fklearn Documentation

Release 1.22.0

Nubank Data Science Team

Apr 28, 2021

Contents

fklearn uses functional programming principles to make it easier to solve real problems with Machine Learning.

The name is a reference to the widely known scikit-learn library.

fklearn Principles

1. Validation should reflect real-life situations.
2. Production models should match validated models.
3. Models should be production-ready with few extra steps.
4. Reproducibility and in-depth analysis of model results should be easy to achieve.

CHAPTER 1

Contents

1.1 Getting started

1.1.1 Installation

The fklearn library is Python 3.6 compatible only. In order to install it using pip, run:

```
pip install fklearn
```

You can also install from the source:

```
# clone the repository
$ git clone -b master https://github.com/nubank/fklearn.git --depth=1

# open the folder
$ cd fklearn

# install the dependencies
$ pip install -e .
```

If you are a MacOs user, you may need to install some dependencies in order to use LGBM. If you have brew installed, run the following command from the root dir:

```
brew bundle
```

1.1.2 Basics

Learners

While in scikit-learn the main abstraction for a model is a class with the methods `fit` and `transform`, in fklearn we use what we call a **learner function**. A learner function takes in some training data (plus other parameters), learns something from it and returns three things: a *prediction function*, the *transformed training data*, and a *log*.

The **prediction function** always has the same signature: it takes in a Pandas dataframe and returns a Pandas dataframe. It should be able to take in any new dataframe, as long as it contains the required columns, and transform it. The tranform in the fklearn library is equivalent to the transform method of the scikit-learn. In this case, the prediction function simply creates a new column with the predictions of the linear regression model that was trained.

The **transformed training data** is usually just the prediction function applied to the training data. It is useful when you want predictions on your training set, or for building pipelines, as we'll see later.

The **log** is a dictionary, and can include any information that is relevant for inspecting or debugging the learner, e.g., what features were used, how many samples there were in the training set, feature importance or coefficients.

Learner functions are usually partially initialized (curried) before being passed to pipelines or applied to data:

```
from fklearn.training.regression import linear_regression_learner
from fklearn.training.transformation import capper, floorer, prediction_ranger

# initialize several learner functions
capper_fn = capper(columns_to_cap=["income"], precomputed_caps={"income": 50,000})
regression_fn = linear_regression_learner(features=["income", "bill_amount"], target=
↪"spend")
ranger_fn = prediction_ranger(prediction_min=0.0, prediction_max=20000.0)

# apply one individually to some data
p, df, log = regression_fn(training_data)
```

Available learner functions in fklearn can be found inside the `fklearn.training` module.

Pipelines

Learner functions are usually composed into pipelines that apply them in order to data:

```
from fklearn.training.pipeline import build_pipeline

learner = build_pipeline(capper_fn, regression_fn, ranger_fn)
predict_fn, training_predictions, logs = learner(train_data)
```

Pipelines behave exactly as individual learner functions. They guarantee that all steps are applied consistently to both traning and testing/production data.

Validation

Once we have our pipeline defined, we can use fklearn's validation tools to evaluate the performance of our model in different scenarios and using multiple metrics:

```
from fklearn.validation.evaluators import r2_evaluator, spearman_evaluator, combined_
↪evaluators
from fklearn.validation.validator import validator
from fklearn.validation.splitters import k_fold_splitter, stability_curve_time_
↪splitter

evaluation_fn = combined_evaluators(evaluators=[r2_evaluator(target_column="spend"),
                                                spearman_evaluator(target_column=
↪"spend")])

cv_split_fn = k_fold_splitter(n_splits=3, random_state=42)
stability_split_fn = stability_curve_time_splitter(training_time_limit=pd.to_datetime(
↪"2018-01-01"),
```

(continues on next page)

(continued from previous page)

```
                                                    time_column="timestamp")

cross_validation_results = validator(train_data=train_data,
                                     split_fn=cv_split_fn,
                                     train_fn=learner,
                                     eval_fn=evaluation_fn)

stability_validation_results = validator(train_data=train_data,
                                         split_fn=stability_split_fn,
                                         train_fn=learner,
                                         eval_fn=evaluation_fn)
```

The `validator` function receives some data, the learner function with our model plus the following: 1. A *splitting function*: these can be found inside the `fklearn.validation.splitters` module. They split the data into training and evaluation folds in different ways, simulating situations where training and testing data differ. 2. A *evaluation function*: these can be found inside the `fklearn.validation.evaluators` module. They compute various performance metrics of interest on our model's predictions. They can be composed by using `combined_evaluators` for example.

1.1.3 Learn More

- Check this jupyter notebook for some additional examples.
- Our blog post (Part I) gives an overview of the library and motivation behind it.

1.2 Examples

In this section we present practical examples to demonstrate various fklearn features.

1.2.1 List of examples

- learning_curves
- nlp_classification
- regression
- causal_inference
- feature_transformation
- fklearn_overview
- fklearn_overview_dataset_generation

1.3 API

This is a list with all relevant **fklearn** functions. Docstrings should provide enough information in order to understand any individual function.

1.3.1 Preprocessing

Rebalancing (fklearn.preprocessing.rebalancing)

rebalance_by_categorical	Resample dataset so that the result contains the same number of lines per category in categ_column.
rebalance_by_continuous	Resample dataset so that the result contains the same number of lines per bucket in a continuous column.

Splitting (fklearn.preprocessing.splitting)

space_time_split_dataset	Splits panel data using both ID and Time columns, resulting in four datasets
time_split_dataset	Splits temporal data into a training and testing datasets such that all training data comes before the testings one.

1.3.2 Training

Calibration (fklearn.training.calibration)

isotonic_calibration_learner	Fits a single feature isotonic regression to the dataset.

Classification (fklearn.training.classification)

lgbm_classification_learner	Fits an LGBM classifier to the dataset.
logistic_classification_learner	Fits an logistic regression classifier to the dataset.
nlp_logistic_classification_learner	Fits a text vectorizer (TfidfVectorizer) followed by a logistic regression (LogisticRegression).
xgb_classification_learner	Fits an XGBoost classifier to the dataset.

Ensemble (fklearn.training.ensemble)

xgb_octopus_classification_learner	Octopus ensemble allows you to inject domain specific knowledge to force a split in an initial feature, instead of assuming the tree model will do that intelligent split on its own.

Imputation (fklearn.training.imputation)

imputer	Fits a missing value imputer to the dataset.
placeholder_imputer	Fills missing values with a fixed value.

Pipeline (fklearn.training.pipeline)

build_pipeline(*learners, has_repeated_learners)	Builds a pipeline of different chained learners functions with the possibility of using keyword arguments in the predict functions of the pipeline.

Regression (fklearn.training.regression)

gp_regression_learner	Fits an gaussian process regressor to the dataset.
lgbm_regression_learner	Fits an LGBM regressor to the dataset.
linear_regression_learner	Fits an linear regression classifier to the dataset.
xgb_regression_learner	Fits an XGBoost regressor to the dataset.

Transformation (fklearn.training.transformation)

apply_replacements(df, columns, vec, Dict], ...)	Base function to apply the replacements values found on the "vec" vectors into the df DataFrame.
capper(df, columns_to_cap, precomputed_caps, ...)	Learns the maximum value for each of the *columns_to_cap* and used that as the cap for those columns.
count_categorizer(df, columns_to_categorize, ...)	Replaces categorical variables by count.
custom_transformer(df, columns_to_transform, ...)	Applies a custom function to the desired columns.
discrete_ecdfer	Learns an Empirical Cumulative Distribution Function from the specified column in the input DataFrame.
ecdfer	Learns an Empirical Cumulative Distribution Function from the specified column in the input DataFrame.
floorer(df, columns_to_floor, ...)	Learns the minimum value for each of the *columns_to_floor* and used that as the floot for those columns.
label_categorizer(df, columns_to_categorize, ...)	Replaces categorical variables with a numeric identifier.
missing_warner	Creates a new column to warn about rows that columns that don't have missing in the training set but have missing on the scoring
null_injector	Injects null into columns
onehot_categorizer(df, ...)	Onehot encoding on categorical columns.
prediction_ranger	Caps and floors the specified prediction column to a set range.
quantile_biner(df, columns_to_bin, q, right)	Discretize continuous numerical columns into its quantiles.
rank_categorical(df, columns_to_rank, ...)	Rank categorical features by their frequency in the train set.
selector	Filters a DataFrames by selecting only the desired columns.
standard_scaler(df, columns_to_scale)	Fits a standard scaler to the dataset.
target_categorizer(df, ...)	Replaces categorical variables with the smoothed mean of the target variable by category.
truncate_categorical(df, ...)	Truncate infrequent categories and replace them by a single one.

Continued on next page

Table 9 – continued from previous page

value_mapper(df, value_maps, Dict] =, . . .)	Map values in selected columns in the DataFrame according to dictionaries of replacements.

Unsupervised (fklearn.training.unsupervised)

isolation_forest_learner	Fits an anomaly detection algorithm (Isolation Forest) to the dataset

1.3.3 Tuning

Model Agnostic Feature Choice (fklearn.tuning.model_agnostic_fc)

correlation_feature_selection	Feature selection based on correlation
variance_feature_selection	Feature selection based on variance

Parameter Tuning (fklearn.tuning.parameter_tuners)

grid_search_cv
random_search_tuner
seed

Samplers (fklearn.tuning.samplers)

remove_by_feature_importance	Performs feature selection based on feature importance
remove_by_feature_shuffling	Performs feature selection based on the evaluation of the test vs the evaluation of the test with randomly shuffled features
remove_features_subsets	Performs feature selection based on the best performing model out of several trained models

Selectors (fklearn.tuning.selectors)

backward_subset_feature_selection
feature_importance_backward_selection
poor_man_boruta_selection

Stoppers (fklearn.tuning.stoppers)

aggregate_stop_funcs(*stop_funcs)	Aggregate stop functions
stop_by_iter_num	Checks for logs to see if feature selection should stop
stop_by_no_improvement	Checks for logs to see if feature selection should stop
stop_by_no_improvement_parallel	Checks for logs to see if feature selection should stop
stop_by_num_features	Checks for logs to see if feature selection should stop
stop_by_num_features_parallel	Selects the best log out of a list to see if feature selection should stop

1.3.4 Validation

Evaluators (fklearn.validation.evaluators)

roc_auc_evaluator	Computes the ROC AUC score, given true label and prediction scores.
pr_auc_evaluator	Computes the PR AUC score, given true label and prediction scores.
brier_score_evaluator	Computes the Brier score, given true label and prediction scores.
combined_evaluators	Combine partially applies evaluation functions.
correlation_evaluator	Computes the Pearson correlation between prediction and target.
expected_calibration_error_evaluator	Computes the expected calibration error (ECE), given true label and prediction scores.
fbeta_score_evaluator	Computes the F-beta score, given true label and prediction scores.
generic_sklearn_evaluator(name_prefix, ...)	Returns an evaluator build from a metric from sklearn.metrics
hash_evaluator	Computes the hash of a pandas dataframe, filtered by hash columns.
logloss_evaluator	Computes the logloss score, given true label and prediction scores.
mean_prediction_evaluator	Computes mean for the specified column.
mse_evaluator	Computes the Mean Squared Error, given true label and predictions.
permutation_evaluator	Permutation importance evaluator.
precision_evaluator	Computes the precision score, given true label and prediction scores.
r2_evaluator	Computes the R2 score, given true label and predictions.
recall_evaluator	Computes the recall score, given true label and prediction scores.
spearman_evaluator	Computes the Spearman correlation between prediction and target.
ndcg_evaluator	Computes the Normalized Discount Cumulative Gain (NDCG) between of the original and predicted rankings: https://en.wikipedia.org/wiki/Discounted_cumulative_gain
split_evaluator	Splits the dataset into the categories in *split_col* and evaluate model performance in each split.
temporal_split_evaluator	Splits the dataset into the temporal categories by *time_col* and evaluate model performance in each split.

