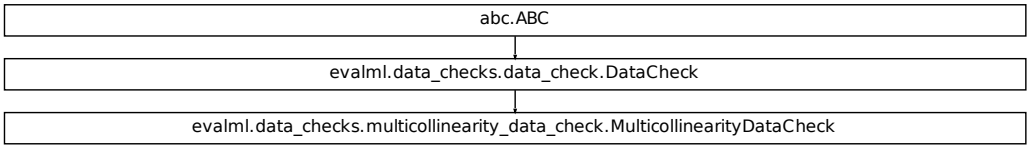
Check if any set of features are likely to be multicollinear.

Parameters **X** (*pd.DataFrame, np.ndarray*) – The input features to check

Returns dict with a DataCheckWarning if there are any potentially multicollinear columns.

Return type dict

Class Inheritance



evalml.data_checks.DateTimeNaNDataCheck

class evalml.data_checks.**DateTimeNaNDataCheck**

Checks if datetime columns contain NaN values.

name = 'DateTimeNaNDataCheck'

Methods:

__init__	Checks each column in the input for datetime features and will issue an error if NaN values are present.
validate	Checks if any datetime columns contain NaN values.

evalml.data_checks.DateTimeNaNDataCheck.__init__

DateTimeNaNDataCheck.**__init__**()

Checks each column in the input for datetime features and will issue an error if NaN values are present.

evalml.data_checks.DateTimeNaNDataCheck.validate

DateTimeNaNDataCheck.**validate**(*X*, *y=None*)

Checks if any datetime columns contain NaN values.

Parameters

- **X** (*pd.DataFrame, np.ndarray*) – Features.
- **y** (*pd.Series, np.ndarray*) – Ignored. Defaults to None.

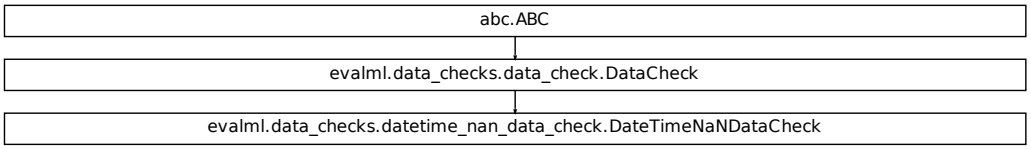
Returns dict with a DataCheckError if NaN values are present in datetime columns.

Return type dict

Example

```
>>> import pandas as pd
>>> import woodwork as ww
>>> import numpy as np
>>> dates = np.arange(np.datetime64('2017-01-01'), np.datetime64('2017-01-08
↪'))
>>> dates[0] = np.datetime64('NaT')
>>> df = pd.DataFrame(dates, columns=['index'])
>>> df.ww.init()
>>> dt_nan_check = DateTimeNaNDataCheck()
>>> assert dt_nan_check.validate(df) == {"warnings": [],
...                                      "actions": [],
...                                      "errors":␣
↪[DataCheckError(message='Input datetime column(s) (index) contains NaN␣
↪values. Please impute NaN values or drop these rows or columns.',
...                                                                  data_
↪check_name=DateTimeNaNDataCheck.name,
...                                                                  ␣
↪message_code=DataCheckMessageCode.DATETIME_HAS_NAN,
...                                                                  ␣
↪details={"columns": 'index'}).to_dict()]}
```

Class Inheritance



evalml.data_checks.NaturalLanguageNaNDataCheck

class evalml.data_checks.**NaturalLanguageNaNDataCheck**

Checks if natural language columns contain NaN values.

name = 'NaturalLanguageNaNDataCheck'

Methods:

__init__	Checks each column in the input for natural language features and will issue an error if NaN values are present.
validate	Checks if any natural language columns contain NaN values.

evalml.data_checks.NaturalLanguageNaNDataCheck.__init__

NaturalLanguageNaNDataCheck.**__init__**()

Checks each column in the input for natural language features and will issue an error if NaN values are present.

evalml.data_checks.NaturalLanguageNaNDataCheck.validate

NaturalLanguageNaNDataCheck.**validate**(*X*, *y=None*)

Checks if any natural language columns contain NaN values.

Parameters

- **X** (*pd.DataFrame, np.ndarray*) – Features.
- **y** (*pd.Series, np.ndarray*) – Ignored. Defaults to None.

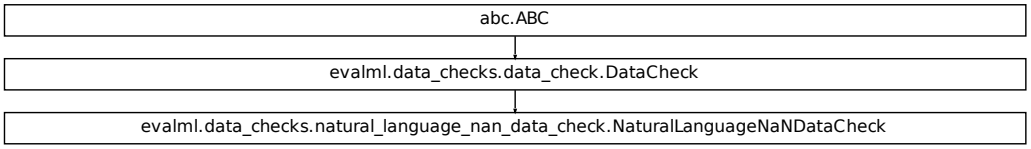
Returns dict with a DataCheckError if NaN values are present in natural language columns.

Return type dict

Example

```
>>> import pandas as pd
>>> import woodwork as ww
>>> import numpy as np
>>> data = pd.DataFrame()
>>> data['A'] = [None, "string_that_is_long_enough_for_natural_language"]
>>> data['B'] = ['string_that_is_long_enough_for_natural_language', 'string_
↪that_is_long_enough_for_natural_language']
>>> data['C'] = np.random.randint(0, 3, size=len(data))
>>> data.ww.init(logical_types={'A': 'NaturalLanguage', 'B': 'NaturalLanguage
↪'})
>>> nl_nan_check = NaturalLanguageNaNDataCheck()
>>> assert nl_nan_check.validate(data) == {
...         "warnings": [],
...         "actions": [],
...         "errors": [DataCheckError(message='Input natural language␣
↪column(s) (A) contains NaN values. Please impute NaN values or drop these␣
↪rows or columns.',
...                         data_check_name=NaturalLanguageNaNDataCheck.name,
...                         message_code=DataCheckMessageCode.NATURAL_LANGUAGE_
↪HAS_NAN,
...                         details={"columns": 'A'}).to_dict()]
...     }
```

Class Inheritance



DataChecks	A collection of data checks.
DefaultDataChecks	A collection of basic data checks that is used by AutoML by default.

evalml.data_checks.DataChecks

class evalml.data_checks.**DataChecks**(*data_checks=None*, *data_check_params=None*)

A collection of data checks.

Methods

__init__	A collection of data checks.
validate	Inspects and validates the input data against data checks and returns a list of warnings and errors if applicable.

evalml.data_checks.DataChecks.__init__

DataChecks.**__init__**(*data_checks=None*, *data_check_params=None*)

A collection of data checks.

Parameters **data_checks** (*list (DataCheck)*) – List of DataCheck objects

evalml.data_checks.DataChecks.validate

DataChecks.**validate**(*X*, *y=None*)

Inspects and validates the input data against data checks and returns a list of warnings and errors if applicable.

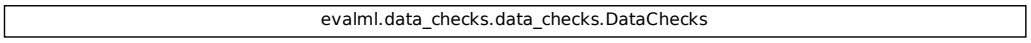
Parameters

- **X** (*pd.DataFrame, np.ndarray*) – The input data of shape [n_samples, n_features]
- **y** (*pd.Series, np.ndarray*) – The target data of length [n_samples]

Returns Dictionary containing DataCheckMessage objects

Return type dict

Class Inheritance



evalml.data_checks.DefaultDataChecks

class evalml.data_checks.**DefaultDataChecks**(*problem_type*, *objective*, *n_splits=3*)

A collection of basic data checks that is used by AutoML by default. Includes:

- *HighlyNullDataCheck*
- *HighlyNullRowsDataCheck*
- *IDColumnsDataCheck*
- *TargetLeakageDataCheck*
- *InvalidTargetDataCheck*
- *NoVarianceDataCheck*
- *ClassImbalanceDataCheck* (for classification problem types)
- *DateTimeNaNDataCheck*
- *NaturalLanguageNaNDataCheck*

Methods

__init__	A collection of basic data checks.
validate	Inspects and validates the input data against data checks and returns a list of warnings and errors if applicable.

evalml.data_checks.DefaultDataChecks.__init__

DefaultDataChecks.**__init__**(*problem_type*, *objective*, *n_splits=3*)

A collection of basic data checks.

Parameters

- **problem_type** (*str*) – The problem type that is being validated. Can be regression, binary, or multiclass.
- **objective** (*str or ObjectiveBase*) – Name or instance of the objective class.

- **n_splits** (*int*) – The number of splits as determined by the data splitter being used.

evalml.data_checks.DefaultDataChecks.validate

DefaultDataChecks.**validate**(*X*, *y=None*)

Inspects and validates the input data against data checks and returns a list of warnings and errors if applicable.

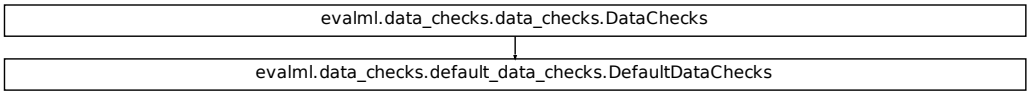
Parameters

- **X** (*pd.DataFrame, np.ndarray*) – The input data of shape [n_samples, n_features]
- **y** (*pd.Series, np.ndarray*) – The target data of length [n_samples]

Returns Dictionary containing DataCheckMessage objects

Return type dict

Class Inheritance



5.12.2 Data Check Messages

DataCheckMessage	Base class for all DataCheckMessages.
DataCheckError	DataCheckMessage subclass for errors returned by data checks.
DataCheckWarning	DataCheckMessage subclass for warnings returned by data checks.

evalml.data_checks.DataCheckMessage

class evalml.data_checks.**DataCheckMessage**(*message*, *data_check_name*, *message_code=None*, *details=None*)

Base class for all DataCheckMessages.

message_type = None

Methods:

__init__	Message returned by a DataCheck, tagged by name.
to_dict	
__str__	String representation of data check message, equivalent to self.message attribute.
__eq__	Checks for equality.

evalml.data_checks.DataCheckMessage.__init__

DataCheckMessage.**__init__**(*message*, *data_check_name*, *message_code=None*, *details=None*)

Message returned by a DataCheck, tagged by name.

Parameters

- **message** (*str*) – Message string
- **data_check_name** (*str*) – Name of data check
- **message_code** (DataCheckMessageCode, *optional*) – Message code associated with message.
- **details** (*dict, optional*) – Additional useful information associated with the message

evalml.data_checks.DataCheckMessage.to_dict

DataCheckMessage.**to_dict**()

evalml.data_checks.DataCheckMessage.__str__

DataCheckMessage.**__str__**()

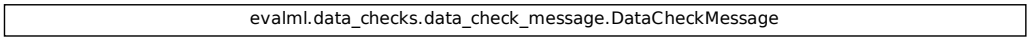
String representation of data check message, equivalent to self.message attribute.

evalml.data_checks.DataCheckMessage.__eq__

DataCheckMessage.**__eq__**(*other*)

Checks for equality. Two DataCheckMessage objs are considered equivalent if all of their attributes are equivalent.

Class Inheritance



evalml.data_checks.DataCheckError

class evalml.data_checks.**DataCheckError**(*message*, *data_check_name*, *message_code=None*, *details=None*)

DataCheckMessage subclass for errors returned by data checks.

message_type = 'error'

Methods:

__init__	Message returned by a DataCheck, tagged by name.
to_dict	
__str__	String representation of data check message, equivalent to self.message attribute.
__eq__	Checks for equality.

evalml.data_checks.DataCheckError.__init__

DataCheckError.**__init__**(*message*, *data_check_name*, *message_code=None*, *details=None*)

Message returned by a DataCheck, tagged by name.