Splitters (fklearn.validation.splitters)

forward_stability_curve_time_splitter	Splits the data into temporal buckets with both the training and testing folds both moving forward.
k_fold_splitter	Makes K random train/test split folds for cross validation.

Continued on next page

Table 17 – continued from previous page

out_of_time_and_space_splitter	Makes K grouped train/test split folds for cross validation.
reverse_time_learning_curve_splitter	Splits the data into temporal buckets given by the specified frequency.
spatial_learning_curve_splitter	Splits the data for a spatial learning curve.
stability_curve_time_in_space_splitter	Splits the data into temporal buckets given by the specified frequency.
stability_curve_time_space_splitter	Splits the data into temporal buckets given by the specified frequency.
stability_curve_time_splitter	Splits the data into temporal buckets given by the specified frequency.
time_and_space_learning_curve_splitter	Splits the data into temporal buckets given by the specified frequency.
time_learning_curve_splitter	Splits the data into temporal buckets given by the specified frequency.

Validator (fklearn.validation.validator)

parallel_validator
validator
validator_iteration

1.3.5 Definitions

fklearn.data.datasets.**make_confounded_data**(*n: int*) → Tuple[pandas.core.frame.DataFrame, pandas.core.frame.DataFrame, pandas.core.frame.DataFrame]

Generates fake data for counterfactual experimentation. The covariants are sex, age and severity, the treatment is a binary variable, medication and the response days until recovery.

Parameters **n** (*int*) – The number of samples to generate

Returns

- **df_rnd** (*pd.DataFrame*) – A dataframe where the treatment is randomly assigned.
- **df_obs** (*pd.DataFrame*) – A dataframe with confounding.
- **df_df** (*pd.DataFrame*) – A counter factual dataframe with confounding. Same as df_obs, but with the treatment flipped.

fklearn.data.datasets.**make_tutorial_data**(*n: int*) → pandas.core.frame.DataFrame

Generates fake data for a tutorial. There are 3 numerical features (“num1”, “num3” and “num3”) and tow categorical features (“cat1” and “cat2”) sex, age and severity, the treatment is a binary variable, medication and the response days until recovery.

Parameters **n** (*int*) – The number of samples to generate

Returns **df** – A tutorial dataset

Return type pd.DataFrame

fklearn.preprocessing.rebalancing.**rebalance_by_categorical**

Resample dataset so that the result contains the same number of lines per category in categ_column.

Parameters

- **dataset** (*pandas.DataFrame*) – A Pandas' DataFrame with an categ_column column
- **categ_column** (*str*) – The name of the categorical column
- **max_lines_by_categ** (*int (default None)*) – The maximum number of lines by category. If None it will be set to the number of lines for the smallest category
- **seed** (*int (default 1)*) – Random state for consistency.

Returns **rebalanced_dataset** – A dataset with fewer lines than dataset, but with the same number of lines per category in categ_column

Return type pandas.DataFrame

fklearn.preprocessing.rebalancing.**rebalance_by_continuous**

Resample dataset so that the result contains the same number of lines per bucket in a continuous column.

Parameters

- **dataset** (*pandas.DataFrame*) – A Pandas' DataFrame with an categ_column column
- **continuous_column** (*str*) – The name of the continuous column
- **buckets** (*int*) – The number of buckets to split the continuous column into
- **max_lines_by_categ** (*int (default None)*) – The maximum number of lines by category. If None it will be set to the number of lines for the smallest category
- **by_quantile** (*bool (default False)*) – If True, uses pd.qcut instead of pd.cut to get the buckets from the continuous column
- **seed** (*int (default 1)*) – Random state for consistency.

Returns **rebalanced_dataset** – A dataset with fewer lines than dataset, but with the same number of lines per category in categ_column

Return type pandas.DataFrame

fklearn.preprocessing.splitting.**space_time_split_dataset**

Splits panel data using both ID and Time columns, resulting in four datasets

1. A training set;
2. An in training time, but out sample id hold out dataset;
3. An out of training time, but in sample id hold out dataset;
4. An out of training time and out of sample id hold out dataset.

Parameters

- **dataset** (*pandas.DataFrame*) – A Pandas' DataFrame with an Identifier Column and a Date Column. The model will be trained to predict the target column from the features.
- **train_start_date** (*str*) – A date string representing a the starting time of the training data. It should be in the same format as the Date Column in *dataset*.
- **train_end_date** (*str*) – A date string representing a the ending time of the training data. This will also be used as the start date of the holdout period if no *holdout_start_date* is given. It should be in the same format as the Date Column in *dataset*.
- **holdout_end_date** (*str*) – A date string representing a the ending time of the holdout data. It should be in the same format as the Date Column in *dataset*.
- **split_seed** (*int*) – A seed used by the random number generator.

- **space_holdout_percentage** (*float*) – The out of id holdout size as a proportion of the in id training size.
- **space_column** (*str*) – The name of the Identifier column of *dataset*.
- **time_column** (*str*) – The name of the Date column of *dataset*.
- **holdout_space** (*np.array*) – An array containing the hold out IDs. If not specified, A random subset of IDs will be selected for holdout.
- **holdout_start_date** (*str*) – A date string representing the starting time of the holdout data. If *None* is given it will be equal to *train_end_date*. It should be in the same format as the Date Column in *dataset*.

Returns

- **train_set** (*pandas.DataFrame*) – The in ID sample and in time training set.
- **intime_outspace_hdout** (*pandas.DataFrame*) – The out of ID sample and in time hold out set.
- **outime_inspace_hdout** (*pandas.DataFrame*) – The in ID sample and out of time hold out set.
- **outime_outspace_hdout** (*pandas.DataFrame*) – The out of ID sample and out of time hold out set.

fklearn.preprocessing.splitting.**stratified_split_dataset**

Splits data into a training and testing datasets such that they maintain the same class ratio of the original dataset.

Parameters

- **dataset** (*pandas.DataFrame*) – A Pandas' DataFrame with the target column. The model will be trained to predict the target column from the features.
- **target_column** (*str*) – The name of the target column of *dataset*.
- **test_size** (*float*) – Represent the proportion of the dataset to include in the test split. should be between 0.0 and 1.0.
- **random_state** (*int or None, optional (default=None)*) – If int, random_state is the seed used by the random number generator; If None, the random number generator is the RandomState instance used by *np.random*.

Returns

- **train_set** (*pandas.DataFrame*) – The train dataset sampled from the full dataset.
- **test_set** (*pandas.DataFrame*) – The test dataset sampled from the full dataset.

fklearn.preprocessing.splitting.**time_split_dataset**

Splits temporal data into a training and testing datasets such that all training data comes before the testings one.

Parameters

- **dataset** (*pandas.DataFrame*) – A Pandas' DataFrame with an Identifier Column and a Date Column. The model will be trained to predict the target column from the features.
- **train_start_date** (*str*) – A date string representing a the starting time of the training data. It should be in the same format as the Date Column in *dataset*.
- **train_end_date** (*str*) – A date string representing a the ending time of the training data. This will also be used as the start date of the holdout period if no *holdout_start_date* is given. It should be in the same format as the Date Column in *dataset*.

- **holdout_end_date** (*str*) – A date string representing a the ending time of the holdout data. It should be in the same format as the Date Column in *dataset*.
- **time_column** (*str*) – The name of the Date column of *dataset*.
- **holdout_start_date** (*str*) – A date string representing the starting time of the holdout data. If *None* is given it will be equal to *train_end_date*. It should be in the same format as the Date Column in *dataset*.

Returns

- **train_set** (*pandas.DataFrame*) – The in ID sample and in time training set.
- **test_set** (*pandas.DataFrame*) – The out of ID sample and in time hold out set.

fklearn.training.calibration.**find_thresholds_with_same_risk**

Calculate fair calibration, where for each band any sensitive factor group have the same target mean.

Parameters

- **df** (*pandas.DataFrame*) – A Pandas' DataFrame with features and target columns. The model will be trained to predict the target column from the features.
- **sensitive_factor** (*str*) – Column where we have the different group classifications that we want to have the same target mean
- **unfair_band_column** (*str*) – Column with the original bands
- **model_prediction_output** (*str*) – Risk model's output
- **target_column** (*str*) – The name of the column in *df* that should be used as target for the model. This column should be binary, since this is a classification model.
- **output_column_name** (*str*) – The name of the column with the fair bins.

Returns

- **p** (*function pandas.DataFrame -> pandas.DataFrame*) – A function that when applied to a DataFrame with the same columns as *df* returns a new DataFrame with a new column with predictions from the model.
- **new_df** (*pandas.DataFrame*) – A *df*-like DataFrame with the same columns as the input *df* plus a column with predictions from the model.
- **log** (*dict*) – A log-like Dict that stores information of the find_thresholds_with_same_risk model.

fklearn.training.calibration.**isotonic_calibration_learner**

Fits a single feature isotonic regression to the dataset.

Parameters

- **df** (*pandas.DataFrame*) – A Pandas' DataFrame with features and target columns. The model will be trained to predict the target column from the features.
- **target_column** (*str*) – The name of the column in *df* that should be used as target for the model. This column should be binary, since this is a classification model.
- **prediction_column** (*str*) – The name of the column with the uncalibrated predictions from the model.
- **output_column** (*str*) – The name of the column with the calibrated predictions from the model.
- **y_min** (*float*) – Lower bound of Isotonic Regression

- **y_max** (*float*) – Upper bound of Isotonic Regression

Returns

- **p** (*function pandas.DataFrame -> pandas.DataFrame*) – A function that when applied to a DataFrame with the same columns as *df* returns a new DataFrame with a new column with predictions from the model.
- **new_df** (*pandas.DataFrame*) – A *df*-like DataFrame with the same columns as the input *df* plus a column with predictions from the model.
- **log** (*dict*) – A log-like Dict that stores information of the Isotonic Calibration model.

fklearn.training.classification.**catboost_classification_learner**

Fits an CatBoost classifier to the dataset. It first generates a DMatrix with the specified features and labels from *df*. Then, it fits a CatBoost model to this DMatrix. Return the predict function for the model and the predictions for the input dataset.

Parameters

- **df** (*pandas.DataFrame*) – A Pandas' DataFrame with features and target columns. The model will be trained to predict the target column from the features.
- **features** (*list of str*) – A list os column names that are used as features for the model. All this names should be in *df*.
- **target** (*str*) – The name of the column in *df* that should be used as target for the model. This column should be discrete, since this is a classification model.
- **learning_rate** (*float*) – Float in the range (0, 1] Step size shrinkage used in update to prevents overfitting. After each boosting step, we can directly get the weights of new features. and eta actually shrinks the feature weights to make the boosting process more conservative. See the eta hyper-parameter in: https://catboost.ai/docs/concepts/python-reference_parameters-list.html
- **num_estimators** (*int*) – Int in the range (0, inf) Number of boosted trees to fit. See the n_estimators hyper-parameter in: https://catboost.ai/docs/concepts/python-reference_parameters-list.html
- **extra_params** (*dict, optional*) – Dictionary in the format {"hyperparameter_name" : hyperparameter_value}. Other parameters for the CatBoost model. See the list in: https://catboost.ai/docs/concepts/python-reference_catboostregressor.html If not passed, the default will be used.
- **prediction_column** (*str*) – The name of the column with the predictions from the model. If a multiclass problem, additional prediction_column_i columns will be added for i in range(0,n_classes).
- **weight_column** (*str, optional*) – The name of the column with scores to weight the data.
- **encode_extra_cols** (*bool (default: True)*) – If True, treats all columns in *df* with name pattern fklearn_feat__col==val‘ as feature columns.

Returns

- **p** (*function pandas.DataFrame -> pandas.DataFrame*) – A function that when applied to a DataFrame with the same columns as *df* returns a new DataFrame with a new column with predictions from the model.
- **new_df** (*pandas.DataFrame*) – A *df*-like DataFrame with the same columns as the input *df* plus a column with predictions from the model.

- **log** (*dict*) – A log-like Dict that stores information of the catboost_classification_learner model.

fklearn.training.classification.**lgbm_classification_learner**

Fits an LGBM classifier to the dataset.

It first generates a Dataset with the specified features and labels from *df*. Then, it fits a LGBM model to this Dataset. Return the predict function for the model and the predictions for the input dataset.

Parameters

- **df** (*pandas.DataFrame*) – A pandas DataFrame with features and target columns. The model will be trained to predict the target column from the features.
- **features** (*list of str*) – A list os column names that are used as features for the model. All this names should be in *df*.
- **target** (*str*) – The name of the column in *df* that should be used as target for the model. This column should be discrete, since this is a classification model.
- **learning_rate** (*float*) – Float in the range (0, 1] Step size shrinkage used in update to prevents overfitting. After each boosting step, we can directly get the weights of new features. and eta actually shrinks the feature weights to make the boosting process more conservative. See the learning_rate hyper-parameter in: https://github.com/Microsoft/LightGBM/blob/master/docs/Parameters.rst
- **num_estimators** (*int*) – Int in the range (0, inf) Number of boosted trees to fit. See the num_iterations hyper-parameter in: https://github.com/Microsoft/LightGBM/blob/master/docs/Parameters.rst
- **extra_params** (*dict, optional*) – Dictionary in the format {"hyperparameter_name" : hyperparameter_value}. Other parameters for the LGBM model. See the list in: https://github.com/Microsoft/LightGBM/blob/master/docs/Parameters.rst If not passed, the default will be used.
- **prediction_column** (*str*) – The name of the column with the predictions from the model.
- **weight_column** (*str, optional*) – The name of the column with scores to weight the data.
- **encode_extra_cols** (*bool (default: True)*) – If True, treats all columns in *df* with name pattern fklearn_feat__col==val‘ as feature columns.

Returns

- **p** (*function pandas.DataFrame -> pandas.DataFrame*) – A function that when applied to a DataFrame with the same columns as *df* returns a new DataFrame with a new column with predictions from the model.
- **new_df** (*pandas.DataFrame*) – A *df*-like DataFrame with the same columns as the input *df* plus a column with predictions from the model.
- **log** (*dict*) – A log-like Dict that stores information of the LGBM Classifier model.

fklearn.training.classification.**logistic_classification_learner**

Fits an logistic regression classifier to the dataset. Return the predict function for the model and the predictions for the input dataset.

Parameters

- **df** (*pandas.DataFrame*) – A Pandas' DataFrame with features and target columns. The model will be trained to predict the target column from the features.

- **features** (*list of str*) – A list os column names that are used as features for the model. All this names should be in *df*.
- **target** (*str*) – The name of the column in *df* that should be used as target for the model. This column should be discrete, since this is a classification model.
- **params** (*dict*) – The LogisticRegression parameters in the format {"par_name": param}. See: http://scikit-learn.org/stable/modules/generated/sklearn.linear_model.LogisticRegression.html
- **prediction_column** (*str*) – The name of the column with the predictions from the model. If a multiclass problem, additional prediction_column_i columns will be added for i in range(0,n_classes).
- **weight_column** (*str, optional*) – The name of the column with scores to weight the data.
- **encode_extra_cols** (*bool (default: True)*) – If True, treats all columns in *df* with name pattern fklearn_feat__col==val‘ as feature columns.

Returns

- **p** (*function pandas.DataFrame -> pandas.DataFrame*) – A function that when applied to a DataFrame with the same columns as *df* returns a new DataFrame with a new column with predictions from the model.
- **new_df** (*pandas.DataFrame*) – A *df*-like DataFrame with the same columns as the input *df* plus a column with predictions from the model.
- **log** (*dict*) – A log-like Dict that stores information of the Logistic Regression model.

fklearn.training.classification.**nlp_logistic_classification_learner**

Fits a text vectorizer (TfidfVectorizer) followed by a logistic regression (LogisticRegression).

Parameters

- **df** (*pandas.DataFrame*) – A Pandas' DataFrame with features and target columns. The model will be trained to predict the target column from the features.
- **text_feature_cols** (*list of str*) – A list of column names of the text features used for the model. All these names should be in *df*.
- **target** (*str*) – The name of the column in *df* that should be used as target for the model. This column should be discrete, since this is a classification model.
- **vectorizer_params** (*dict*) – The TfidfVectorizer parameters in the format {"par_name": param}. See: http://scikit-learn.org/stable/modules/generated/sklearn.feature_extraction.text.TfidfVectorizer.html
- **logistic_params** (*dict*) – The LogisticRegression parameters in the format {"par_name": param}. See: http://scikit-learn.org/stable/modules/generated/sklearn.linear_model.LogisticRegression.html
- **prediction_column** (*str*) – The name of the column with the predictions from the model.

Returns

- **p** (*function pandas.DataFrame -> pandas.DataFrame*) – A function that when applied to a DataFrame with the same columns as *df* returns a new DataFrame with a new column with predictions from the model.
- **new_df** (*pandas.DataFrame*) – A *df*-like DataFrame with the same columns as the input *df* plus a column with predictions from the model.

- **log** (*dict*) – A log-like Dict that stores information of the NLP Logistic Regression model.

fklearn.training.classification.**xgb_classification_learner**

Fits an XGBoost classifier to the dataset. It first generates a DMatrix with the specified features and labels from *df*. Then, it fits a XGBoost model to this DMatrix. Return the predict function for the model and the predictions for the input dataset.

Parameters

- **df** (*pandas.DataFrame*) – A Pandas' DataFrame with features and target columns. The model will be trained to predict the target column from the features.
- **features** (*list of str*) – A list os column names that are used as features for the model. All this names should be in *df*.
- **target** (*str*) – The name of the column in *df* that should be used as target for the model. This column should be discrete, since this is a classification model.
- **learning_rate** (*float*) – Float in the range (0, 1] Step size shrinkage used in update to prevents overfitting. After each boosting step, we can directly get the weights of new features. and eta actually shrinks the feature weights to make the boosting process more conservative. See the eta hyper-parameter in: http://xgboost.readthedocs.io/en/latest/parameter.html
- **num_estimators** (*int*) – Int in the range (0, inf) Number of boosted trees to fit. See the n_estimators hyper-parameter in: http://xgboost.readthedocs.io/en/latest/python/python_api.html
- **extra_params** (*dict, optional*) – Dictionary in the format {"hyperparameter_name" : hyperparameter_value}. Other parameters for the XGBoost model. See the list in: http://xgboost.readthedocs.io/en/latest/parameter.html If not passed, the default will be used.
- **prediction_column** (*str*) – The name of the column with the predictions from the model. If a multiclass problem, additional prediction_column_i columns will be added for i in range(0,n_classes).
- **weight_column** (*str, optional*) – The name of the column with scores to weight the data.
- **encode_extra_cols** (*bool (default: True)*) – If True, treats all columns in *df* with name pattern fklearn_feat__col==val' as feature columns.

Returns

- **p** (*function pandas.DataFrame -> pandas.DataFrame*) – A function that when applied to a DataFrame with the same columns as *df* returns a new DataFrame with a new column with predictions from the model.
- **new_df** (*pandas.DataFrame*) – A *df*-like DataFrame with the same columns as the input *df* plus a column with predictions from the model.
- **log** (*dict*) – A log-like Dict that stores information of the XGboost Classifier model.

fklearn.training.ensemble.**xgb_octopus_classification_learner**

Octopus ensemble allows you to inject domain specific knowledge to force a split in an initial feature, instead of assuming the tree model will do that intelligent split on its own. It works by first defining a split on your dataset and then training one individual model in each separated dataset.

Parameters

- **train_set** (*pd.DataFrame*) – A Pandas' DataFrame with features, target columns and a splitting column that must be categorical.

- **learning_rate_by_bin** (*dict*) – A dictionary of learning rate in the XGBoost model to use in each model split. Ex: if you want to split your training by tenure and you have a tenure column with integer values [1,2,3,...,12], you have to specify a list of learning rates for each split:

```
{
    1: 0.08,
    2: 0.08,
    ...
    12: 0.1
}
```

- **num_estimators_by_bin** (*dict*) – A dictionary of number of tree estimators in the XGBoost model to use in each model split. Ex: if you want to split your training by tenure and you have a tenure column with integer values [1,2,3,...,12], you have to specify a list of estimators for each split:

```
{
    1: 300,
    2: 250,
    ...
    12: 300
}
```

- **extra_params_by_bin** (*dict*) – A dictionary of extra parameters dictionaries in the XGBoost model to use in each model split. Ex: if you want to split your training by tenure and you have a tenure column with integer values [1,2,3,...,12], you have to specify a list of extra parameters for each split:

```
{
    1: {
        'reg_alpha': 0.0,
        'colsample_bytree': 0.4,
        ...
        'colsample_bylevel': 0.8
        }
    2: {
        'reg_alpha': 0.1,
        'colsample_bytree': 0.6,
        ...
        'colsample_bylevel': 0.4
        }
    ...
    12: {
        'reg_alpha': 0.0,
        'colsample_bytree': 0.7,
        ...
        'colsample_bylevel': 1.0
        }
}
```

- **features_by_bin** (*dict*) – A dictionary of features to use in each model split. Ex: if you want to split your training by tenure and you have a tenure column with integer values [1,2,3,...,12], you have to specify a list of features for each split:

```
{
    1: [feature-1, feature-2, feature-3, ...],
```

(continues on next page)

(continued from previous page)

```
    2: [feature-1, feature-3, feature-5, ...],
    ...
    12: [feature-2, feature-4, feature-8, ...]
}
```

- **train_split_col** (*str*) – The name of the categorical column where the model will make the splits. Ex: if you want to split your training by tenure, you can have a categorical column called "tenure".
- **train_split_bins** (*list*) – A list with the actual values of the categories from the *train_split_col*. Ex: if you want to split your training by tenure and you have a tenure column with integer values [1,2,3,...,12] you can pass this list and you will split your training into 12 different models.
- **nthread** (*int*) – Number of threads for the XGBoost learners.
- **target_column** (*str*) – The name of the target column.
- **prediction_column** (*str*) – The name of the column with the predictions from the model.

Returns

- **p** (*function pandas.DataFrame -> pandas.DataFrame*) – A function that when applied to a DataFrame with the same columns as *df* returns a new DataFrame with a new column with predictions from the model.
- **new_df** (*pandas.DataFrame*) – A *df*-like DataFrame with the same columns as the input *df* plus a column with predictions from the model.
- **log** (*dict*) – A log-like Dict that stores information of the Octopus XGB Classifier model.

fklearn.training.imputation.**imputer**

Fits a missing value imputer to the dataset.

Parameters

- **df** (*pandas.DataFrame*) – A Pandas' DataFrame with columns to impute missing values. It must contain all columns listed in *columns_to_impute*
- **columns_to_impute** (*List of strings*) – A list of names of the columns for missing value imputation.
- **impute_strategy** (*String, (default="median")*) – The imputation strategy. - If "mean", then replace missing values using the mean along the axis. - If "median", then replace missing values using the median along the axis. - If "most_frequent", then replace missing using the most frequent value along the axis.
- **placeholder_value** (*Any, (default=None)*) – if not None, use this as default value when some features only contains NA values on training. For transformation, NA values on those features will be replaced by *fill_value*.