Parameters

- **message** (*str*) – Message string
- **data_check_name** (*str*) – Name of data check
- **message_code** (DataCheckMessageCode, *optional*) – Message code associated with message.
- **details** (*dict, optional*) – Additional useful information associated with the message

evalml.data_checks.DataCheckError.to_dict

DataCheckError.**to_dict**()

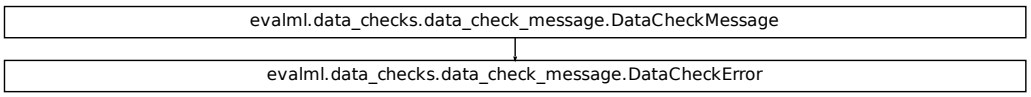
evalml.data_checks.DataCheckError.__str__

DataCheckError.**__str__**()
String representation of data check message, equivalent to self.message attribute.

evalml.data_checks.DataCheckError.__eq__

DataCheckError.**__eq__**(*other*)
Checks for equality. Two DataCheckMessage objs are considered equivalent if all of their attributes are equivalent.

Class Inheritance



evalml.data_checks.DataCheckWarning

class evalml.data_checks.**DataCheckWarning**(*message*, *data_check_name*, *message_code=None*, *details=None*)
DataCheckMessage subclass for warnings returned by data checks.

message_type = 'warning'

Methods:

__init__	Message returned by a DataCheck, tagged by name.
to_dict	
__str__	String representation of data check message, equivalent to self.message attribute.
__eq__	Checks for equality.

evalml.data_checks.DataCheckWarning.__init__

DataCheckWarning.**__init__**(*message*, *data_check_name*, *message_code=None*, *details=None*)

Message returned by a DataCheck, tagged by name.

Parameters

- **message** (*str*) – Message string
- **data_check_name** (*str*) – Name of data check
- **message_code** (DataCheckMessageCode, *optional*) – Message code associated with message.
- **details** (*dict, optional*) – Additional useful information associated with the message

evalml.data_checks.DataCheckWarning.to_dict

DataCheckWarning.**to_dict**()

evalml.data_checks.DataCheckWarning.__str__

DataCheckWarning.**__str__**()

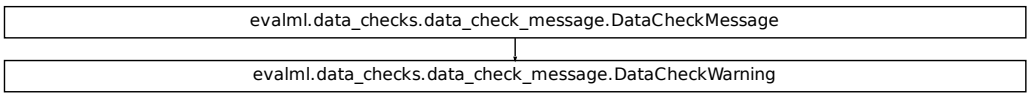
String representation of data check message, equivalent to self.message attribute.

evalml.data_checks.DataCheckWarning.__eq__

DataCheckWarning.**__eq__**(*other*)

Checks for equality. Two DataCheckMessage objs are considered equivalent if all of their attributes are equivalent.

Class Inheritance



5.12.3 Data Check Message Types

DataCheckMessageType	Enum for type of data check message: WARNING or ERROR.

evalml.data_checks.DataCheckMessageType

class evalml.data_checks.**DataCheckMessageType**(*value*)
Enum for type of data check message: WARNING or ERROR.

Attributes

ERROR	Error message returned by a data check.
WARNING	Warning message returned by a data check.

5.12.4 Data Check Message Codes

DataCheckMessageCode	Enum for data check message code.

evalml.data_checks.DataCheckMessageCode

class evalml.data_checks.**DataCheckMessageCode**(*value*)
Enum for data check message code.

Attributes

CLASS_IMBALANCE_BELOW_FOLDS	Message code for when the number of values for each target is below 2 * number of CV folds.
CLASS_IMBALANCE_BELOW_THRESHOLD	Message code for when balance in classes is less than the threshold.
CLASS_IMBALANCE_SEVERE	Message code for when balance in classes is less than the threshold and minimum class is less than minimum number of accepted samples.
DATETIME_HAS_NAN	Message code for when input datetime columns contain NaN values.
HAS_ID_COLUMN	Message code for data that has ID columns.
HAS_OUTLIERS	Message code for when outliers are detected.
HIGHLY_NULL_COLS	Message code for highly null columns.
HIGHLY_NULL_ROWS	Message code for highly null rows.
HIGH_VARIANCE	Message code for when high variance is detected for cross-validation.
IS_MULTICOLLINEAR	Message code for when data is potentially multicollinear.
MISMATCHED_INDICES	Message code for when input target and features have mismatched indices.

continues on next page

Table 212 – continued from previous page

MISMATCHED_INDICES_ORDER	Message code for when input target and features have mismatched indices order.
MISMATCHED_LENGTHS	Message code for when input target and features have different lengths.
NATURAL_LANGUAGE_HAS_NAN	Message code for when input natural language columns contain NaN values.
NOT_UNIQUE_ENOUGH	Message code for when data does not possess enough unique values.
NO_VARIANCE	Message code for when data has no variance (1 unique value).
NO_VARIANCE_WITH_NULL	Message code for when data has one unique value and NaN values.
TARGET_BINARY_INVALID_VALUES	Message code for target data for a binary classification problem with numerical values not equal to {0, 1}.
TARGET_BINARY_NOT_TWO_UNIQUE_VALUES	Message code for target data for a binary classification problem that does not have two unique values.
TARGET_HAS_NULL	Message code for target data that has null values.
TARGET_INCOMPATIBLE_OBJECTIVE	Message code for target data that has incompatible values for the specified objective
TARGET_IS_EMPTY_OR_FULLY_NULL	Message code for target data that is empty or has all null values.
TARGET_IS_NONE	Message code for when target is None.
TARGET_LEAKAGE	Message code for when target leakage is detected.
TARGET_MULTICLASS_HIGH_UNIQUE_CLASS	Message code for target data for a multi classification problem that has an abnormally large number of unique classes relative to the number of target values.
TARGET_MULTICLASS_NOT_ENOUGH_CLASSES	Message code for target data for a multi classification problem that does not have more than two unique classes.
TARGET_MULTICLASS_NOT_TWO_EXAMPLES_PER_CLASS	Message code for target data for a multi classification problem that does not have two examples per class.
TARGET_UNSUPPORTED_TYPE	Message code for target data that is of an unsupported type.
TOO_SPARSE	Message code for when multiclass data has values that are too sparsely populated.
TOO_UNIQUE	Message code for when data possesses too many unique values.

5.13 Utils

5.13.1 General Utils

import_or_raise	Attempts to import the requested library by name.
convert_to_seconds	Converts a string describing a length of time to its length in seconds.
get_random_state	Generates a numpy.random.RandomState instance using seed.

continues on next page

Table 213 – continued from previous page

get_random_seed	Given a numpy.random.RandomState object, generate an int representing a seed value for another random number generator.
pad_with_nans	Pad the beginning num_to_pad rows with nans.
drop_rows_with_nans	Drop rows that have any NaNs in all dataframes or series.
infer_feature_types	Create a Woodwork structure from the given list, pandas, or numpy input, with specified types for columns.
save_plot	Saves fig to filepath if specified, or to a default location if not.
is_all_numeric	Checks if the given DataFrame contains only numeric values
get_importable_subclasses	Get importable subclasses of a base class.

evalml.utils.import_or_raise

evalml.utils.**import_or_raise**(*library*, *error_msg=None*, *warning=False*)

Attempts to import the requested library by name. If the import fails, raises an ImportError or warning.

Parameters

- **library** (*str*) – the name of the library
- **error_msg** (*str*) – error message to return if the import fails
- **warning** (*bool*) – if True, import_or_raise gives a warning instead of ImportError. Defaults to False.

evalml.utils.convert_to_seconds

evalml.utils.**convert_to_seconds**(*input_str*)

Converts a string describing a length of time to its length in seconds.

evalml.utils.get_random_state

evalml.utils.**get_random_state**(*seed*)

Generates a numpy.random.RandomState instance using seed.

Parameters **seed** (*None, int, np.random.RandomState object*) – seed to use to generate numpy.random.RandomState. Must be between SEED_BOUNDS.min_bound and SEED_BOUNDS.max_bound, inclusive. Otherwise, an exception will be thrown.

evalml.utils.get_random_seed

evalml.utils.**get_random_seed**(*random_state*, *min_bound=0*, *max_bound=2147483647*)

Given a numpy.random.RandomState object, generate an int representing a seed value for another random number generator. Or, if given an int, return that int.

To protect against invalid input to a particular library's random number generator, if an int value is provided, and it is outside the bounds "[min_bound, max_bound)", the value will be projected into the range between the min_bound (inclusive) and max_bound (exclusive) using modular arithmetic.

Parameters

- **random_state** (*int, numpy.random.RandomState*) – random state
- **min_bound** (*None, int*) – if not default of None, will be min bound when generating seed (inclusive). Must be less than max_bound.
- **max_bound** (*None, int*) – if not default of None, will be max bound when generating seed (exclusive). Must be greater than min_bound.

Returns seed for random number generator

Return type int

evalml.utils.pad_with_nans

evalml.utils.**pad_with_nans**(*pd_data*, *num_to_pad*)

Pad the beginning num_to_pad rows with nans.

Parameters **pd_data** (*pd.DataFrame or pd.Series*) – Data to pad.

Returns pd.DataFrame or pd.Series

evalml.utils.drop_rows_with_nans

evalml.utils.**drop_rows_with_nans**(**pd_data*)

Drop rows that have any NaNs in all dataframes or series.

Parameters ***pd_data** (*sequence of pd.Series or pd.DataFrame or None*) –

Returns list of pd.DataFrame or pd.Series or None

evalml.utils.infer_feature_types

evalml.utils.**infer_feature_types**(*data*, *feature_types=None*)

Create a Woodwork structure from the given list, pandas, or numpy input, with specified types for columns. If a column's type is not specified, it will be inferred by Woodwork.

Parameters

- **data** (*pd.DataFrame, pd.Series*) – Input data to convert to a Woodwork data structure.
- **feature_types** (*string, ww.logical_type obj, dict, optional*) – If data is a 2D structure, feature_types must be a dictionary mapping column names to the type of data represented in the column. If data is a 1D structure, then feature_types must be a Woodwork logical type or a string representing a Woodwork logical type ("Double", "Integer", "Boolean", "Categorical", "Datetime", "NaturalLanguage")

Returns A Woodwork data structure where the data type of each column was either specified or inferred.

evalml.utils.save_plot

evalml.utils.**save_plot**(*fig*, *filepath=None*, *format='png'*, *interactive=False*, *return_filepath=False*)
Saves fig to filepath if specified, or to a default location if not.

Parameters

- **fig** (*Figure*) – Figure to be saved.
- **filepath** (*str or Path, optional*) – Location to save file. Default is with filename "test_plot".
- **format** (*str*) – Extension for figure to be saved as. Ignored if interactive is True and fig
- **of type plotly.Figure. Defaults to 'png'.** (*is*) –
- **interactive** (*bool, optional*) – If True and fig is of type plotly.Figure, saves the fig as interactive
- **of static** (*instead*) –
- **format will be set to 'html'. Defaults to False.** (*and*) –
- **return_filepath** (*bool, optional*) – Whether to return the final filepath the image is saved to. Defaults to False.

Returns String representing the final filepath the image was saved to if return_filepath is set to True. Defaults to None.

evalml.utils.is_all_numeric

evalml.utils.**is_all_numeric**(*df*)
Checks if the given DataFrame contains only numeric values

Parameters **df** (*pd.DataFrame*) – The DataFrame to check data types of.

Returns True if all the columns are numeric and are not missing any values, False otherwise.

evalml.utils.get_importable_subclasses

evalml.utils.**get_importable_subclasses**(*base_class*, *used_in_automl=True*)
Get importable subclasses of a base class. Used to list all of our estimators, transformers, components and pipelines dynamically.