Returns

- **p** (*function pandas.DataFrame -> pandas.DataFrame*) – A function that when applied to a DataFrame with the same columns as *df* returns a new DataFrame with a new column with predictions from the model.
- **new_df** (*pandas.DataFrame*) – A *df*-like DataFrame with the same columns as the input *df* plus a column with predictions from the model.
- **log** (*dict*) – A log-like Dict that stores information of the SimpleImputer model.

fklearn.training.imputation.**placeholder_imputer**

Fills missing values with a fixed value.

Parameters

- **df** (*pandas.DataFrame*) – A Pandas' DataFrame with columns to fill missing values. It must contain all columns listed in *columns_to_impute*
- **columns_to_impute** (*List of strings*) – A list of names of the columns for filling missing value.
- **placeholder_value** (*Any, (default=-999)*) – The value used to fill in missing values.

Returns

- **p** (*function pandas.DataFrame -> pandas.DataFrame*) – A function that when applied to a DataFrame with the same columns as *df* returns a new DataFrame with a new column with predictions from the model.
- **new_df** (*pandas.DataFrame*) – A *df*-like DataFrame with the same columns as the input *df* plus a column with predictions from the model.
- **log** (*dict*) – A log-like Dict that stores information of the Placeholder SimpleImputer model.

fklearn.training.pipeline.**build_pipeline**(**learners*, *has_repeated_learners: bool = False*) → Callable[pandas.core.frame.DataFrame, Tuple[Callable[pandas.core.frame.DataFrame, pandas.core.frame.DataFrame], pandas.core.frame.DataFrame, Dict[str, Dict[str, Any]]]]

Builds a pipeline of different chained learners functions with the possibility of using keyword arguments in the predict functions of the pipeline.

Say you have two learners, you create a pipeline with *pipeline = build_pipeline(learner1, learner2)*. Those learners must be functions with just one unfilled argument (the dataset itself).

Then, you train the pipeline with *predict_fn, transformed_df, logs = pipeline(df)*, which will be like applying the learners in the following order: *learner2(learner1(df))*.

Finally, you predict on different datasets with *pred_df = predict_fn(new_df)*, with optional kwargs. For example, if you have XGBoost or LightGBM, you can get SHAP values with *predict_fn(new_df, apply_shap=True)*.

Parameters

- **learners** (*partially-applied learner functions.*) –
- **has_repeated_learners** (*bool*) – Boolean value indicating wheter the pipeline contains learners with the same name or not.

Returns

- **p** (*function pandas.DataFrame, **kwargs -> pandas.DataFrame*) – A function that when applied to a DataFrame will apply all learner functions in sequence, with optional kwargs.
- **new_df** (*pandas.DataFrame*) – A DataFrame that is the result of applying all learner function in sequence.
- **log** (*dict*) – A log-like Dict that stores information of all learner functions.

fklearn.training.regression.**catboost_regressor_learner**

Fits an CatBoost regressor to the dataset. It first generates a Pool with the specified features and labels from *df*. Then it fits a CatBoost model to this Pool. Return the predict function for the model and the predictions for the input dataset.

Parameters

- **df** (*pandas.DataFrame*) – A Pandas' DataFrame with features and target columns. The model will be trained to predict the target column from the features.
- **features** (*list of str*) – A list os column names that are used as features for the model. All this names should be in *df*.
- **target** (*str*) – The name of the column in *df* that should be used as target for the model. This column should be numerical and continuous, since this is a regression model.
- **learning_rate** (*float*) – Float in range [0,1]. Step size shrinkage used in update to prevents overfitting. After each boosting step, we can directly get the weights of new features. and eta actually shrinks the feature weights to make the boosting process more conservative. See the eta hyper-parameter in: https://catboost.ai/docs/concepts/python-reference_parameters-list.html
- **num_estimators** (*int*) – Int in range [0, inf] Number of boosted trees to fit. See the n_estimators hyper-parameter in: https://catboost.ai/docs/concepts/python-reference_parameters-list.html
- **extra_params** (*dict, optional*) – Dictionary in the format {"hyperparameter_name" : hyperparameter_value. Other parameters for the CatBoost model. See the list in: https://catboost.ai/docs/concepts/python-reference_catboostregressor.html If not passed, the default will be used.
- **prediction_column** (*str*) – The name of the column with the predictions from the model.
- **weight_column** (*str, optional*) – The name of the column with scores to weight the data.

Returns

- **p** (*function pandas.DataFrame -> pandas.DataFrame*) – A function that when applied to a DataFrame with the same columns as *df* returns a new DataFrame with a new column with predictions from the model.
- **new_df** (*pandas.DataFrame*) – A *df*-like DataFrame with the same columns as the input *df* plus a column with predictions from the model.
- **log** (*dict*) – A log-like Dict that stores information of the CatBoostRegressor model.

fklearn.training.regression.**custom_supervised_model_learner**

Fits a custom model to the dataset. Return the predict function, the predictions for the input dataset and a log describing the model.

Parameters

- **df** (*pandas.DataFrame*) – A Pandas' DataFrame with features and target columns. The model will be trained to predict the target column from features.
- **features** (*list of str*) – A list os column names that are used as features for the model. All this names should be in *df*.
- **target** (*str*) – The name of the column in *df* that should be used as target for the model.
- **model** (*Object*) – Machine learning model to be used for regression or clasisfication. model object must have ".fit" attribute to train the data. For classification problems, it also needs ".predict_proba" attribute. For regression problemsm it needs ".predict" attribute.
- **supervised_type** (*str*) – Type of supervised learning to be used The options are: 'classification' or 'regression'

- **log** (*Dict[str, Dict]*) – Log with additional information of the custom model used. It must start with just one element with the model name.
- **prediction_column** (*str*) – The name of the column with the predictions from the model. For classification problems, all probabilities wiill be added: for i in range(0,n_classes). For regression just prediction_column will be added.

Returns

- **p** (*function pandas.DataFrame -> pandas.DataFrame*) – A function that when applied to a DataFrame with the same columns as *df* returns a new DataFrame with a new column with predictions from the model.
- **new_df** (*pandas.DataFrame*) – A *df*-like DataFrame with the same columns as the input *df* plus a column with predictions from the model.
- **log** (*dict*) – A log-like Dict that stores information of the Custom Supervised Model Learner model.

fklearn.training.regression.**gp_regression_learner**

Fits an gaussian process regressor to the dataset.

Parameters

- **df** (*pandas.DataFrame*) – A Pandas' DataFrame with features and target columns. The model will be trained to predict the target column from the features.
- **features** (*list of str*) – A list os column names that are used as features for the model. All this names should be in *df*.
- **target** (*str*) – The name of the column in *df* that should be used as target for the model. This column should be numerical and continuous, since this is a regression model.
- **kernel** (*sklearn.gaussian_process.kernels*) – The kernel specifying the covariance function of the GP. If None is passed, the kernel "1.0 * RBF(1.0)" is used as default. Note that the kernel's hyperparameters are optimized during fitting.
- **alpha** (*float*) – Value added to the diagonal of the kernel matrix during fitting. Larger values correspond to increased noise level in the observations. This can also prevent a potential numerical issue during fitting, by ensuring that the calculated values form a positive definite matrix.
- **extra_variance** (*float*) – The amount of extra variance to scale to the predictions in standard deviations. If left as the default "fit", Uses the standard deviation of the target.
- **return_std** (*bool*) – If True, the standard-deviation of the predictive distribution at the query points is returned along with the mean.
- **extra_params** (*dict {"hyperparameter_name" : hyperparameter_value}, optional*) – Other parameters for the GaussianProcessRegressor model. See the list in: http://scikit-learn.org/stable/modules/generated/sklearn.gaussian_process.GaussianProcessRegressor.html If not passed, the default will be used.
- **prediction_column** (*str*) – The name of the column with the predictions from the model.
- **encode_extra_cols** (*bool (default: True)*) – If True, treats all columns in *df* with name pattern fklearn_feat__col==val' as feature columns.

Returns

- **p** (*function pandas.DataFrame -> pandas.DataFrame*) – A function that when applied to a DataFrame with the same columns as *df* returns a new DataFrame with a new column with predictions from the model.
- **new_df** (*pandas.DataFrame*) – A *df*-like DataFrame with the same columns as the input *df* plus a column with predictions from the model.
- **log** (*dict*) – A log-like Dict that stores information of the Gaussian Process Regressor model.

fklearn.training.regression.**lgbm_regression_learner**

Fits an LGBM regressor to the dataset.

It first generates a Dataset with the specified features and labels from *df*. Then, it fits a LGBM model to this Dataset. Return the predict function for the model and the predictions for the input dataset.

Parameters

- **df** (*pandas.DataFrame*) – A Pandas' DataFrame with features and target columns. The model will be trained to predict the target column from the features.
- **features** (*list of str*) – A list os column names that are used as features for the model. All this names should be in *df*.
- **target** (*str*) – The name of the column in *df* that should be used as target for the model. This column should be binary, since this is a classification model.
- **learning_rate** (*float*) – Float in the range (0, 1] Step size shrinkage used in update to prevents overfitting. After each boosting step, we can directly get the weights of new features. and eta actually shrinks the feature weights to make the boosting process more conservative. See the learning_rate hyper-parameter in: https://github.com/Microsoft/LightGBM/blob/master/docs/Parameters.rst
- **num_estimators** (*int*) – Int in the range (0, inf) Number of boosted trees to fit. See the num_iterations hyper-parameter in: https://github.com/Microsoft/LightGBM/blob/master/docs/Parameters.rst
- **extra_params** (*dict, optional*) – Dictionary in the format {"hyperparameter_name" : hyperparameter_value}. Other parameters for the LGBM model. See the list in: https://github.com/Microsoft/LightGBM/blob/master/docs/Parameters.rst If not passed, the default will be used.
- **prediction_column** (*str*) – The name of the column with the predictions from the model.
- **weight_column** (*str, optional*) – The name of the column with scores to weight the data.
- **encode_extra_cols** (*bool (default: True)*) – If True, treats all columns in *df* with name pattern fklearn_feat__col==val‘ as feature columns.

Returns

- **p** (*function pandas.DataFrame -> pandas.DataFrame*) – A function that when applied to a DataFrame with the same columns as *df* returns a new DataFrame with a new column with predictions from the model.
- **new_df** (*pandas.DataFrame*) – A *df*-like DataFrame with the same columns as the input *df* plus a column with predictions from the model.
- **log** (*dict*) – A log-like Dict that stores information of the LGBM Regressor model.

fklearn.training.regression.**linear_regression_learner**

Fits an linear regression classifier to the dataset. Return the predict function for the model and the predictions for the input dataset.

Parameters

- **df** (*pandas.DataFrame*) – A Pandas' DataFrame with features and target columns. The model will be trained to predict the target column from the features.
- **features** (*list of str*) – A list os column names that are used as features for the model. All this names should be in *df*.
- **target** (*str*) – The name of the column in *df* that should be used as target for the model. This column should be continuous, since this is a regression model.
- **params** (*dict*) – The LinearRegression parameters in the format {"par_name": param}. See: http://scikit-learn.org/stable/modules/generated/sklearn.linear_model.LinearRegression.html
- **prediction_column** (*str*) – The name of the column with the predictions from the model.
- **weight_column** (*str, optional*) – The name of the column with scores to weight the data.
- **encode_extra_cols** (*bool (default: True)*) – If True, treats all columns in *df* with name pattern fklearn_feat__col==val' as feature columns.

Returns

- **p** (*function pandas.DataFrame -> pandas.DataFrame*) – A function that when applied to a DataFrame with the same columns as *df* returns a new DataFrame with a new column with predictions from the model.
- **new_df** (*pandas.DataFrame*) – A *df*-like DataFrame with the same columns as the input *df* plus a column with predictions from the model.
- **log** (*dict*) – A log-like Dict that stores information of the Linear Regression model.

fklearn.training.regression.**xgb_regression_learner**

Fits an XGBoost regressor to the dataset. It first generates a DMatrix with the specified features and labels from *df*. Then it fits a XGBoost model to this DMatrix. Return the predict function for the model and the predictions for the input dataset.

Parameters

- **df** (*pandas.DataFrame*) – A Pandas' DataFrame with features and target columns. The model will be trained to predict the target column from the features.
- **features** (*list of str*) – A list os column names that are used as features for the model. All this names should be in *df*.
- **target** (*str*) – The name of the column in *df* that should be used as target for the model. This column should be numerical and continuous, since this is a regression model.
- **learning_rate** (*float*) – Float in range [0,1]. Step size shrinkage used in update to prevents overfitting. After each boosting step, we can directly get the weights of new features. and eta actually shrinks the feature weights to make the boosting process more conservative. See the eta hyper-parameter in: http://xgboost.readthedocs.io/en/latest/parameter.html

- **num_estimators** (*int*) – Int in range [0, inf] Number of boosted trees to fit. See the n_estimators hyper-parameter in: http://xgboost.readthedocs.io/en/latest/python/python_api.html
- **extra_params** (*dict, optional*) – Dictionary in the format {"hyperparameter_name" : hyperparameter_value. Other parameters for the XGBoost model. See the list in: http://xgboost.readthedocs.io/en/latest/parameter.html If not passed, the default will be used.
- **prediction_column** (*str*) – The name of the column with the predictions from the model.
- **weight_column** (*str, optional*) – The name of the column with scores to weight the data.
- **encode_extra_cols** (*bool (default: True)*) – If True, treats all columns in *df* with name pattern fklearn_feat__col==val' as feature columns.

Returns

- **p** (*function pandas.DataFrame -> pandas.DataFrame*) – A function that when applied to a DataFrame with the same columns as *df* returns a new DataFrame with a new column with predictions from the model.
- **new_df** (*pandas.DataFrame*) – A *df*-like DataFrame with the same columns as the input *df* plus a column with predictions from the model.
- **log** (*dict*) – A log-like Dict that stores information of the XGboost Regressor model.

fklearn.training.transformation.**apply_replacements**(*df: pandas.core.frame.DataFrame, columns: List[str], vec: Dict[str, Dict], replace_unseen: Any*) → pandas.core.frame.DataFrame

Base function to apply the replacements values found on the "vec" vectors into the df DataFrame.

Parameters

- **df** (*pandas.DataFrame*) – A Pandas DataFrame containing the data to be replaced.
- **columns** (*list of str*) – The df columns names to perform the replacements.
- **vec** (*dict*) – A dict mapping a col to dict mapping a value to its replacement. For example: vec = {"feature1": {1: 2, 3: 5, 6: 8}}
- **replace_unseen** (*Any*) – Default value to replace when original value is not present in the *vec* dict for the feature

fklearn.training.transformation.**capper**(*df: pandas.core.frame.DataFrame = '__no__default__', columns_to_cap: List[str] = '__no__default__', precomputed_caps: Dict[str, float] = None*) → Union[Callable, Tuple[Callable[pandas.core.frame.DataFrame, pandas.core.frame.DataFrame], pandas.core.frame.DataFrame, Dict[str, Dict[str, Any]]]]

Learns the maximum value for each of the *columns_to_cap* and used that as the cap for those columns. If precomputed caps are passed, the function uses that as the cap value instead of computing the maximum.

Parameters

- **df** (*pandas.DataFrame*) – A Pandas' DataFrame that must contain *columns_to_cap* columns.

- **columns_to_cap** (*list of str*) – A list os column names that should be caped.
- **precomputed_caps** (*dict*) – A dictionary on the format {"column_name" : cap_value}. That maps column names to pre computed cap values

Returns

- **p** (*function pandas.DataFrame -> pandas.DataFrame*) – A function that when applied to a DataFrame with the same columns as *df* returns a new DataFrame with a new column with predictions from the model.
- **new_df** (*pandas.DataFrame*) – A *df*-like DataFrame with the same columns as the input *df* plus a column with predictions from the model.
- **log** (*dict*) – A log-like Dict that stores information of the Capper model.

fklearn.training.transformation.**count_categorizer**(*df: pandas.core.frame.DataFrame = '__no__default__', columns_to_categorize: List[str] = '__no__default__', replace_unseen: int = -1, store_mapping: bool = False*) → Union[Callable, Tuple[Callable[pandas.core.frame.DataFrame, pandas.core.frame.DataFrame], pandas.core.frame.DataFrame, Dict[str, Dict[str, Any]]]]

Replaces categorical variables by count.

The default behaviour is to replace the original values. To store the original values in a new column, specify *prefix* or *suffix* in the parameters, or specify a dictionary with the desired column mapping using the *columns_mapping* parameter.

Parameters

- **df** (*pandas.DataFrame*) – A Pandas' DataFrame that must contain *columns_to_categorize* columns.
- **columns_to_categorize** (*list of str*) – A list of categorical column names.
- **replace_unseen** (*int*) – The value to impute unseen categories.
- **store_mapping** (*bool (default: False)*) – Whether to store the feature value -> integer dictionary in the log

Returns

- **p** (*function pandas.DataFrame -> pandas.DataFrame*) – A function that when applied to a DataFrame with the same columns as *df* returns a new DataFrame with a new column with predictions from the model.
- **new_df** (*pandas.DataFrame*) – A *df*-like DataFrame with the same columns as the input *df* plus a column with predictions from the model.
- **log** (*dict*) – A log-like Dict that stores information of the Count Categorizer model.

fklearn.training.transformation.**custom_transformer**(*df: pandas.core.frame.DataFrame = '__no__default__', columns_to_transform: List[str] = '__no__default__', transformation_function: Callable[pandas.core.frame.DataFrame, pandas.core.frame.DataFrame] = '__no__default__', is_vectorized: bool = False*) → Union[Callable, Tuple[Callable[pandas.core.frame.DataFrame, pandas.core.frame.DataFrame], pandas.core.frame.DataFrame, Dict[str, Dict[str, Any]]]]

Applies a custom function to the desired columns.

The default behaviour is to replace the original values. To store the original values in a new column, specify *prefix* or *suffix* in the parameters, or specify a dictionary with the desired column mapping using the *columns_mapping* parameter.

Parameters

- **df** (*pandas.DataFrame*) – A Pandas' DataFrame that must contain *columns*
- **columns_to_transform** (*list of str*) – A list of column names that will remain in the dataframe during training time (fit)
- **transformation_function** (*function(pandas.DataFrame) -> pandas.DataFrame*) – A function that receives a DataFrame as input, performs a transformation on its columns and returns another DataFrame.

Returns

- **p** (*function pandas.DataFrame -> pandas.DataFrame*) – A function that when applied to a DataFrame with the same columns as *df* returns a new DataFrame with a new column with predictions from the model.
- **new_df** (*pandas.DataFrame*) – A *df*-like DataFrame with the same columns as the input *df* plus a column with predictions from the model.
- **log** (*dict*) – A log-like Dict that stores information of the Custom Transformer model.

fklearn.training.transformation.**discrete_ecdfer**

Learns an Empirical Cumulative Distribution Function from the specified column in the input DataFrame. It is usually used in the prediction column to convert a predicted probability into a score from 0 to 1000.

Parameters

- **df** (*Pandas' pandas.DataFrame*) – A Pandas' DataFrame that must contain a *prediction_column* columns.
- **ascending** (*bool*) – Whether to compute an ascending ECDF or a descending one.
- **prediction_column** (*str*) – The name of the column in *df* to learn the ECDF from.
- **ecdf_column** (*str*) – The name of the new ECDF column added by this function.
- **max_range** (*int*) –

  The maximum value for the ECDF. It will go will go from 0 to max_range.
- **round_method** (*Callable*) – A function perform the round of transformed values for ex: (int, ceil, floor, round)

Returns

- **p** (*function pandas.DataFrame -> pandas.DataFrame*) – A function that when applied to a DataFrame with the same columns as *df* returns a new DataFrame with a new column with predictions from the model.
- **new_df** (*pandas.DataFrame*) – A *df*-like DataFrame with the same columns as the input *df* plus a column with predictions from the model.
- **log** (*dict*) – A log-like Dict that stores information of the Discrete ECDFer model.

fklearn.training.transformation.**ecdfer**

Learns an Empirical Cumulative Distribution Function from the specified column in the input DataFrame. It is usually used in the prediction column to convert a predicted probability into a score from 0 to 1000.

Parameters

- **df** (*Pandas' pandas.DataFrame*) – A Pandas' DataFrame that must contain a *prediction_column* columns.
- **ascending** (*bool*) – Whether to compute an ascending ECDF or a descending one.
- **prediction_column** (*str*) – The name of the column in *df* to learn the ECDF from.
- **ecdf_column** (*str*) – The name of the new ECDF column added by this function
- **max_range** (*int*) –

  The maximum value for the ECDF. It will go will go from 0 to max_range.

Returns

- **p** (*function pandas.DataFrame -> pandas.DataFrame*) – A function that when applied to a DataFrame with the same columns as *df* returns a new DataFrame with a new column with predictions from the model.
- **new_df** (*pandas.DataFrame*) – A *df*-like DataFrame with the same columns as the input *df* plus a column with predictions from the model.
- **log** (*dict*) – A log-like Dict that stores information of the ECDFer model.

fklearn.training.transformation.**floorer**(*df: pandas.core.frame.DataFrame = '__no__default__', columns_to_floor: List[str] = '__no__default__', precomputed_floors: Dict[str, float] = None*) → Union[Callable, Tuple[Callable[pandas.core.frame.DataFrame, pandas.core.frame.DataFrame], pandas.core.frame.DataFrame, Dict[str, Dict[str, Any]]]]

Learns the minimum value for each of the *columns_to_floor* and used that as the floot for those columns. If precomputed floors are passed, the function uses that as the cap value instead of computing the minimun.

Parameters

- **df** (*pandas.DataFrame*) – A Pandas' DataFrame that must contain *columns_to_floor* columns.
- **columns_to_floor** (*list of str*) – A list os column names that should be floored.
- **precomputed_floors** (*dict*) – A dictionary on the format {"column_name" : floor_value} that maps column names to pre computed floor values

Returns

- **p** (*function pandas.DataFrame -> pandas.DataFrame*) – A function that when applied to a DataFrame with the same columns as *df* returns a new DataFrame with a new column with predictions from the model.
- **new_df** (*pandas.DataFrame*) – A *df*-like DataFrame with the same columns as the input *df* plus a column with predictions from the model.
- **log** (*dict*) – A log-like Dict that stores information of the Floorer model.

fklearn.training.transformation.**label_categorizer**(*df: pandas.core.frame.DataFrame = '__no__default__', columns_to_categorize: List[str] = '__no__default__', replace_unseen: Union[str, float] = nan, store_mapping: bool = False*) → Union[Callable, Tuple[Callable[pandas.core.frame.DataFrame, pandas.core.frame.DataFrame], pandas.core.frame.DataFrame, Dict[str, Dict[str, Any]]]]

Replaces categorical variables with a numeric identifier.

The default behaviour is to replace the original values. To store the original values in a new column, specify *prefix* or *suffix* in the parameters, or specify a dictionary with the desired column mapping using the *columns_mapping* parameter.