Parameters

- **base_class** (*abc.ABCMeta*) – Base class to find all of the subclasses for.
- **args** (*list*) – Args used to instantiate the subclass. [{}] for a pipeline, and [] for all other classes.
- **used_in_automl** – Not all components/pipelines/estimators are used in automl search. If True, only include those subclasses that are used in the search. This would mean excluding classes related to ExtraTrees, ElasticNet, and Baseline estimators.

Returns List of subclasses.

CHAPTER

SIX

RELEASE NOTES

Future Release

- Enhancements
- Fixes
- Changes
- Documentation Changes
- Testing Changes

> **Warning: Breaking Changes**

v0.28.0 Jul. 2, 2021

- **Enhancements**
  - Added support for showing a Individual Conditional Expectations plot when graphing Partial Dependence #2386
  - Exposed `thread_count` for Catboost estimators as `n_jobs` parameter #2410
  - Updated Objectives API to allow for sample weighting #2433
- **Fixes**
  - Deleted unreachable line from `IterativeAlgorithm` #2464
- **Changes**
  - Pinned Woodwork version between 0.4.1 and 0.4.2 #2460
  - Updated psutils minimum version in requirements #2438
  - Updated `log_error_callback` to not include filepath in logged message #2429
- **Documentation Changes**
  - Sped up docs #2430
  - Removed mentions of `DataTable` and `DataColumn` from the docs #2445
- **Testing Changes**
  - Added slack integration for nightlies tests #2436
  - Changed `build_conda_pkg` CI job to run only when dependencies are updates #2446
  - Updated workflows to store pytest runtimes as test artifacts #2448
  - Added `AutoMLTestEnv` test fixture for making it easy to mock automl tests #2406

v0.27.0 Jun. 22, 2021

- **Enhancements**
  - Adds force plots for prediction explanations #2157
  - Removed self-reference from `AutoMLSearch` #2304
  - Added support for nonlinear pipelines for `generate_pipeline_code` #2332
  - Added `inverse_transform` method to pipelines #2256
  - Add optional automatic update checker #2350
  - Added `search_order` to `AutoMLSearch`'s `rankings` and `full_rankings` tables #2345
  - Updated threshold optimization method for binary classification #2315
  - Updated demos to pull data from S3 instead of including demo data in package #2387
  - Upgrade woodwork version to v0.4.1 #2379
- **Fixes**
  - Preserve user-specified woodwork types throughout pipeline fit/predict #2297
  - Fixed `ComponentGraph` appending target to `final_component_features` if there is a component that returns both X and y #2358
  - Fixed partial dependence graph method failing on multiclass problems when the class labels are numeric #2372
  - Added `thresholding_objective` argument to `AutoMLSearch` for binary classification problems #2320
  - Added change for `k_neighbors` parameter in SMOTE Oversamplers to automatically handle small samples #2375
  - Changed naming for `Logistic Regression Classifier` file #2399
  - Pinned pytest-timeout to fix minimum dependence checker #2425
  - Replaced `Elastic Net Classifier` base class with `Logistsic Regression` to avoid `NaN` outputs #2420
- **Changes**
  - Cleaned up `PipelineBase`'s `component_graph` and `_component_graph` attributes. Updated `PipelineBase` `__repr__` and added `__eq__` for `ComponentGraph` #2332
  - Added and applied `black` linting package to the EvalML repo in place of `autopep8` #2306
  - Separated *custom_hyperparameters* from pipelines and added them as an argument to `AutoMLSearch` #2317
  - Replaced *allowed_pipelines* with *allowed_component_graphs* #2364
  - Removed private method `_compute_features_during_fit` from `PipelineBase` #2359
  - Updated `compute_order` in `ComponentGraph` to be a read-only property #2408
  - Unpinned PyZMQ version in requirements.txt #2389
  - Uncapping LightGBM version in requirements.txt #2405
  - Updated minimum version of plotly #2415

  - Removed `SensitivityLowAlert` objective from core objectives #2418

- **Documentation Changes**
  - Fixed lead scoring weights in the demos documentation #2315
  - Fixed start page code and description dataset naming discrepancy #2370
- **Testing Changes**
  - Update minimum unit tests to run on all pull requests #2314
  - Pass token to authorize uploading of codecov reports #2344
  - Add `pytest-timeout`. All tests that run longer than 6 minutes will fail. #2374
  - Separated the dask tests out into separate github action jobs to isolate dask failures. #2376
  - Refactored dask tests #2377
  - Added the combined dask/non-dask unit tests back and renamed the dask only unit tests. #2382
  - Sped up unit tests and split into separate jobs #2365
  - Change CI job names, run lint for python 3.9, run nightlies on python 3.8 at 3am EST #2395 #2398
  - Set fail-fast to false for CI jobs that run for PRs #2402

Warning:

Breaking Changes

- *AutoMLSearch* will accept *allowed_component_graphs* instead of *allowed_pipelines* #2364
- Removed `PipelineBase`'s `_component_graph` attribute. Updated `PipelineBase` `__repr__` and added `__eq__` for `ComponentGraph` #2332
- *pipeline_parameters* will no longer accept *skopt.space* variables since hyperparameter ranges will now be specified through *custom_hyperparameters* #2317

v0.25.0 Jun. 01, 2021

- **Enhancements**
  - Upgraded minimum woodwork to version 0.3.1. Previous versions will not be supported #2181
  - Added a new callback parameter for `explain_predictions_best_worst` #2308
- Fixes
- **Changes**
  - Deleted the `return_pandas` flag from our demo data loaders #2181
  - Moved `default_parameters` to `ComponentGraph` from `PipelineBase` #2307
- **Documentation Changes**
  - Updated the release procedure documentation #2230
- **Testing Changes**
  - Ignoring `test_saving_png_file` while building conda package #2323

Warning:

Breaking Changes

- Deleted the `return_pandas` flag from our demo data loaders #2181
- Upgraded minimum woodwork to version 0.3.1. Previous versions will not be supported #2181
- Due to the weak-ref in woodwork, set the result of `infer_feature_types` to a variable before accessing woodwork #2181

v0.24.2 May. 24, 2021

- **Enhancements**
  - Added oversamplers to AutoMLSearch #2213 #2286
  - Added dictionary input functionality for `Undersampler` component #2271
  - Changed the default parameter values for `Elastic Net Classifier` and `Elastic Net Regressor` #2269
  - Added dictionary input functionality for the Oversampler components #2288
- **Fixes**
  - Set default *n_jobs* to 1 for *StackedEnsembleClassifier* and *StackedEnsembleRegressor* until fix for text-based parallelism in sklearn stacking can be found #2295
- **Changes**
  - Updated `start_iteration_callback` to accept a pipeline instance instead of a pipeline class and no longer accept pipeline parameters as a parameter #2290
  - Refactored `calculate_permutation_importance` method and add per-column permutation importance method #2302
  - Updated logging information in `AutoMLSearch.__init__` to clarify pipeline generation #2263
- **Documentation Changes**
  - Minor changes to the release procedure #2230
- **Testing Changes**
  - Use codecov action to update coverage reports #2238
  - Removed MarkupSafe dependency version pin from requirements.txt and moved instead into RTD docs build CI #2261

Warning:

Breaking Changes

- Updated `start_iteration_callback` to accept a pipeline instance instead of a pipeline class and no longer accept pipeline parameters as a parameter #2290
- Moved `default_parameters` to `ComponentGraph` from `PipelineBase`. A pipeline's `default_parameters` is now accessible via `pipeline.component_graph.default_parameters` #2307

v0.24.1 May. 16, 2021

- **Enhancements**
  - Integrated `ARIMARegressor` into AutoML #2009
  - Updated `HighlyNullDataCheck` to also perform a null row check #2222
  - Set `max_depth` to 1 in calls to featuretools dfs #2231
- **Fixes**
  - Removed data splitter sampler calls during training #2253
  - Set minimum required version for for pyzmq, colorama, and docutils #2254
  - Changed BaseSampler to return None instead of y #2272
- **Changes**
  - Removed ensemble split and indices in `AutoMLSearch` #2260
  - Updated pipeline `repr()` and `generate_pipeline_code` to return pipeline instances without generating custom pipeline class #2227
- **Documentation Changes**
  - Capped Sphinx version under 4.0.0 #2244
- **Testing Changes**
  - Change number of cores for pytest from 4 to 2 #2266
  - Add minimum dependency checker to generate minimum requirement files #2267
  - Add unit tests with minimum dependencies #2277

v0.24.0 May. 04, 2021

- **Enhancements**
  - Added *date_index* as a required parameter for TimeSeries problems #2217
  - Have the `OneHotEncoder` return the transformed columns as booleans rather than floats #2170
  - Added Oversampler transformer component to EvalML #2079
  - Added Undersampler to AutoMLSearch, as well as arguments `_sampler_method` and `sampler_balanced_ratio` #2128
  - Updated prediction explanations functions to allow pipelines with XGBoost estimators #2162
  - Added partial dependence for datetime columns #2180
  - Update precision-recall curve with positive label index argument, and fix for 2d predicted probabilities #2090
  - Add pct_null_rows to `HighlyNullDataCheck` #2211
  - Added a standalone AutoML *search* method for convenience, which runs data checks and then runs automl #2152
  - Make the first batch of AutoML have a predefined order, with linear models first and complex models last #2223 #2225
  - Added sampling dictionary support to `BalancedClassficationSampler` #2235
- **Fixes**
  - Fixed partial dependence not respecting grid resolution parameter for numerical features #2180
  - Enable prediction explanations for catboost for multiclass problems #2224

- **Changes**
  - Deleted baseline pipeline classes #2202
  - Reverting user specified date feature PR #2155 until *pmdarima* installation fix is found #2214
  - Updated pipeline API to accept component graph and other class attributes as instance parameters. Old pipeline API still works but will not be supported long-term. #2091
  - Removed all old datasplitters from EvalML #2193
  - Deleted `make_pipeline_from_components` #2218
- **Documentation Changes**
  - Renamed dataset to clarify that its gzipped but not a tarball #2183
  - Updated documentation to use pipeline instances instead of pipeline subclasses #2195
  - Updated contributing guide with a note about GitHub Actions permissions #2090
  - Updated automl and model understanding user guides #2090
- **Testing Changes**
  - Use machineFL user token for dependency update bot, and add more reviewers #2189

Warning:

Breaking Changes

- All baseline pipeline classes (`BaselineBinaryPipeline`, `BaselineMulticlassPipeline`, `BaselineRegressionPipeline`, etc.) have been deleted #2202
- Updated pipeline API to accept component graph and other class attributes as instance parameters. Old pipeline API still works but will not be supported long-term. Pipelines can now be initialized by specifying the component graph as the first parameter, and then passing in optional arguments such as `custom_name`, `parameters`, etc. For example, `BinaryClassificationPipeline([ "Random Forest Classifier"], parameters={}).` #2091
- Removed all old datasplitters from EvalML #2193
- Deleted utility method `make_pipeline_from_components` #2218

v0.23.0 Apr. 20, 2021

- **Enhancements**
  - Refactored `EngineBase` and `SequentialEngine` api. Adding `DaskEngine` #1975.
  - Added optional `engine` argument to `AutoMLSearch` #1975
  - Added a warning about how time series support is still in beta when a user passes in a time series problem to `AutoMLSearch` #2118
  - Added `NaturalLanguageNaNDataCheck` data check #2122
  - Added ValueError to `partial_dependence` to prevent users from computing partial dependence on columns with all NaNs #2120
  - Added standard deviation of cv scores to rankings table #2154
- **Fixes**