Parameters

- **df** (*pandas.DataFrame*) – A Pandas' DataFrame that must contain *columns_to_categorize* columns.
- **columns_to_categorize** (*list of str*) – A list of categorical column names.
- **replace_unseen** (*int, str, float, or nan*) – The value to impute unseen categories.
- **store_mapping** (*bool (default: False)*) – Whether to store the feature value -> integer dictionary in the log

Returns

- **p** (*function pandas.DataFrame -> pandas.DataFrame*) – A function that when applied to a DataFrame with the same columns as *df* returns a new DataFrame with a new column with predictions from the model.
- **new_df** (*pandas.DataFrame*) – A *df*-like DataFrame with the same columns as the input *df* plus a column with predictions from the model.
- **log** (*dict*) – A log-like Dict that stores information of the Label Categorizer model.

fklearn.training.transformation.**missing_warner**

Creates a new column to warn about rows that columns that don't have missing in the training set but have missing on the scoring

Parameters

- **df** (*pandas.DataFrame*) – A Pandas' DataFrame.
- **cols_list** (*list of str*) – List of columns to consider when evaluating missingness
- **new_column_name** (*str*) – Name of the column created to alert the existence of missing values

Returns

- **p** (*function pandas.DataFrame -> pandas.DataFrame*) – A function that when applied to a DataFrame with the same columns as *df* returns a new DataFrame with a new column with predictions from the model.
- **new_df** (*pandas.DataFrame*) – A *df*-like DataFrame with the same columns as the input *df* plus a column with predictions from the model.
- **log** (*dict*) – A log-like Dict that stores information of the Missing Alerter model.

fklearn.training.transformation.**null_injector**

Injects null into columns

Parameters

- **df** (*pandas.DataFrame*) – A Pandas' DataFrame that must contain *columns_to_inject* as columns
- **columns_to_inject** (*list of str*) – A list of features to inject nulls. If groups is not None it will be ignored.
- **proportion** (*float*) – Proportion of nulls to inject in the columns.
- **groups** (*list of list of str (default = None)*) – A list of group of features. If not None, feature in the same group will be set to NaN together.
- **seed** (*int*) – Random seed for consistency.

Returns

- **p** (*function pandas.DataFrame -> pandas.DataFrame*) – A function that when applied to a DataFrame with the same columns as *df* returns a new DataFrame with a new column with predictions from the model.
- **new_df** (*pandas.DataFrame*) – A *df*-like DataFrame with the same columns as the input *df* plus a column with predictions from the model.
- **log** (*dict*) – A log-like Dict that stores information of the Null Injector model.

fklearn.training.transformation.**onehot_categorizer**(*df: pandas.core.frame.DataFrame = '__no__default__', columns_to_categorize: List[str] = '__no__default__', hardcode_nans: bool = False, drop_first_column: bool = False, store_mapping: bool = False*) → Union[Callable, Tuple[Callable[pandas.core.frame.DataFrame, pandas.core.frame.DataFrame], pandas.core.frame.DataFrame, Dict[str, Dict[str, Any]]]]

Onehot encoding on categorical columns. Encoded columns are removed and substituted by columns named *fklearn_feat__col==val*, where *col* is the name of the column and *val* is one of the values the feature can assume.

The default behaviour is to replace the original values. To store the original values in a new column, specify *prefix* or *suffix* in the parameters, or specify a dictionary with the desired column mapping using the *columns_mapping* parameter.

Parameters

- **df** (*pd.DataFrame*) – A Pandas' DataFrame that must contain *columns_to_categorize* columns.

- **columns_to_categorize** (*list of str*) – A list of categorical column names. Must be non-empty.
- **hardcode_nans** (*bool*) – Hardcodes an extra column with: 1 if nan or unseen else 0.
- **drop_first_column** (*bool*) – Drops the first column to create (k-1)-sized one-hot arrays for k features per categorical column. Can be used to avoid colinearity.
- **store_mapping** (*bool (default: False)*) – Whether to store the feature value -> integer dictionary in the log

Returns

- **p** (*function pandas.DataFrame -> pandas.DataFrame*) – A function that when applied to a DataFrame with the same columns as *df* returns a new DataFrame with a new column with predictions from the model.
- **new_df** (*pandas.DataFrame*) – A *df*-like DataFrame with the same columns as the input *df* plus a column with predictions from the model.
- **log** (*dict*) – A log-like Dict that stores information of the Onehot Categorizer model.

fklearn.training.transformation.**prediction_ranger**

Caps and floors the specified prediction column to a set range.

Parameters

- **df** (*pandas.DataFrame*) – A Pandas' DataFrame that must contain a *prediction_column* columns.
- **prediction_min** (*float*) – The floor for the prediction.
- **prediction_max** (*float*) – The cap for the prediction.
- **prediction_column** (*str*) – The name of the column in *df* to cap and floor

Returns

- **p** (*function pandas.DataFrame -> pandas.DataFrame*) – A function that when applied to a DataFrame with the same columns as *df* returns a new DataFrame with a new column with predictions from the model.
- **new_df** (*pandas.DataFrame*) – A *df*-like DataFrame with the same columns as the input *df* plus a column with predictions from the model.
- **log** (*dict*) – A log-like Dict that stores information of the Prediction Ranger model.

fklearn.training.transformation.**quantile_biner**(*df: pandas.core.frame.DataFrame = '__no__default__', columns_to_bin: List[str] = '__no__default__', q: int = 4, right: bool = False*) → Union[Callable, Tuple[Callable[pandas.core.frame.DataFrame, pandas.core.frame.DataFrame], pandas.core.frame.DataFrame, Dict[str, Dict[str, Any]]]]

Discretize continuous numerical columns into its quantiles. Uses pandas.qcut to find the bins and then numpy.digitize to fit the columns into bins.

The default behaviour is to replace the original values. To store the original values in a new column, specify *prefix* or *suffix* in the parameters, or specify a dictionary with the desired column mapping using the *columns_mapping* parameter.

Parameters

- **df** (*pandas.DataFrame*) – A Pandas' DataFrame that must contain *columns_to_categorize* columns.
- **columns_to_bin** (*list of str*) – A list of numerical column names.
- **q** (*int*) – Number of quantiles. 10 for deciles, 4 for quartiles, etc. Alternately array of quantiles, e.g. [0, .25, .5, .75, 1.] for quartiles. See https://pandas.pydata.org/pandas-docs/stable/generated/pandas.qcut.html
- **right** (*bool*) – Indicating whether the intervals include the right or the left bin edge. Default behavior is (right==False) indicating that the interval does not include the right edge. The left bin end is open in this case, i.e., bins[i-1] <= x < bins[i] is the default behavior for monotonically increasing bins. See https://docs.scipy.org/doc/numpy-1.13.0/reference/generated/numpy.digitize.html

Returns

- **p** (*function pandas.DataFrame -> pandas.DataFrame*) – A function that when applied to a DataFrame with the same columns as *df* returns a new DataFrame with a new column with predictions from the model.
- **new_df** (*pandas.DataFrame*) – A *df*-like DataFrame with the same columns as the input *df* plus a column with predictions from the model.
- **log** (*dict*) – A log-like Dict that stores information of the Quantile Biner model.

fklearn.training.transformation.**rank_categorical**(*df: pandas.core.frame.DataFrame = '__no__default__', columns_to_rank: List[str] = '__no__default__', replace_unseen: Union[str, float] = nan, store_mapping: bool = False*) → Union[Callable, Tuple[Callable[[pandas.core.frame.DataFrame, pandas.core.frame.DataFrame], pandas.core.frame.DataFrame, Dict[str, Dict[str, Any]]]]]

Rank categorical features by their frequency in the train set.

The default behaviour is to replace the original values. To store the original values in a new column, specify *prefix* or *suffix* in the parameters, or specify a dictionary with the desired column mapping using the *columns_mapping* parameter.

Parameters

- **df** (*Pandas' DataFrame*) – A Pandas' DataFrame that must contain a *prediction_column* columns.
- **columns_to_rank** (*list of str*) – The df columns names to perform the rank.
- **replace_unseen** (*int, str, float, or nan*) – The value to impute unseen categories.
- **store_mapping** (*bool (default: False)*) – Whether to store the feature value -> integer dictionary in the log

Returns

- **p** (*function pandas.DataFrame -> pandas.DataFrame*) – A function that when applied to a DataFrame with the same columns as *df* returns a new DataFrame with a new column with predictions from the model.

- **new_df** (*pandas.DataFrame*) – A *df*-like DataFrame with the same columns as the input *df* plus a column with predictions from the model.
- **log** (*dict*) – A log-like Dict that stores information of the Rank Categorical model.

fklearn.training.transformation.**selector**

Filters a DataFrames by selecting only the desired columns.

Parameters

- **df** (*pandas.DataFrame*) – A Pandas' DataFrame that must contain *columns*
- **training_columns** (*list of str*) – A list of column names that will remain in the dataframe during training time (fit)
- **predict_columns** (*list of str*) – A list of column names that will remain in the dataframe during prediction time (transform) If None, it defaults to *training_columns*.

Returns

- **p** (*function pandas.DataFrame -> pandas.DataFrame*) – A function that when applied to a DataFrame with the same columns as *df* returns a new DataFrame with a new column with predictions from the model.
- **new_df** (*pandas.DataFrame*) – A *df*-like DataFrame with the same columns as the input *df* plus a column with predictions from the model.
- **log** (*dict*) – A log-like Dict that stores information of the Selector model.

fklearn.training.transformation.**standard_scaler**(*df: pandas.core.frame.DataFrame = '__no__default__', columns_to_scale: List[str] = '__no__default__'*) → Union[Callable, Tuple[Callable[pandas.core.frame.DataFrame, pandas.core.frame.DataFrame], pandas.core.frame.DataFrame, Dict[str, Dict[str, Any]]]]

Fits a standard scaler to the dataset.

The default behaviour is to replace the original values. To store the original values in a new column, specify *prefix* or *suffix* in the parameters, or specify a dictionary with the desired column mapping using the *columns_mapping* parameter.

Parameters

- **df** (*pandas.DataFrame*) – A Pandas' DataFrame with columns to scale. It must contain all columns listed in *columns_to_scale*.
- **columns_to_scale** (*list of str*) – A list of names of the columns for standard scaling.

Returns

- **p** (*function pandas.DataFrame -> pandas.DataFrame*) – A function that when applied to a DataFrame with the same columns as *df* returns a new DataFrame with a new column with predictions from the model.
- **new_df** (*pandas.DataFrame*) – A *df*-like DataFrame with the same columns as the input *df* plus a column with predictions from the model.
- **log** (*dict*) – A log-like Dict that stores information of the Standard Scaler model.

fklearn.training.transformation.**target_categorizer**(*df: pandas.core.frame.DataFrame = '__no__default__', columns_to_categorize: List[str] = '__no__default__', target_column: str = '__no__default__', smoothing: float = 1.0, ignore_unseen: bool = True, store_mapping: bool = False*) → Union[Callable, Tuple[Callable[[pandas.core.frame.DataFrame, pandas.core.frame.DataFrame], pandas.core.frame.DataFrame, Dict[str, Dict[str, Any]]]]

Replaces categorical variables with the smoothed mean of the target variable by category. Uses a weighted average with the overall mean of the target variable for smoothing.

The default behaviour is to replace the original values. To store the original values in a new column, specify *prefix* or *suffix* in the parameters, or specify a dictionary with the desired column mapping using the *columns_mapping* parameter.

Parameters

- **df** (*pandas.DataFrame*) – A Pandas' DataFrame that must contain *columns_to_categorize* and *target_column* columns.
- **columns_to_categorize** (*list of str*) – A list of categorical column names.
- **target_column** (*str*) – Target column name. Target can be binary or continuous.
- **smoothing** (*float (default: 1.0)*) – Weight given to overall target mean against target mean by category. The value must be greater than or equal to 0
- **ignore_unseen** (*bool (default: True)*) – If True, unseen values will be encoded as nan If False, these will be replaced by target mean.
- **store_mapping** (*bool (default: False)*) – Whether to store the feature value -> float dictionary in the log.

Returns

- **p** (*function pandas.DataFrame -> pandas.DataFrame*) – A function that when applied to a DataFrame with the same columns as *df* returns a new DataFrame with a new column with predictions from the model.
- **new_df** (*pandas.DataFrame*) – A *df*-like DataFrame with the same columns as the input *df* plus a column with predictions from the model.
- **log** (*dict*) – A log-like Dict that stores information of the Target Categorizer model.

fklearn.training.transformation.**truncate_categorical**(*df: pandas.core.frame.DataFrame = '__no__default__', columns_to_truncate: List[str] = '__no__default__', percentile: float = '__no__default__', replacement: Union[str, float] = -9999, replace_unseen: Union[str, float] = -9999, store_mapping: bool = False*) → Union[Callable, Tuple[Callable[[pandas.core.frame.DataFrame], pandas.core.frame.DataFrame], Dict[str, Dict[str, Any]]]]

Truncate infrequent categories and replace them by a single one. You can think of it like "others" category.

The default behaviour is to replace the original values. To store the original values in a new column, specify *prefix* or *suffix* in the parameters, or specify a dictionary with the desired column mapping using the *columns_mapping* parameter.

Parameters

- **df** (*pandas.DataFrame*) – A Pandas' DataFrame that must contain a *prediction_column* columns.
- **columns_to_truncate** (*list of str*) – The df columns names to perform the truncation.
- **percentile** (*float*) – Categories less frequent than the percentile will be replaced by the same one.
- **replacement** (*int, str, float or nan*) – The value to use when a category is less frequent that the percentile variable.
- **replace_unseen** (*int, str, float, or nan*) – The value to impute unseen categories.
- **store_mapping** (*bool (default: False)*) – Whether to store the feature value -> integer dictionary in the log.

Returns

- **p** (*function pandas.DataFrame -> pandas.DataFrame*) – A function that when applied to a DataFrame with the same columns as *df* returns a new DataFrame with a new column with predictions from the model.
- **new_df** (*pandas.DataFrame*) – A *df*-like DataFrame with the same columns as the input *df* plus a column with predictions from the model.
- **log** (*dict*) – A log-like Dict that stores information of the Truncate Categorical model.

fklearn.training.transformation.**value_mapper**(*df: pandas.core.frame.DataFrame = '__no__default__', value_maps: Dict[str, Dict] = '__no__default__', ignore_unseen: bool = True, replace_unseen_to: Any = nan*) → Union[Callable, Tuple[Callable[pandas.core.frame.DataFrame, pandas.core.frame.DataFrame], pandas.core.frame.DataFrame, Dict[str, Dict[str, Any]]]]

Map values in selected columns in the DataFrame according to dictionaries of replacements. Learner wrapper for apply_replacements

Parameters

- **df** (*pandas.DataFrame*) – A Pandas DataFrame containing the data to be replaced.
- **value_maps** (*dict of dicts*) – A dict mapping a col to dict mapping a value to its replacement. For example: value_maps = {"feature1": {1: 2, 3: 5, 6: 8}}
- **ignore_unseen** (*bool*) – If True, values not explicitly declared in value_maps will be left as is. If False, these will be replaced by replace_unseen_to.
- **replace_unseen_to** (*Any*) – Default value to replace when original value is not present in the *vec* dict for the feature.

fklearn.training.unsupervised.**isolation_forest_learner**

Fits an anomaly detection algorithm (Isolation Forest) to the dataset

Parameters

- **df** (*pandas.DataFrame*) – A Pandas' DataFrame with features and target columns. The model will be trained to predict the target column from the features.
- **features** (*list of str*) – A list os column names that are used as features for the model. All this names should be in *df*.
- **params** (*dict*) – The IsolationForest parameters in the format {"par_name": param}. See: http://scikit-learn.org/stable/modules/generated/sklearn.ensemble.IsolationForest.html
- **prediction_column** (*str*) – The name of the column with the predictions from the model.
- **encode_extra_cols** (*bool (default: True)*) – If True, treats all columns in *df* with name pattern fklearn_feat__col==val' as feature columns.

Returns

- **p** (*function pandas.DataFrame -> pandas.DataFrame*) – A function that when applied to a DataFrame with the same columns as *df* returns a new DataFrame with a new column with predictions from the model.
- **new_df** (*pandas.DataFrame*) – A *df*-like DataFrame with the same columns as the input *df* plus a column with predictions from the model.
- **log** (*dict*) – A log-like Dict that stores information of the Isolation Forest model.

fklearn.training.utils.**expand_features_encoded**(*df: pandas.core.frame.DataFrame, features: List[str]*) → List[str]

Expand the list of features to include features created automatically by fklearn in encoders such as Onehotencoder. All features created by fklearn have the naming pattern *fklearn_feat__col==val*. This function looks for these names in the DataFrame columns, checks if they can be derivative of any of the features listed in *features*, adds them to the new list of features and removes the original names from the list.

E.g. df has columns *col1* with values 0 and 1 and *col2*. After Onehot-encoding *col1* df will have columns *fklearn_feat_col1==0*, *fklearn_feat_col1==1*, *col2*. This function will then add *fklearn_feat_col1==0* and *fklearn_feat_col1==1* to the list of features and remove *col1*. If for some reason df also has another column *fklearn_feat_col3==x* but *col3* is not on the list of features, this column will not be added.

Parameters

- **df** (*pd.DataFrame*) – A Pandas' DataFrame with all features.
- **features** (*list of str*) – The original list of features.

fklearn.tuning.model_agnostic_fc.**correlation_feature_selection**

Feature selection based on correlation

Parameters

- **train_set** (*pd.DataFrame*) – A Pandas' DataFrame with the training data
- **features** (*list of str*) – The list of features to consider when dropping with correlation
- **threshold** (*float*) – The correlation threshold. Will drop features with correlation equal or above this threshold

Returns

Return type log with feature correlation, features to drop and final features

fklearn.tuning.model_agnostic_fc.**variance_feature_selection**

Feature selection based on variance

Parameters

- **train_set** (*pd.DataFrame*) – A Pandas' DataFrame with the training data
- **features** (*list of str*) – The list of features to consider when dropping with variance
- **threshold** (*float*) – The variance threshold. Will drop features with variance equal or bellow this threshold

Returns

Return type log with feature variance, features to drop and final features

fklearn.tuning.samplers.**remove_by_feature_importance**

Performs feature selection based on feature importance

Parameters

- **log** (*dict*) – Dictionaries evaluations.
- **num_removed_by_step** (*int (default 5)*) – The number of features to remove

Returns **features** – The remaining features after removing based on feature importance

Return type list of str

fklearn.tuning.samplers.**remove_by_feature_shuffling**

Performs feature selection based on the evaluation of the test vs the evaluation of the test with randomly shuffled features

Parameters

- **log** (*LogType*) – Dictionaries evaluations.

- **predict_fn** (*function pandas.DataFrame -> pandas.DataFrame*) – A partially defined predictor that takes a DataFrame and returns the predicted score for this dataframe
- **eval_fn** (*function DataFrame -> log dict*) – A partially defined evaluation function that takes a dataset with prediction and returns the evaluation logs.
- **eval_data** (*pandas.DataFrame*) – Data used to evaluate the model after shuffling
- **extractor** (*function str -> float*) – A extractor that take a string and returns the value of that string on a dict
- **metric_name** (*str*) – String with the name of the column that refers to the metric column to be extracted
- **max_removed_by_step** (*int (default 5)*) – The maximum number of features to remove. It will only consider the least max_removed_by_step in terms of feature importance. If speed_up_by_importance=True it will first filter the least relevant feature an shuffle only those. If speed_up_by_importance=False it will shuffle all features and drop the last max_removed_by_step in terms of PIMP. In both cases, the features will only be removed if drop in performance is up to the defined threshold.
- **threshold** (*float (default 0.005)*) – Threshold for model performance comparison
- **speed_up_by_importance** (*bool (default True)*) – If it should narrow search looking at feature importance first before getting PIMP importance. If True, will only shuffle the top num_removed_by_step in terms of feature importance.
- **parallel** (*bool (default False)*) –
- **nthread** (*int (default 1)*) –
- **seed** (*int (default 7)*) – Random seed

Returns **features** – The remaining features after removing based on feature importance

Return type list of str

fklearn.tuning.samplers.**remove_features_subsets**

Performs feature selection based on the best performing model out of several trained models

Parameters

- **log_list** (*list of dict*) – A list of log-like lists of dictionaries evaluations.
- **extractor** (*function string -> float*) – A extractor that take a string and returns the value of that string on a dict
- **metric_name** (*str*) – String with the name of the column that refers to the metric column to be extracted
- **num_removed_by_step** (*int (default 1)*) – The number of features to remove

Returns **keys** – The remaining keys of feature sets after choosing the current best subset

Return type list of str

fklearn.tuning.stoppers.**aggregate_stop_funcs**(**stop_funcs*) → Callable[List[List[Dict[str, Any]]], bool]

Aggregate stop functions

Parameters **stop_funcs** (*list of function list of dict -> bool*) –

Returns **l** – Function that performs the Or logic of all stop_fn applied to the logs

Return type function logs -> bool

fklearn.tuning.stoppers.**stop_by_iter_num**
Checks for logs to see if feature selection should stop

Parameters

- **logs** (*list of list of dict*) – A list of log-like lists of dictionaries evaluations.
- **iter_limit** (*int (default 50)*) – Limit of Iterations

Returns **stop** – A boolean whether to stop recursion or not

Return type bool

fklearn.tuning.stoppers.**stop_by_no_improvement**
Checks for logs to see if feature selection should stop

Parameters

- **logs** (*list of list of dict*) – A list of log-like lists of dictionaries evaluations.
- **extractor** (*function str -> float*) – A extractor that take a string and returns the value of that string on a dict
- **metric_name** (*str*) – String with the name of the column that refers to the metric column to be extracted
- **early_stop** (*int (default 3)*) – Number of iteration without improval before stopping
- **threshold** (*float (default 0.001)*) – Threshold for model performance comparison

Returns **stop** – A boolean whether to stop recursion or not

Return type bool

fklearn.tuning.stoppers.**stop_by_no_improvement_parallel**
Checks for logs to see if feature selection should stop

Parameters

- **logs** (*list of list of dict*) – A list of log-like lists of dictionaries evaluations.
- **extractor** (*function str -> float*) – A extractor that take a string and returns the value of that string on a dict
- **metric_name** (*str*) – String with the name of the column that refers to the metric column to be extracted
- **early_stop** (*int (default 3)*) – Number of iterations without improvements before stopping
- **threshold** (*float (default 0.001)*) – Threshold for model performance comparison

Returns **stop** – A boolean whether to stop recursion or not

Return type bool

fklearn.tuning.stoppers.**stop_by_num_features**
Checks for logs to see if feature selection should stop

Parameters

- **logs** (*list of list of dict*) – A list of log-like lists of dictionaries evaluations.

- **min_num_features** (*int (default 50)*) – The minimun number of features the model can have before stopping

Returns **stop** – A boolean whether to stop recursion or not

Return type bool

fklearn.tuning.stoppers.**stop_by_num_features_parallel**

Selects the best log out of a list to see if feature selection should stop

Parameters

- **logs** (*list of list of list of dict*) – A list of log-like lists of dictionaries evaluations.
- **extractor** (*function str -> float*) – A extractor that take a string and returns the value of that string on a dict
- **metric_name** (*str*) – String with the name of the column that refers to the metric column to be extracted
- **min_num_features** (*int (default 50)*) – The minimun number of features the model can have before stopping

Returns **stop** – A boolean whether to stop recursion or not

Return type bool

fklearn.validation.evaluators.**auc_evaluator**

Computes the ROC AUC score, given true label and prediction scores.