  - Fixed `BalancedClassificationDataCVSplit`, `BalancedClassificationDataTVSplit`, and `BalancedClassificationSampler` to use `minority:majority` ratio instead of `majority:minority` #2077
  - Fixed bug where two-way partial dependence plots with categorical variables were not working correctly #2117
  - Fixed bug where `hyperparameters` were not displaying properly for pipelines with a list `component_graph` and duplicate components #2133
  - Fixed bug where `pipeline_parameters` argument in `AutoMLSearch` was not applied to pipelines passed in as `allowed_pipelines` #2133
  - Fixed bug where `AutoMLSearch` was not applying custom hyperparameters to pipelines with a list `component_graph` and duplicate components #2133

- **Changes**
  - Removed `hyperparameter_ranges` from Undersampler and renamed `balanced_ratio` to `sampling_ratio` for samplers #2113
  - Renamed `TARGET_BINARY_NOT_TWO_EXAMPLES_PER_CLASS` data check message code to `TARGET_MULTICLASS_NOT_TWO_EXAMPLES_PER_CLASS` #2126
  - Modified one-way partial dependence plots of categorical features to display data with a bar plot #2117
  - Renamed `score` column for `automl.rankings` as `mean_cv_score` #2135
  - Remove 'warning' from docs tool output #2031
- **Documentation Changes**
  - Fixed `conf.py` file #2112
  - Added a sentence to the automl user guide stating that our support for time series problems is still in beta. #2118
  - Fixed documentation demos #2139
  - Update test badge in README to use GitHub Actions #2150
- **Testing Changes**
  - Fixed `test_describe_pipeline` for `pandas v1.2.4` #2129
  - Added a GitHub Action for building the conda package #1870 #2148

Warning:

Breaking Changes

- Renamed `balanced_ratio` to `sampling_ratio` for the `BalancedClassificationDataCVSplit`, `BalancedClassificationDataTVSplit`, `BalancedClassficationSampler`, and Undersampler #2113
- Deleted the "errors" key from automl results #1975
- Deleted the `raise_and_save_error_callback` and the `log_and_save_error_callback` #1975
- Fixed `BalancedClassificationDataCVSplit`, `BalancedClassificationDataTVSplit`, and `BalancedClassificationSampler` to use minority:majority ratio instead of majority:minority #2077

v0.22.0 Apr. 06, 2021

- **Enhancements**
  - Added a GitHub Action for `linux_unit_tests`#2013
  - Added recommended actions for `InvalidTargetDataCheck`, updated `_make_component_list_from_actions` to address new action, and added `TargetImputer` component #1989
  - Updated `AutoMLSearch._check_for_high_variance` to not emit `RuntimeWarning` #2024
  - Added exception when pipeline passed to `explain_predictions` is a `Stacked Ensemble` pipeline #2033
  - Added sensitivity at low alert rates as an objective #2001
  - Added `Undersampler` transformer component #2030
- **Fixes**
  - Updated Engine's `train_batch` to apply undersampling #2038
  - Fixed bug in where Time Series Classification pipelines were not encoding targets in `predict` and `predict_proba` #2040
  - Fixed data splitting errors if target is float for classification problems #2050
  - Pinned `docutils` to <0.17 to fix ReadtheDocs warning issues #2088
- **Changes**
  - Removed lists as acceptable hyperparameter ranges in `AutoMLSearch` #2028
  - Renamed "details" to "metadata" for data check actions #2008
- **Documentation Changes**
  - Catch and suppress warnings in documentation #1991 #2097
  - Change spacing in `start.ipynb` to provide clarity for `AutoMLSearch` #2078
  - Fixed start code on README #2108
- Testing Changes

v0.21.0 Mar. 24, 2021

- **Enhancements**
  - Changed `AutoMLSearch` to default `optimize_thresholds` to True #1943
  - Added multiple oversampling and undersampling sampling methods as data splitters for imbalanced classification #1775
  - Added params to balanced classification data splitters for visibility #1966
  - Updated `make_pipeline` to not add `Imputer` if input data does not have numeric or categorical columns #1967
  - Updated `ClassImbalanceDataCheck` to better handle multiclass imbalances #1986
  - Added recommended actions for the output of data check's `validate` method #1968
  - Added error message for `partial_dependence` when features are mostly the same value #1994

  - Updated `OneHotEncoder` to drop one redundant feature by default for features with two categories #1997
  - Added a `PolynomialDetrender` component #1992
  - Added `DateTimeNaNDataCheck` data check #2039
- **Fixes**
  - Changed best pipeline to train on the entire dataset rather than just ensemble indices for ensemble problems #2037
  - Updated binary classification pipelines to use objective decision function during scoring of custom objectives #1934
- **Changes**
  - Removed `data_checks` parameter, `data_check_results` and data checks logic from `AutoMLSearch` #1935
  - Deleted `random_state` argument #1985
  - Updated Woodwork version requirement to `v0.0.11` #1996
- Documentation Changes
- **Testing Changes**
  - Removed `build_docs` CI job in favor of RTD GH builder #1974
  - Added tests to confirm support for Python 3.9 #1724
  - Added tests to support Dask AutoML/Engine #1990
  - Changed `build_conda_pkg` job to use `latest_release_changes` branch in the feedstock. #1979

Warning:

Breaking Changes

- Changed `AutoMLSearch` to default `optimize_thresholds` to True #1943
- Removed `data_checks` parameter, `data_check_results` and data checks logic from `AutoMLSearch`. To run the data checks which were previously run by default in `AutoMLSearch`, please call `DefaultDataChecks().validate(X_train, y_train)` or take a look at our documentation for more examples. #1935
- Deleted `random_state` argument #1985

v0.20.0 Mar. 10, 2021

- **Enhancements**
  - Added a GitHub Action for Detecting dependency changes #1933
  - Create a separate CV split to train stacked ensembler on for AutoMLSearch #1814
  - Added a GitHub Action for Linux unit tests #1846
  - Added `ARIMARegressor` estimator #1894
  - Added `DataCheckAction` class and `DataCheckActionCode` enum #1896
  - Updated `Woodwork` requirement to `v0.0.10` #1900

  - Added `BalancedClassificationDataCVSplit` and `BalancedClassificationDataTVSplit` to AutoMLSearch #1875
  - Update default classification data splitter to use downsampling for highly imbalanced data #1875
  - Updated `describe_pipeline` to return more information, including `id` of pipelines used for ensemble models #1909
  - Added utility method to create list of components from a list of `DataCheckAction` #1907
  - Updated `validate` method to include a `action` key in returned dictionary for all `DataCheck``and ``DataChecks` #1916
  - Aggregating the shap values for predictions that we know the provenance of, e.g. OHE, text, and date-time. #1901
  - Improved error message when custom objective is passed as a string in `pipeline.score` #1941
  - Added `score_pipelines` and `train_pipelines` methods to `AutoMLSearch` #1913
  - Added support for `pandas` version 1.2.0 #1708
  - Added `score_batch` and `train_batch` abstact methods to `EngineBase` and implementations in `SequentialEngine` #1913
  - Added ability to handle index columns in `AutoMLSearch` and `DataChecks` #2138
- **Fixes**
  - Removed CI check for `check_dependencies_updated_linux` #1950
  - Added metaclass for time series pipelines and fix binary classification pipeline `predict` not using objective if it is passed as a named argument #1874
  - Fixed stack trace in prediction explanation functions caused by mixed string/numeric pandas column names #1871
  - Fixed stack trace caused by passing pipelines with duplicate names to `AutoMLSearch` #1932
  - Fixed `AutoMLSearch.get_pipelines` returning pipelines with the same attributes #1958
- **Changes**
  - Reversed GitHub Action for Linux unit tests until a fix for report generation is found #1920
  - Updated `add_results` in `AutoMLAlgorithm` to take in entire pipeline results dictionary from `AutoMLSearch` #1891
  - Updated `ClassImbalanceDataCheck` to look for severe class imbalance scenarios #1905
  - Deleted the `explain_prediction` function #1915
  - Removed `HighVarianceCVDataCheck` and convered it to an `AutoMLSearch` method instead #1928
  - Removed warning in `InvalidTargetDataCheck` returned when numeric binary classification targets are not (0, 1) #1959
- **Documentation Changes**
  - Updated `model_understanding.ipynb` to demo the two-way partial dependence capability #1919
- Testing Changes

Warning:

Breaking Changes

- Deleted the `explain_prediction` function #1915
- Removed `HighVarianceCVDataCheck` and convered it to an `AutoMLSearch` method instead #1928
- Added `score_batch` and `train_batch` abstact methods to `EngineBase`. These need to be implemented in Engine subclasses #1913

v0.19.0 Feb. 23, 2021

- **Enhancements**
  - Added a GitHub Action for Python windows unit tests #1844
  - Added a GitHub Action for checking updated release notes #1849
  - Added a GitHub Action for Python lint checks #1837
  - Adjusted `explain_prediction`, `explain_predictions` and `explain_predictions_best_worst` to handle timeseries problems. #1818
  - Updated `InvalidTargetDataCheck` to check for mismatched indices in target and features #1816
  - Updated `Woodwork` structures returned from components to support `Woodwork` logical type overrides set by the user #1784
  - Updated estimators to keep track of input feature names during `fit()` #1794
  - Updated `visualize_decision_tree` to include feature names in output #1813
  - Added `is_bounded_like_percentage` property for objectives. If true, the `calculate_percent_difference` method will return the absolute difference rather than relative difference #1809
  - Added full error traceback to AutoMLSearch logger file #1840
  - Changed `TargetEncoder` to preserve custom indices in the data #1836
  - Refactored `explain_predictions` and `explain_predictions_best_worst` to only compute features once for all rows that need to be explained #1843
  - Added custom random undersampler data splitter for classification #1857
  - Updated `OutliersDataCheck` implementation to calculate the probability of having no outliers #1855
  - Added `Engines` pipeline processing API #1838
- **Fixes**
  - Changed EngineBase random_state arg to random_seed and same for user guide docs #1889
- **Changes**
  - Modified `calculate_percent_difference` so that division by 0 is now inf rather than nan #1809
  - Removed `text_columns` parameter from `LSA` and `TextFeaturizer` components #1652
  - Added `random_seed` as an argument to our automl/pipeline/component API. Using `random_state` will raise a warning #1798

  - Added `DataCheckError` message in `InvalidTargetDataCheck` if input target is None and removed exception raised #1866
- Documentation Changes
- **Testing Changes**
  - Added back coverage for `_get_feature_provenance` in `TextFeaturizer` after `text_columns` was removed #1842
  - Pin graphviz version for windows builds #1847
  - Unpin graphviz version for windows builds #1851

Warning:

Breaking Changes

- Added a deprecation warning to `explain_prediction`. It will be deleted in the next release. #1860

v0.18.2 Feb. 10, 2021

- **Enhancements**
  - Added uniqueness score data check #1785
  - Added "dataframe" output format for prediction explanations #1781
  - Updated LightGBM estimators to handle `pandas.MultiIndex` #1770
  - Sped up permutation importance for some pipelines #1762
  - Added sparsity data check #1797
  - Confirmed support for threshold tuning for binary time series classification problems #1803
- Fixes
- Changes
- **Documentation Changes**
  - Added section on conda to the contributing guide #1771
  - Updated release process to reflect freezing *main* before perf tests #1787
  - Moving some prs to the right section of the release notes #1789
  - Tweak README.md. #1800
  - Fixed back arrow on install page docs #1795
  - Fixed docstring for *ClassImbalanceDataCheck.validate()* #1817
- Testing Changes

v0.18.1 Feb. 1, 2021

- **Enhancements**
  - Added `graph_t_sne` as a visualization tool for high dimensional data #1731
  - Added the ability to see the linear coefficients of features in linear models terms #1738
  - Added support for `scikit-learn v0.24.0` #1733
  - Added support for `scipy v1.6.0` #1752

  - Added SVM Classifier and Regressor to estimators #1714 #1761

- **Fixes**
  - Addressed bug with `partial_dependence` and categorical data with more categories than grid resolution #1748
  - Removed `random_state` arg from `get_pipelines` in `AutoMLSearch` #1719
  - Pinned pyzmq at less than 22.0.0 till we add support #1756
  - Remove `ProphetRegressor` from main as windows tests were flaky #1764
- **Changes**
  - Updated components and pipelines to return `Woodwork` data structures #1668
  - Updated `clone()` for pipelines and components to copy over random state automatically #1753
  - Dropped support for Python version 3.6 #1751
  - Removed deprecated `verbose` flag from `AutoMLSearch` parameters #1772
- **Documentation Changes**
  - Add Twitter and Github link to documentation toolbar #1754
  - Added Open Graph info to documentation #1758
- Testing Changes

Warning:

Breaking Changes

- Components and pipelines return `Woodwork` data structures instead of `pandas` data structures #1668
- Python 3.6 will not be actively supported due to discontinued support from EvalML dependencies.
- Deprecated `verbose` flag is removed for `AutoMLSearch` #1772

v0.18.0 Jan. 26, 2021

- **Enhancements**
  - Added RMSLE, MSLE, and MAPE to core objectives while checking for negative target values in `invalid_targets_data_check` #1574
  - Added validation checks for binary problems with regression-like datasets and multiclass problems without true multiclass targets in `invalid_targets_data_check` #1665
  - Added time series support for `make_pipeline` #1566
  - Added target name for output of pipeline `predict` method #1578
  - Added multiclass check to `InvalidTargetDataCheck` for two examples per class #1596
  - Added support for `graphviz v0.16` #1657
  - Enhanced time series pipelines to accept empty features #1651
  - Added KNN Classifier to estimators. #1650
  - Added support for list inputs for objectives #1663
  - Added support for `AutoMLSearch` to handle time series classification pipelines #1666

  - Enhanced `DelayedFeaturesTransformer` to encode categorical features and targets before delaying them #1691
  - Added 2-way dependence plots. #1690
  - Added ability to directly iterate through components within Pipelines #1583
- **Fixes**
  - Fixed inconsistent attributes and added Exceptions to docs #1673
  - Fixed `TargetLeakageDataCheck` to use Woodwork `mutual_information` rather than using Pandas' Pearson Correlation #1616
  - Fixed thresholding for pipelines in `AutoMLSearch` to only threshold binary classification pipelines #1622 #1626
  - Updated `load_data` to return Woodwork structures and update default parameter value for `index` to `None` #1610
  - Pinned scipy at < 1.6.0 while we work on adding support #1629
  - Fixed data check message formatting in `AutoMLSearch` #1633
  - Addressed stacked ensemble component for `scikit-learn` v0.24 support by setting `shuffle=True` for default CV #1613
  - Fixed bug where `Imputer` reset the index on `X` #1590
  - Fixed `AutoMLSearch` stacktrace when a cutom objective was passed in as a primary objective or additional objective #1575
  - Fixed custom index bug for `MAPE` objective #1641
  - Fixed index bug for `TextFeaturizer` and `LSA` components #1644
  - Limited `load_fraud` dataset loaded into `automl.ipynb` #1646
  - `add_to_rankings` updates `AutoMLSearch.best_pipeline` when necessary #1647
  - Fixed bug where time series baseline estimators were not receiving `gap` and `max_delay` in `AutoMLSearch` #1645
  - Fixed jupyter notebooks to help the RTD buildtime #1654
  - Added `positive_only` objectives to `non_core_objectives` #1661
  - Fixed stacking argument `n_jobs` for IterativeAlgorithm #1706
  - Updated CatBoost estimators to return self in `.fit()` rather than the underlying model for consistency #1701
  - Added ability to initialize pipeline parameters in `AutoMLSearch` constructor #1676
- **Changes**
  - Added labeling to `graph_confusion_matrix` #1632
  - Rerunning search for `AutoMLSearch` results in a message thrown rather than failing the search, and removed `has_searched` property #1647
  - Changed tuner class to allow and ignore single parameter values as input #1686
  - Capped LightGBM version limit to remove bug in docs #1711
  - Removed support for *np.random.RandomState* in EvalML #1727
- **Documentation Changes**

  - Update Model Understanding in the user guide to include `visualize_decision_tree` #1678
  - Updated docs to include information about `AutoMLSearch` callback parameters and methods #1577
  - Updated docs to prompt users to install graphiz on Mac #1656
  - Added `infer_feature_types` to the `start.ipynb` guide #1700
  - Added multicollinearity data check to API reference and docs #1707
- Testing Changes

Warning:

Breaking Changes

- Removed `has_searched` property from `AutoMLSearch` #1647
- Components and pipelines return `Woodwork` data structures instead of `pandas` data structures #1668
- Removed support for *np.random.RandomState* in EvalML. Rather than passing `np.random.RandomState` as component and pipeline random_state values, we use int random_seed #1727

v0.17.0 Dec. 29, 2020

- **Enhancements**
  - Added `save_plot` that allows for saving figures from different backends #1588
  - Added `LightGBM Regressor` to regression components #1459
  - Added `visualize_decision_tree` for tree visualization with `decision_tree_data_from_estimator` and `decision_tree_data_from_pipeline` to reformat tree structure output #1511
  - Added *DFS Transformer* component into transformer components #1454
  - Added `MAPE` to the standard metrics for time series problems and update objectives #1510
  - Added `graph_prediction_vs_actual_over_time` and `get_prediction_vs_actual_over_time_data` to the model understanding module for time series problems #1483
  - Added a `ComponentGraph` class that will support future pipelines as directed acyclic graphs #1415
  - Updated data checks to accept `Woodwork` data structures #1481
  - Added parameter to `InvalidTargetDataCheck` to show only top unique values rather than all unique values #1485
  - Added multicollinearity data check #1515
  - Added baseline pipeline and components for time series regression problems #1496
  - Added more information to users about ensembling behavior in `AutoMLSearch` #1527
  - Add woodwork support for more utility and graph methods #1544
  - Changed `DateTimeFeaturizer` to encode features as int #1479
  - Return trained pipelines from `AutoMLSearch.best_pipeline` #1547

  - Added utility method so that users can set feature types without having to learn about Woodwork directly #1555
  - Added Linear Discriminant Analysis transformer for dimensionality reduction #1331
  - Added multiclass support for `partial_dependence` and `graph_partial_dependence` #1554
  - Added `TimeSeriesBinaryClassificationPipeline` and `TimeSeriesMulticlassClassificationPipeline` classes #1528
  - Added `make_data_splitter` method for easier automl data split customization #1568
  - Integrated `ComponentGraph` class into Pipelines for full non-linear pipeline support #1543
  - Update `AutoMLSearch` constructor to take training data instead of `search` and `add_to_leaderboard` #1597
  - Update `split_data` helper args #1597
  - Add problem type utils `is_regression`, `is_classification`, `is_timeseries` #1597
  - Rename `AutoMLSearch` `data_split` arg to `data_splitter` #1569
- **Fixes**
  - Fix AutoML not passing CV folds to `DefaultDataChecks` for usage by `ClassImbalanceDataCheck` #1619
  - Fix Windows CI jobs: install `numba` via conda, required for `shap` #1490
  - Added custom-index support for *reset-index-get_prediction_vs_actual_over_time_data* #1494
  - Fix `generate_pipeline_code` to account for boolean and None differences between Python and JSON #1524 #1531
  - Set max value for plotly and xgboost versions while we debug CI failures with newer versions #1532
  - Undo version pinning for plotly #1533
  - Fix ReadTheDocs build by updating the version of `setuptools` #1561
  - Set `random_state` of data splitter in AutoMLSearch to take int to keep consistency in the resulting splits #1579
  - Pin sklearn version while we work on adding support #1594
  - Pin pandas at <1.2.0 while we work on adding support #1609
  - Pin graphviz at < 0.16 while we work on adding support #1609
- **Changes**
  - Reverting `save_graph` #1550 to resolve kaleido build issues #1585
  - Update circleci badge to apply to `main` #1489
  - Added script to generate github markdown for releases #1487
  - Updated selection using pandas `dtypes` to selecting using Woodwork logical types #1551
  - Updated dependencies to fix `ImportError: cannot import name 'MaskedArray' from 'sklearn.utils.fixes'` error and to address Woodwork and Featuretool dependencies #1540
  - Made `get_prediction_vs_actual_data()` a public method #1553

  - Updated `Woodwork` version requirement to v0.0.7 #1560
  - Move data splitters from `evalml.automl.data_splitters` to `evalml.preprocessing.data_splitters` #1597
  - Rename "# Testing" in automl log output to "# Validation" #1597
- **Documentation Changes**
  - Added partial dependence methods to API reference #1537
  - Updated documentation for confusion matrix methods #1611
- **Testing Changes**
  - Set `n_jobs=1` in most unit tests to reduce memory #1505

Warning:

Breaking Changes

- Updated minimal dependencies: `numpy>=1.19.1`, `pandas>=1.1.0`, `scikit-learn>=0.23.1`, `scikit-optimize>=0.8.1`
- Updated `AutoMLSearch.best_pipeline` to return a trained pipeline. Pass in `train_best_pipeline=False` to AutoMLSearch in order to return an untrained pipeline.
- Pipeline component instances can no longer be iterated through using `Pipeline.component_graph` #1543
- Update `AutoMLSearch` constructor to take training data instead of `search` and `add_to_leaderboard` #1597
- Update `split_data` helper args #1597
- Move data splitters from `evalml.automl.data_splitters` to `evalml.preprocessing.data_splitters` #1597
- Rename `AutoMLSearch` `data_split` arg to `data_splitter` #1569

v0.16.1 Dec. 1, 2020

- **Enhancements**
  - Pin woodwork version to v0.0.6 to avoid breaking changes #1484
  - Updated `Woodwork` to >=0.0.5 in `core-requirements.txt` #1473
  - Removed `copy_dataframe` parameter for `Woodwork`, updated `Woodwork` to >=0.0.6 in `core-requirements.txt` #1478
  - Updated `detect_problem_type` to use `pandas.api.is_numeric_dtype` #1476
- **Changes**
  - Changed `make clean` to delete coverage reports as a convenience for developers #1464
  - Set `n_jobs=-1` by default for stacked ensemble components #1472
- **Documentation Changes**
  - Updated pipeline and component documentation and demos to use `Woodwork` #1466
- **Testing Changes**
  - Update dependency update checker to use everything from core and optional dependencies #1480

v0.16.0 Nov. 24, 2020

- **Enhancements**
  - Updated pipelines and `make_pipeline` to accept `Woodwork` inputs #1393
  - Updated components to accept `Woodwork` inputs #1423
  - Added ability to freeze hyperparameters for `AutoMLSearch` #1284
  - Added `Target Encoder` into transformer components #1401
  - Added callback for error handling in `AutoMLSearch` #1403
  - Added the index id to the `explain_predictions_best_worst` output to help users identify which rows in their data are included #1365
  - The top_k features displayed in `explain_predictions_*` functions are now determined by the magnitude of shap values as opposed to the `top_k` largest and smallest shap values. #1374
  - Added a problem type for time series regression #1386
  - Added a `is_defined_for_problem_type` method to `ObjectiveBase` #1386
  - Added a `random_state` parameter to `make_pipeline_from_components` function #1411
  - Added `DelayedFeaturesTransformer` #1396
  - Added a `TimeSeriesRegressionPipeline` class #1418
  - Removed `core-requirements.txt` from the package distribution #1429
  - Updated data check messages to include a *"code"* and *"details"* fields #1451, #1462
  - Added a `TimeSeriesSplit` data splitter for time series problems #1441
  - Added a `problem_configuration` parameter to AutoMLSearch #1457
- **Fixes**
  - Fixed `IndexError` raised in `AutoMLSearch` when `ensembling = True` but only one pipeline to iterate over #1397
  - Fixed stacked ensemble input bug and LightGBM warning and bug in `AutoMLSearch` #1388
  - Updated enum classes to show possible enum values as attributes #1391
  - Updated calls to `Woodwork`'s `to_pandas()` to `to_series()` and `to_dataframe()` #1428
  - Fixed bug in OHE where column names were not guaranteed to be unique #1349
  - Fixed bug with percent improvement of `ExpVariance` objective on data with highly skewed target #1467
  - Fix SimpleImputer error which occurs when all features are bool type #1215
- **Changes**
  - Changed `OutliersDataCheck` to return the list of columns, rather than rows, that contain outliers #1377
  - Simplified and cleaned output for Code Generation #1371
  - Reverted changes from #1337 #1409
  - Updated data checks to return dictionary of warnings and errors instead of a list #1448

  - Updated `AutoMLSearch` to pass `Woodwork` data structures to every pipeline (instead of pandas DataFrames) #1450
  - Update `AutoMLSearch` to default to `max_batches=1` instead of `max_iterations=5` #1452
  - Updated _evaluate_pipelines to consolidate side effects #1410
- **Documentation Changes**
  - Added description of CLA to contributing guide, updated description of draft PRs #1402
  - Updated documentation to include all data checks, `DataChecks`, and usage of data checks in AutoML #1412
  - Updated docstrings from `np.array` to `np.ndarray` #1417
  - Added section on stacking ensembles in AutoMLSearch documentation #1425
- **Testing Changes**
  - Removed `category_encoders` from test-requirements.txt #1373
  - Tweak codecov.io settings again to avoid flakes #1413
  - Modified `make lint` to check notebook versions in the docs #1431
  - Modified `make lint-fix` to standardize notebook versions in the docs #1431
  - Use new version of pull request Github Action for dependency check (#1443)
  - Reduced number of workers for tests to 4 #1447

Warning:

Breaking Changes

- The `top_k` and `top_k_features` parameters in `explain_predictions_*` functions now return `k` features as opposed to `2 * k` features #1374
- Renamed `problem_type` to `problem_types` in `RegressionObjective`, `BinaryClassificationObjective`, and `MulticlassClassificationObjective` #1319
- Data checks now return a dictionary of warnings and errors instead of a list #1448

v0.15.0 Oct. 29, 2020

- **Enhancements**
  - Added stacked ensemble component classes (`StackedEnsembleClassifier`, `StackedEnsembleRegressor`) #1134
  - Added stacked ensemble components to `AutoMLSearch` #1253
  - Added `DecisionTreeClassifier` and `DecisionTreeRegressor` to AutoML #1255
  - Added `graph_prediction_vs_actual` in `model_understanding` for regression problems #1252
  - Added parameter to `OneHotEncoder` to enable filtering for features to encode for #1249
  - Added percent-better-than-baseline for all objectives to automl.results #1244
  - Added `HighVarianceCVDataCheck` and replaced synonymous warning in `AutoMLSearch` #1254

  - Added *PCA Transformer* component for dimensionality reduction #1270
  - Added `generate_pipeline_code` and `generate_component_code` to allow for code generation given a pipeline or component instance #1306
  - Added `PCA Transformer` component for dimensionality reduction #1270
  - Updated `AutoMLSearch` to support `Woodwork` data structures #1299
  - Added cv_folds to `ClassImbalanceDataCheck` and added this check to `DefaultDataChecks` #1333
  - Make `max_batches` argument to `AutoMLSearch.search` public #1320
  - Added text support to automl search #1062
  - Added `_pipelines_per_batch` as a private argument to `AutoMLSearch` #1355

- **Fixes**
  - Fixed ML performance issue with ordered datasets: always shuffle data in automl's default CV splits #1265
  - Fixed broken `evalml info` CLI command #1293
  - Fixed `boosting type='rf'` for LightGBM Classifier, as well as `num_leaves` error #1302
  - Fixed bug in `explain_predictions_best_worst` where a custom index in the target variable would cause a `ValueError` #1318
  - Added stacked ensemble estimators to to `evalml.pipelines.__init__` file #1326
  - Fixed bug in OHE where calls to transform were not deterministic if `top_n` was less than the number of categories in a column #1324
  - Fixed LightGBM warning messages during AutoMLSearch #1342
  - Fix warnings thrown during AutoMLSearch in `HighVarianceCVDataCheck` #1346
  - Fixed bug where TrainingValidationSplit would return invalid location indices for dataframes with a custom index #1348
  - Fixed bug where the AutoMLSearch `random_state` was not being passed to the created pipelines #1321

- **Changes**
  - Allow `add_to_rankings` to be called before AutoMLSearch is called #1250
  - Removed Graphviz from test-requirements to add to requirements.txt #1327
  - Removed `max_pipelines` parameter from `AutoMLSearch` #1264
  - Include editable installs in all install make targets #1335
  - Made pip dependencies *featuretools* and *nlp_primitives* core dependencies #1062
  - Removed *PartOfSpeechCount* from *TextFeaturizer* transform primitives #1062
  - Added warning for `partial_dependency` when the feature includes null values #1352

- **Documentation Changes**
  - Fixed and updated code blocks in Release Notes #1243
  - Added DecisionTree estimators to API Reference #1246
  - Changed class inheritance display to flow vertically #1248

  - Updated cost-benefit tutorial to use a holdout/test set #1159
  - Added `evalml info` command to documentation #1293
  - Miscellaneous doc updates #1269
  - Removed conda pre-release testing from the release process document #1282
  - Updates to contributing guide #1310
  - Added Alteryx footer to docs with Twitter and Github link #1312
  - Added documentation for evalml installation for Python 3.6 #1322
  - Added documentation changes to make the API Docs easier to understand #1323
  - Fixed documentation for `feature_importance` #1353
  - Added tutorial for running *AutoML* with text data #1357
  - Added documentation for woodwork integration with automl search #1361
- **Testing Changes**
  - Added tests for `jupyter_check` to handle IPython #1256
  - Cleaned up `make_pipeline` tests to test for all estimators #1257
  - Added a test to check conda build after merge to main #1247
  - Removed code that was lacking codecov for `__main__.py` and unnecessary #1293
  - Codecov: round coverage up instead of down #1334
  - Add DockerHub credentials to CI testing environment #1356
  - Add DockerHub credentials to conda testing environment #1363

Warning:

Breaking Changes

- Renamed `LabelLeakageDataCheck` to `TargetLeakageDataCheck` #1319
- `max_pipelines` parameter has been removed from `AutoMLSearch`. Please use `max_iterations` instead. #1264
- `AutoMLSearch.search()` will now log a warning if the input is not a `Woodwork` data structure (`pandas`, `numpy`) #1299
- Make `max_batches` argument to `AutoMLSearch.search` public #1320
- Removed unused argument *feature_types* from AutoMLSearch.search #1062

v0.14.1 Sep. 29, 2020

- **Enhancements**
  - Updated partial dependence methods to support calculating numeric columns in a dataset with non-numeric columns #1150
  - Added `get_feature_names` on `OneHotEncoder` #1193
  - Added `detect_problem_type` to `problem_type/utils.py` to automatically detect the problem type given targets #1194
  - Added LightGBM to `AutoMLSearch` #1199

- Updated `scikit-learn` and `scikit-optimize` to use latest versions - 0.23.2 and 0.8.1 respectively #1141
- Added `__str__` and `__repr__` for pipelines and components #1218
- Included internal target check for both training and validation data in `AutoMLSearch` #1226
- Added `ProblemTypes.all_problem_types` helper to get list of supported problem types #1219
- Added `DecisionTreeClassifier` and `DecisionTreeRegressor` classes #1223
- Added `ProblemTypes.all_problem_types` helper to get list of supported problem types #1219
- `DataChecks` can now be parametrized by passing a list of `DataCheck` classes and a parameter dictionary #1167
- Added first CV fold score as validation score in `AutoMLSearch.rankings` #1221
- Updated `flake8` configuration to enable linting on `__init__.py` files #1234
- Refined `make_pipeline_from_components` implementation #1204

- **Fixes**
  - Updated GitHub URL after migration to Alteryx GitHub org #1207
  - Changed Problem Type enum to be more similar to the string name #1208
  - Wrapped call to scikit-learn's partial dependence method in a `try`/`finally` block #1232
- **Changes**
  - Added `allow_writing_files` as a named argument to CatBoost estimators. #1202
  - Added `solver` and `multi_class` as named arguments to `LogisticRegressionClassifier` #1202
  - Replaced pipeline's `._transform` method to evaluate all the preprocessing steps of a pipeline with `.compute_estimator_features` #1231
  - Changed default large dataset train/test splitting behavior #1205
- **Documentation Changes**
  - Included description of how to access the component instances and features for pipeline user guide #1163
  - Updated API docs to refer to target as "target" instead of "labels" for non-classification tasks and minor docs cleanup #1160
  - Added Class Imbalance Data Check to `api_reference.rst` #1190 #1200
  - Added pipeline properties to API reference #1209
  - Clarified what the objective parameter in AutoML is used for in AutoML API reference and AutoML user guide #1222
  - Updated API docs to include `skopt.space.Categorical` option for component hyperparameter range definition #1228
  - Added install documentation for `libomp` in order to use LightGBM on Mac #1233
  - Improved description of `max_iterations` in documentation #1212
  - Removed unused code from sphinx conf #1235

- Testing Changes

Warning:

Breaking Changes

- `DefaultDataChecks` now accepts a `problem_type` parameter that must be specified #1167
- Pipeline's `._transform` method to evaluate all the preprocessing steps of a pipeline has been replaced with `.compute_estimator_features` #1231
- `get_objectives` has been renamed to `get_core_objectives`. This function will now return a list of valid objective instances #1230

v0.13.2 Sep. 17, 2020

- **Enhancements**
  - Added `output_format` field to explain predictions functions #1107
  - Modified `get_objective` and `get_objectives` to be able to return any objective in `evalml.objectives` #1132
  - Added a `return_instance` boolean parameter to `get_objective` #1132
  - Added `ClassImbalanceDataCheck` to determine whether target imbalance falls below a given threshold #1135
  - Added label encoder to LightGBM for binary classification #1152
  - Added labels for the row index of confusion matrix #1154
  - Added `AutoMLSearch` object as another parameter in search callbacks #1156
  - Added the corresponding probability threshold for each point displayed in `graph_roc_curve` #1161
  - Added `__eq__` for `ComponentBase` and `PipelineBase` #1178
  - Added support for multiclass classification for `roc_curve` #1164
  - Added `categories` accessor to `OneHotEncoder` for listing the categories associated with a feature #1182
  - Added utility function to create pipeline instances from a list of component instances #1176
- **Fixes**
  - Fixed XGBoost column names for partial dependence methods #1104
  - Removed dead code validating column type from `TextFeaturizer` #1122
  - Fixed issue where `Imputer` cannot fit when there is None in a categorical or boolean column #1144
  - `OneHotEncoder` preserves the custom index in the input data #1146
  - Fixed representation for `ModelFamily` #1165
  - Removed duplicate `nbsphinx` dependency in `dev-requirements.txt` #1168
  - Users can now pass in any valid kwargs to all estimators #1157
  - Remove broken accessor `OneHotEncoder.get_feature_names` and unneeded base class #1179

  - Removed LightGBM Estimator from AutoML models #1186

- **Changes**
  - Pinned `scikit-optimize` version to 0.7.4 #1136
  - Removed `tqdm` as a dependency #1177
  - Added lightgbm version 3.0.0 to `latest_dependency_versions.txt` #1185
  - Rename `max_pipelines` to `max_iterations` #1169
- **Documentation Changes**
  - Fixed API docs for `AutoMLSearch` `add_result_callback` #1113
  - Added a step to our release process for pushing our latest version to conda-forge #1118
  - Added warning for missing ipywidgets dependency for using `PipelineSearchPlots` on Jupyterlab #1145
  - Updated `README.md` example to load demo dataset #1151
  - Swapped mapping of breast cancer targets in `model_understanding.ipynb` #1170
- **Testing Changes**
  - Added test confirming `TextFeaturizer` never outputs null values #1122
  - Changed Python version of `Update Dependencies` action to 3.8.x #1137
  - Fixed release notes check-in test for `Update Dependencies` actions #1172

Warning:

Breaking Changes

- `get_objective` will now return a class definition rather than an instance by default #1132
- Deleted `OPTIONS` dictionary in `evalml.objectives.utils.py` #1132
- If specifying an objective by string, the string must now match the objective's name field, case-insensitive #1132
- **Passing "Cost Benefit Matrix", "Fraud Cost", "Lead Scoring", "Mean Squared Log Error",** "Recall", "Recall Macro", "Recall Micro", "Recall Weighted", or "Root Mean Squared Log Error" to `AutoMLSearch` will now result in a `ValueError` rather than an `ObjectiveNotFoundError` #1132
- Search callbacks `start_iteration_callback` and `add_results_callback` have changed to include a copy of the AutoMLSearch object as a third parameter #1156
- Deleted `OneHotEncoder.get_feature_names` method which had been broken for a while, in favor of pipelines' `input_feature_names` #1179
- Deleted empty base class `CategoricalEncoder` which `OneHotEncoder` component was inheriting from #1176
- Results from `roc_curve` will now return as a list of dictionaries with each dictionary representing a class #1164
- `max_pipelines` now raises a `DeprecationWarning` and will be removed in the next release. `max_iterations` should be used instead. #1169

v0.13.1 Aug. 25, 2020

- **Enhancements**
  - Added Cost-Benefit Matrix objective for binary classification #1038
  - Split `fill_value` into `categorical_fill_value` and `numeric_fill_value` for Imputer #1019
  - Added `explain_predictions` and `explain_predictions_best_worst` for explaining multiple predictions with SHAP #1016
  - Added new LSA component for text featurization #1022
  - Added guide on installing with conda #1041
  - Added a "cost-benefit curve" util method to graph cost-benefit matrix scores vs. binary classification thresholds #1081
  - Standardized error when calling transform/predict before fit for pipelines #1048
  - Added `percent_better_than_baseline` to AutoML search rankings and full rankings table #1050
  - Added one-way partial dependence and partial dependence plots #1079
  - Added "Feature Value" column to prediction explanation reports. #1064
  - Added LightGBM classification estimator #1082, #1114
  - Added `max_batches` parameter to `AutoMLSearch` #1087
- **Fixes**
  - Updated `TextFeaturizer` component to no longer require an internet connection to run #1022
  - Fixed non-deterministic element of `TextFeaturizer` transformations #1022
  - Added a StandardScaler to all ElasticNet pipelines #1065
  - Updated cost-benefit matrix to normalize score #1099
  - Fixed logic in `calculate_percent_difference` so that it can handle negative values #1100
- **Changes**
  - Added `needs_fitting` property to `ComponentBase` #1044
  - Updated references to data types to use datatype lists defined in `evalml.utils.gen_utils` #1039
  - Remove maximum version limit for SciPy dependency #1051
  - Moved `all_components` and other component importers into runtime methods #1045
  - Consolidated graphing utility methods under `evalml.utils.graph_utils` #1060
  - Made slight tweaks to how `TextFeaturizer` uses `featuretools`, and did some refactoring of that and of LSA #1090
  - Changed `show_all_features` parameter into `importance_threshold`, which allows for thresholding feature importance #1097, #1103
- **Documentation Changes**
  - Update `setup.py` URL to point to the github repo #1037
  - Added tutorial for using the cost-benefit matrix objective #1088

  - Updated `model_understanding.ipynb` to include documentation for using plotly on Jupyter Lab #1108

- **Testing Changes**
  - Refactor CircleCI tests to use matrix jobs (#1043)
  - Added a test to check that all test directories are included in evalml package #1054

Warning:

Breaking Changes

- `confusion_matrix` and `normalize_confusion_matrix` have been moved to `evalml.utils` #1038
- All graph utility methods previously under `evalml.pipelines.graph_utils` have been moved to `evalml.utils.graph_utils` #1060

v0.12.2 Aug. 6, 2020

- **Enhancements**
  - Add save/load method to components #1023
  - Expose pickle `protocol` as optional arg to save/load #1023
  - Updated estimators used in AutoML to include ExtraTrees and ElasticNet estimators #1030
- Fixes
- **Changes**
  - Removed `DeprecationWarning` for `SimpleImputer` #1018
- **Documentation Changes**
  - Add note about version numbers to release process docs #1034
- **Testing Changes**
  - Test files are now included in the evalml package #1029

v0.12.0 Aug. 3, 2020

- **Enhancements**
  - Added string and categorical targets support for binary and multiclass pipelines and check for numeric targets for `DetectLabelLeakage` data check #932
  - Added clear exception for regression pipelines if target datatype is string or categorical #960
  - Added target column names and class labels in `predict` and `predict_proba` output for pipelines #951
  - Added `_compute_shap_values` and `normalize_values` to `pipelines/explanations` module #958
  - Added `explain_prediction` feature which explains single predictions with SHAP #974
  - Added Imputer to allow different imputation strategies for numerical and categorical dtypes #991
  - Added support for configuring logfile path using env var, and don't create logger if there are filesystem errors #975

  - Updated catboost estimators' default parameters and automl hyperparameter ranges to speed up fit time #998

- **Fixes**
  - Fixed ReadtheDocs warning failure regarding embedded gif #943
  - Removed incorrect parameter passed to pipeline classes in `_add_baseline_pipelines` #941
  - Added universal error for calling `predict`, `predict_proba`, `transform`, and `feature_importances` before fitting #969, #994
  - Made `TextFeaturizer` component and pip dependencies `featuretools` and `nlp_primitives` optional #976
  - Updated imputation strategy in automl to no longer limit impute strategy to `most_frequent` for all features if there are any categorical columns #991
  - Fixed `UnboundLocalError` for `cv_pipeline` when automl search errors #996
  - Fixed `Imputer` to reset dataframe index to preserve behavior expected from `SimpleImputer` #1009
- **Changes**
  - Moved `get_estimators` to `evalml.pipelines.components.utils` #934
  - Modified Pipelines to raise `PipelineScoreError` when they encounter an error during scoring #936
  - Moved `evalml.model_families.list_model_families` to `evalml.pipelines.components.allowed_model_families` #959
  - Renamed `DateTimeFeaturization` to `DateTimeFeaturizer` #977
  - Added check to stop search and raise an error if all pipelines in a batch return NaN scores #1015
- **Documentation Changes**
  - Updated `README.md` #963
  - Reworded message when errors are returned from data checks in search #982
  - Added section on understanding model predictions with `explain_prediction` to User Guide #981
  - Added a section to the user guide and api reference about how XGBoost and CatBoost are not fully supported. #992
  - Added custom components section in user guide #993
  - Updated FAQ section formatting #997
  - Updated release process documentation #1003
- **Testing Changes**
  - Moved `predict_proba` and `predict` tests regarding string / categorical targets to `test_pipelines.py` #972
  - Fixed dependency update bot by updating python version to 3.7 to avoid frequent github version updates #1002

Warning:

Breaking Changes

- `get_estimators` has been moved to `evalml.pipelines.components.utils` (previously was under `evalml.pipelines.utils`) #934
- Removed the `raise_errors` flag in AutoML search. All errors during pipeline evaluation will be caught and logged. #936
- `evalml.model_families.list_model_families` has been moved to `evalml.pipelines.components.allowed_model_families` #959
- `TextFeaturizer`: the `featuretools` and `nlp_primitives` packages must be installed after installing evalml in order to use this component #976
- Renamed `DateTimeFeaturization` to `DateTimeFeaturizer` #977

v0.11.2 July 16, 2020

- **Enhancements**
  - Added `NoVarianceDataCheck` to `DefaultDataChecks` #893
  - Added text processing and featurization component `TextFeaturizer` #913, #924
  - Added additional checks to `InvalidTargetDataCheck` to handle invalid target data types #929
  - `AutoMLSearch` will now handle `KeyboardInterrupt` and prompt user for confirmation #915
- **Fixes**
  - Makes automl results a read-only property #919
- **Changes**
  - Deleted static pipelines and refactored tests involving static pipelines, removed `all_pipelines()` and `get_pipelines()` #904
  - Moved `list_model_families` to `evalml.model_family.utils` #903
  - Updated `all_pipelines`, `all_estimators`, `all_components` to use the same mechanism for dynamically generating their elements #898
  - Rename `master` branch to `main` #918
  - Add pypi release github action #923
  - Updated `AutoMLSearch.search` stdout output and logging and removed tqdm progress bar #921
  - Moved automl config checks previously in `search()` to init #933
- **Documentation Changes**
  - Reorganized and rewrote documentation #937
  - Updated to use pydata sphinx theme #937
  - Updated docs to use `release_notes` instead of `changelog` #942
- **Testing Changes**
  - Cleaned up fixture names and usages in tests #895

Warning:

Breaking Changes

- `list_model_families` has been moved to `evalml.model_family.utils` (previously was under `evalml.pipelines.utils`) #903
- `get_estimators` has been moved to `evalml.pipelines.components.utils` (previously was under `evalml.pipelines.utils`) #934
- Static pipeline definitions have been removed, but similar pipelines can still be constructed via creating an instance of `PipelineBase` #904
- `all_pipelines()` and `get_pipelines()` utility methods have been removed #904

v0.11.0 June 30, 2020

- **Enhancements**
  - Added multiclass support for ROC curve graphing #832
  - Added preprocessing component to drop features whose percentage of NaN values exceeds a specified threshold #834
  - Added data check to check for problematic target labels #814
  - Added PerColumnImputer that allows imputation strategies per column #824
  - Added transformer to drop specific columns #827
  - Added support for `categories`, `handle_error`, and `drop` parameters in `OneHotEncoder` #830 #897
  - Added preprocessing component to handle DateTime columns featurization #838
  - Added ability to clone pipelines and components #842
  - Define getter method for component `parameters` #847
  - Added utility methods to calculate and graph permutation importances #860, #880
  - Added new utility functions necessary for generating dynamic preprocessing pipelines #852
  - Added kwargs to all components #863
  - Updated `AutoSearchBase` to use dynamically generated preprocessing pipelines #870
  - Added SelectColumns transformer #873
  - Added ability to evaluate additional pipelines for automl search #874
  - Added `default_parameters` class property to components and pipelines #879
  - Added better support for disabling data checks in automl search #892
  - Added ability to save and load AutoML objects to file #888
  - Updated `AutoSearchBase.get_pipelines` to return an untrained pipeline instance #876
  - Saved learned binary classification thresholds in automl results cv data dict #876
- **Fixes**
  - Fixed bug where SimpleImputer cannot handle dropped columns #846
  - Fixed bug where PerColumnImputer cannot handle dropped columns #855
  - Enforce requirement that builtin components save all inputted values in their parameters dict #847

  - Don't list base classes in `all_components` output #847
  - Standardize all components to output pandas data structures, and accept either pandas or numpy #853
  - Fixed rankings and full_rankings error when search has not been run #894
- **Changes**
  - Update `all_pipelines` and `all_components` to try initializing pipelines/components, and on failure exclude them #849
  - Refactor `handle_components` to `handle_components_class`, standardize to `ComponentBase` subclass instead of instance #850
  - Refactor "blacklist"/"whitelist" to "allow"/"exclude" lists #854
  - Replaced `AutoClassificationSearch` and `AutoRegressionSearch` with `AutoMLSearch` #871
  - Renamed feature_importances and permutation_importances methods to use singular names (feature_importance and permutation_importance) #883
  - Updated `automl` default data splitter to train/validation split for large datasets #877
  - Added open source license, update some repo metadata #887
  - Removed dead code in `_get_preprocessing_components` #896
- **Documentation Changes**
  - Fix some typos and update the EvalML logo #872
- **Testing Changes**
  - Update the changelog check job to expect the new branching pattern for the deps update bot #836
  - Check that all components output pandas datastructures, and can accept either pandas or numpy #853
  - Replaced `AutoClassificationSearch` and `AutoRegressionSearch` with `AutoMLSearch` #871

Warning:

Breaking Changes

- Pipelines' static `component_graph` field must contain either `ComponentBase` subclasses or `str`, instead of `ComponentBase` subclass instances #850
- Rename `handle_component` to `handle_component_class`. Now standardizes to `ComponentBase` subclasses instead of `ComponentBase` subclass instances #850
- Renamed automl's `cv` argument to `data_split` #877
- Pipelines' and classifiers' `feature_importances` is renamed `feature_importance`, `graph_feature_importances` is renamed `graph_feature_importance` #883
- Passing `data_checks=None` to automl search will not perform any data checks as opposed to default checks. #892
- Pipelines to search for in AutoML are now determined automatically, rather than using the statically-defined pipeline classes. #870

- Updated `AutoSearchBase.get_pipelines` to return an untrained pipeline instance, instead of one which happened to be trained on the final cross-validation fold #876

v0.10.0 May 29, 2020

- **Enhancements**
  - Added baseline models for classification and regression, add functionality to calculate baseline models before searching in AutoML #746
  - Port over highly-null guardrail as a data check and define `DefaultDataChecks` and `DisableDataChecks` classes #745
  - Update `Tuner` classes to work directly with pipeline parameters dicts instead of flat parameter lists #779
  - Add Elastic Net as a pipeline option #812
  - Added new Pipeline option `ExtraTrees` #790
  - Added precicion-recall curve metrics and plot for binary classification problems in `evalml.pipeline.graph_utils` #794
  - Update the default automl algorithm to search in batches, starting with default parameters for each pipeline and iterating from there #793
  - Added `AutoMLAlgorithm` class and `IterativeAlgorithm` impl, separated from `AutoSearchBase` #793
- **Fixes**
  - Update pipeline `score` to return `nan` score for any objective which throws an exception during scoring #787
  - Fixed bug introduced in #787 where binary classification metrics requiring predicted probabilities error in scoring #798
  - CatBoost and XGBoost classifiers and regressors can no longer have a learning rate of 0 #795
- **Changes**
  - Cleanup pipeline `score` code, and cleanup codecov #711
  - Remove `pass` for abstract methods for codecov #730
  - Added __str__ for AutoSearch object #675
  - Add util methods to graph ROC and confusion matrix #720
  - Refactor `AutoBase` to `AutoSearchBase` #758
  - Updated AutoBase with `data_checks` parameter, removed previous `detect_label_leakage` parameter, and added functionality to run data checks before search in AutoML #765
  - Updated our logger to use Python's logging utils #763
  - Refactor most of `AutoSearchBase._do_iteration` impl into `AutoSearchBase._evaluate` #762
  - Port over all guardrails to use the new DataCheck API #789
  - Expanded `import_or_raise` to catch all exceptions #759
  - Adds RMSE, MSLE, RMSLE as standard metrics #788

  - Don't allow `Recall` to be used as an objective for AutoML #784
  - Removed feature selection from pipelines #819
  - Update default estimator parameters to make automl search faster and more accurate #793
- **Documentation Changes**
  - Add instructions to freeze `master` on `release.md` #726
  - Update release instructions with more details #727 #733
  - Add objective base classes to API reference #736
  - Fix components API to match other modules #747
- **Testing Changes**
  - Delete codecov yml, use codecov.io's default #732
  - Added unit tests for fraud cost, lead scoring, and standard metric objectives #741
  - Update codecov client #782
  - Updated AutoBase __str__ test to include no parameters case #783
  - Added unit tests for `ExtraTrees` pipeline #790
  - If codecov fails to upload, fail build #810
  - Updated Python version of dependency action #816
  - Update the dependency update bot to use a suffix when creating branches #817

Warning:

Breaking Changes

- The `detect_label_leakage` parameter for AutoML classes has been removed and replaced by a `data_checks` parameter #765
- Moved ROC and confusion matrix methods from `evalml.pipeline.plot_utils` to `evalml.pipeline.graph_utils` #720
- `Tuner` classes require a pipeline hyperparameter range dict as an init arg instead of a space definition #779
- `Tuner.propose` and `Tuner.add` work directly with pipeline parameters dicts instead of flat parameter lists #779
- `PipelineBase.hyperparameters` and `custom_hyperparameters` use pipeline parameters dict format instead of being represented as a flat list #779
- All guardrail functions previously under `evalml.guardrails.utils` will be removed and replaced by data checks #789
- `Recall` disallowed as an objective for AutoML #784
- `AutoSearchBase` parameter `tuner` has been renamed to `tuner_class` #793
- `AutoSearchBase` parameter `possible_pipelines` and `possible_model_families` have been renamed to `allowed_pipelines` and `allowed_model_families` #793

v0.9.0 Apr. 27, 2020

- **Enhancements**

  - Added `Accuracy` as an standard objective #624
  - Added verbose parameter to load_fraud #560
  - Added Balanced Accuracy metric for binary, multiclass #612 #661
  - Added XGBoost regressor and XGBoost regression pipeline #666
  - Added `Accuracy` metric for multiclass #672
  - Added objective name in `AutoBase.describe_pipeline` #686
  - Added `DataCheck` and `DataChecks`, `Message` classes and relevant subclasses #739
- **Fixes**
  - Removed direct access to `cls.component_graph` #595
  - Add testing files to .gitignore #625
  - Remove circular dependencies from `Makefile` #637
  - Add error case for `normalize_confusion_matrix()` #640
  - Fixed `XGBoostClassifier` and `XGBoostRegressor` bug with feature names that contain [,], or < #659
  - Update `make_pipeline_graph` to not accidentally create empty file when testing if path is valid #649
  - Fix pip installation warning about docsutils version, from boto dependency #664
  - Removed zero division warning for F1/precision/recall metrics #671
  - Fixed `summary` for pipelines without estimators #707
- **Changes**
  - Updated default objective for binary/multiclass 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 `PipelinePlot` class #228
  - Standarizing inputs as `pd.Dataframe` / `pd.Series` #130
  - Enforcing that pipelines must have an estimator as last component #277
  - Added `ipywidgets` as a dependency in `requirements.txt` #278
  - Added Random and Grid Search Tuners #240
- **Documentation Changes**
  - Adding class properties to API reference #244
  - Fix and filter FutureWarnings from scikit-learn #249, #257
  - Adding Linear Regression to API reference and cleaning up some Sphinx warnings #227
- **Testing Changes**
  - Added support for testing on Windows with CircleCI #226
  - Added support for doctests #233

Warning: Breaking Changes

- The `fit()` method for `AutoClassifier` and `AutoRegressor` has been renamed to `search()`.
- `AutoClassifier` has been renamed to `AutoClassificationSearch`
- `AutoRegressor` has been renamed to `AutoRegressionSearch`
- `AutoClassificationSearch.results` and `AutoRegressionSearch.results` now is a dictionary with `pipeline_results` and `search_order` keys. `pipeline_results` can be used to access a dictionary that is identical to the old `.results` dictionary. Whereas, `search_order` returns a list of the search order in terms of `pipeline_id`.
- Pipelines now require an estimator as the last component in `component_list`. Slicing pipelines now throws an `NotImplementedError` to avoid returning pipelines without an estimator.

v0.5.2 Nov. 18, 2019

- **Enhancements**
  - Adding basic pipeline structure visualization #211
- **Documentation Changes**
  - Added notebooks to build process #212

v0.5.1 Nov. 15, 2019

- **Enhancements**
  - Added basic outlier detection guardrail #151
  - Added basic ID column guardrail #135
  - Added support for unlimited pipelines with a `max_time` limit #70
  - Updated .readthedocs.yaml to successfully build #188
- **Fixes**
  - Removed MSLE from default additional objectives #203
  - Fixed `random_state` passed in pipelines #204
  - Fixed slow down in RFRegressor #206
- **Changes**
  - Pulled information for describe_pipeline from pipeline's new describe method #190
  - Refactored pipelines #108
  - Removed guardrails from Auto(*) #202, #208
- **Documentation Changes**
  - Updated documentation to show `max_time` enhancements #189
  - Updated release instructions for RTD #193
  - Added notebooks to build process #212
  - Added contributing instructions #213
  - Added new content #222

v0.5.0 Oct. 29, 2019

- **Enhancements**

  - Added basic one hot encoding #73
  - Use enums for model_type #110
  - Support for splitting regression datasets #112
  - Auto-infer multiclass classification #99
  - Added support for other units in `max_time` #125
  - Detect highly null columns #121
  - Added additional regression objectives #100
  - Show an interactive iteration vs. score plot when using fit() #134
- **Fixes**
  - Reordered `describe_pipeline` #94
  - Added type check for `model_type` #109
  - Fixed `s` units when setting string `max_time` #132
  - Fix objectives not appearing in API documentation #150
- **Changes**
  - Reorganized tests #93
  - Moved logging to its own module #119
  - Show progress bar history #111
  - Using `cloudpickle` instead of pickle to allow unloading of custom objectives #113
  - Removed render.py #154
- **Documentation Changes**
  - Update release instructions #140
  - Include additional_objectives parameter #124
  - Added Changelog #136
- **Testing Changes**
  - Code coverage #90
  - Added CircleCI tests for other Python versions #104
  - Added doc notebooks as tests #139
  - Test metadata for CircleCI and 2 core parallelism #137

v0.4.1 Sep. 16, 2019

- **Enhancements**
  - Added AutoML for classification and regressor using Autobase and Skopt #7 #9
  - Implemented standard classification and regression metrics #7
  - Added logistic regression, random forest, and XGBoost pipelines #7
  - Implemented support for custom objectives #15
  - Feature importance for pipelines #18
  - Serialization for pipelines #19

  - Allow fitting on objectives for optimal threshold #27
  - Added detect label leakage #31
  - Implemented callbacks #42
  - Allow for multiclass classification #21
  - Added support for additional objectives #79
- **Fixes**
  - Fixed feature selection in pipelines #13
  - Made `random_seed` usage consistent #45
- **Documentation Changes**
  - Documentation Changes
  - Added docstrings #6
  - Created notebooks for docs #6
  - Initialized readthedocs EvalML #6
  - Added favicon #38
- **Testing Changes**
  - Added testing for loading data #39

v0.2.0 Aug. 13, 2019

- **Enhancements**
  - Created fraud detection objective #4

v0.1.0 July. 31, 2019

- *First Release*
- **Enhancements**
  - Added lead scoring objecitve #1
  - Added basic classifier #1
- **Documentation Changes**
  - Initialized Sphinx for docs #1

INDEX

Symbols

A

B

C

E

F

G

H

K

L

M

N

O

P

R

S

T

U

V

X