Parameters

- **test_data** (*Pandas' DataFrame*) – A Pandas' DataFrame with target and prediction scores.
- **prediction_column** (*Strings*) – The name of the column in *test_data* with the prediction scores.
- **target_column** (*String*) – The name of the column in *test_data* with the binary target.
- **eval_name** (*String, optional (default=None)*) – the name of the evaluator as it will appear in the logs.

Returns **log** – A log-like dictionary with the ROC AUC Score

Return type dict

fklearn.validation.evaluators.**brier_score_evaluator**

Computes the Brier score, given true label and prediction scores.

Parameters

- **test_data** (*Pandas' DataFrame*) – A Pandas' DataFrame with with target and prediction scores.
- **prediction_column** (*Strings*) – The name of the column in *test_data* with the prediction scores.
- **target_column** (*String*) – The name of the column in *test_data* with the binary target.
- **eval_name** (*String, optional (default=None)*) – The name of the evaluator as it will appear in the logs.

Returns **log** – A log-like dictionary with the Brier score.

Return type dict

fklearn.validation.evaluators.**combined_evaluators**
Combine partially applies evaluation functions.

Parameters

- **test_data** (*Pandas' DataFrame*) – A Pandas' DataFrame to apply the evaluators on
- **evaluators** (*List*) – List of evaluator functions

Returns **log** – A log-like dictionary with the column mean

Return type dict

fklearn.validation.evaluators.**correlation_evaluator**
Computes the Pearson correlation between prediction and target.

Parameters

- **test_data** (*Pandas' DataFrame*) – A Pandas' DataFrame with with target and prediction.
- **prediction_column** (*Strings*) – The name of the column in *test_data* with the prediction.
- **target_column** (*String*) – The name of the column in *test_data* with the continuous target.
- **eval_name** (*String, optional (default=None)*) – the name of the evaluator as it will appear in the logs.

Returns **log** – A log-like dictionary with the Pearson correlation

Return type dict

fklearn.validation.evaluators.**expected_calibration_error_evaluator**
Computes the expected calibration error (ECE), given true label and prediction scores. See "On Calibration of Modern Neural Networks"(https://arxiv.org/abs/1706.04599) for more information.

The ECE is the distance between the actuals observed frequency and the predicted probabilities, for a given choice of bins.

Perfect calibration results in a score of 0.

For example, if for the bin [0, 0.1] we have the three data points:

1. prediction: 0.1, actual: 0
2. prediction: 0.05, actual: 1
3. prediction: 0.0, actual 0

Then the predicted average is (0.1 + 0.05 + 0.00)/3 = 0.05, and the empirical frequency is (0 + 1 + 0)/3 = 1/3. Therefore, the distance for this bin is:

```
|1/3 - 0.05| ~= 0.28.
```

Graphical intuition:

```
Actuals (empirical frequency between 0 and 1)
|      *
|    *
| *
 _______ Predictions (probabilties between 0 and 1)
```

Parameters

- **test_data** (*Pandas' DataFrame*) – A Pandas' DataFrame with with target and prediction scores.
- **prediction_column** (*Strings*) – The name of the column in *test_data* with the prediction scores.
- **target_column** (*String*) – The name of the column in *test_data* with the binary target.
- **eval_name** (*String, optional (default=None)*) – The name of the evaluator as it will appear in the logs.
- **n_bins** (*Int (default=100)*) – The number of bins. This is a trade-off between the number of points in each bin and the probability range they span. You want a small enough range that still contains a significant number of points for the distance to work.
- **bin_choice** (*String (default="count")*) – Two possibilities: "count" for equally populated bins (e.g. uses *pandas.qcut* for the bins) "prob" for equally spaced probabilities (e.g. uses *pandas.cut* for the bins), with distance weighed by the number of samples in each bin.

Returns **log** – A log-like dictionary with the expected calibration error.

Return type dict

fklearn.validation.evaluators.**fbeta_score_evaluator**

Computes the F-beta score, given true label and prediction scores.

Parameters

- **test_data** (*pandas.DataFrame*) – A Pandas' DataFrame with with target and prediction scores.
- **threshold** (*float*) –

  A threshold for the prediction column above which samples will be classified as 1
- **beta** (*float*) – The beta parameter determines the weight of precision in the combined score. beta < 1 lends more weight to precision, while beta > 1 favors recall (beta -> 0 considers only precision, beta -> inf only recall).
- **prediction_column** (*str*) – The name of the column in *test_data* with the prediction scores.
- **target_column** (*str*) – The name of the column in *test_data* with the binary target.
- **eval_name** (*str, optional (default=None)*) – the name of the evaluator as it will appear in the logs.

Returns **log** – A log-like dictionary with the Precision Score

Return type dict

fklearn.validation.evaluators.**generic_sklearn_evaluator**(*name_prefix: str, sklearn_metric: Callable[..., float]*) → Callable[..., Dict[str, Union[float, Dict]]]

Returns an evaluator build from a metric from sklearn.metrics

Parameters

- **name_prefix** (*str*) – The default name of the evaluator will be name_prefix + target_column.

- **sklearn_metric** (*Callable*) – Metric function from sklearn.metrics. It should take as parameters y_true, y_score, kwargs.

Returns **eval_fn** – An evaluator function that uses the provided metric

Return type Callable

fklearn.validation.evaluators.**hash_evaluator**

Computes the hash of a pandas dataframe, filtered by hash columns. The purpose is to uniquely identify a dataframe, to be able to check if two dataframes are equal or not.

Parameters

- **test_data** (*Pandas' DataFrame*) – A Pandas' DataFrame to be hashed.
- **hash_columns** (*List[str], optional (default=None)*) – A list of column names to filter the dataframe before hashing. If None, it will hash the dataframe with all the columns
- **eval_name** (*String, optional (default=None)*) – the name of the evaluator as it will appear in the logs.
- **consider_index** (*bool, optional (default=False)*) – If true, will consider the index of the dataframe to calculate the hash. The default behaviour will ignore the index and just hash the content of the features.

Returns **log** – A log-like dictionary with the hash of the dataframe

Return type dict

fklearn.validation.evaluators.**logloss_evaluator**

Computes the logloss score, given true label and prediction scores.

Parameters

- **test_data** (*Pandas' DataFrame*) – A Pandas' DataFrame with with target and prediction scores.
- **prediction_column** (*Strings*) – The name of the column in *test_data* with the prediction scores.
- **target_column** (*String*) – The name of the column in *test_data* with the binary target.
- **eval_name** (*String, optional (default=None)*) – the name of the evaluator as it will appear in the logs.

Returns **log** – A log-like dictionary with the logloss score.

Return type dict

fklearn.validation.evaluators.**mean_prediction_evaluator**

Computes mean for the specified column.

Parameters

- **test_data** (*Pandas' DataFrame*) – A Pandas' DataFrame with a column to compute the mean
- **prediction_column** (*Strings*) – The name of the column in *test_data* to compute the mean.
- **eval_name** (*String, optional (default=None)*) – the name of the evaluator as it will appear in the logs.

Returns **log** – A log-like dictionary with the column mean

Return type dict

fklearn.validation.evaluators.**mse_evaluator**

Computes the Mean Squared Error, given true label and predictions.

Parameters

- **test_data** (*Pandas' DataFrame*) – A Pandas' DataFrame with with target and predictions.
- **prediction_column** (*Strings*) – The name of the column in *test_data* with the predictions.
- **target_column** (*String*) – The name of the column in *test_data* with the continuous target.
- **eval_name** (*String, optional (default=None)*) – the name of the evaluator as it will appear in the logs.

Returns **log** – A log-like dictionary with the MSE Score

Return type dict

fklearn.validation.evaluators.**ndcg_evaluator**

Computes the Normalized Discount Cumulative Gain (NDCG) between of the original and predicted rankings: https://en.wikipedia.org/wiki/Discounted_cumulative_gain

Parameters

- **test_data** (*Pandas DataFrame*) – A Pandas' DataFrame with with target and prediction scores.
- **prediction_column** (*String*) – The name of the column in *test_data* with the prediction scores.
- **target_column** (*String*) – The name of the column in *test_data* with the target.
- **k** (*int, optional (default=None)*) – The size of the rank that is used to fit (highest k scores) the NDCG score. If None, use all outputs. Otherwise, this value must be between [1, len(test_data[prediction_column])].
- **exponential_gain** (*bool (default=True)*) – If False, then use the linear gain. The exponential gain places a stronger emphasis on retrieving relevant items. If the relevance of these items is binary values in {0,1}, then the two approaches are the same, which is the linear case.
- **eval_name** (*String, optional (default=None)*) – The name of the evaluator as it will appear in the logs.

Returns **log** – A log-like dictionary with the NDCG score, float in [0,1].

Return type dict

fklearn.validation.evaluators.**permutation_evaluator**

Permutation importance evaluator. It works by shuffling one or more features on test_data dataframe, getting the preditions with predict_fn, and evaluating the results with eval_fn.

Parameters

- **test_data** (*Pandas' DataFrame*) – A Pandas' DataFrame with with target, predictions and features.
- **predict_fn** (*function DataFrame -> DataFrame*) – Function that receives the input dataframe and returns a dataframe with the pipeline predictions.

- **eval_fn** (*function DataFrame -> Log Dict*) – A partially applied evaluation function.
- **baseline** (*bool*) – Also evaluates the predict_fn on an unshuffled baseline.
- **features** (*List of strings*) – The features to shuffle and then evaluate eval_fn on the shuffled results. The default case shuffles all dataframe columns.
- **shuffle_all_at_once** (*bool*) – Shuffle all features at once instead of one per turn.
- **random_state** (*int*) – Seed to be used by the random number generator.
- **eval_name** (*String, optional (default=None)*) – the name of the evaluator as it will appear in the logs.

Returns **log** – A log-like dictionary with evaluation results by feature shuffle. Use the permutation_extractor for better visualization of the results.

Return type dict

fklearn.validation.evaluators.**pr_auc_evaluator**

Computes the PR AUC score, given true label and prediction scores.

Parameters

- **test_data** (*Pandas' DataFrame*) – A Pandas' DataFrame with target and prediction scores.
- **prediction_column** (*Strings*) – The name of the column in *test_data* with the prediction scores.
- **target_column** (*String*) – The name of the column in *test_data* with the binary target.
- **eval_name** (*String, optional (default=None)*) – the name of the evaluator as it will appear in the logs.

Returns

Return type A log-like dictionary with the PR AUC Score

fklearn.validation.evaluators.**precision_evaluator**

Computes the precision score, given true label and prediction scores.

Parameters

- **test_data** (*pandas.DataFrame*) – A Pandas' DataFrame with with target and prediction scores.
- **threshold** (*float*) –

  A threshold for the prediction column above which samples will be classified as 1
- **prediction_column** (*str*) – The name of the column in *test_data* with the prediction scores.
- **target_column** (*str*) – The name of the column in *test_data* with the binary target.
- **eval_name** (*str, optional (default=None)*) – the name of the evaluator as it will appear in the logs.

Returns **log** – A log-like dictionary with the Precision Score

Return type dict

fklearn.validation.evaluators.**r2_evaluator**

Computes the R2 score, given true label and predictions.

Parameters

- **test_data** (*Pandas' DataFrame*) – A Pandas' DataFrame with with target and prediction.
- **prediction_column** (*Strings*) – The name of the column in *test_data* with the prediction.
- **target_column** (*String*) – The name of the column in *test_data* with the continuous target.
- **eval_name** (*String, optional (default=None)*) – the name of the evaluator as it will appear in the logs.

Returns **log** – A log-like dictionary with the R2 Score

Return type dict

fklearn.validation.evaluators.**recall_evaluator**

Computes the recall score, given true label and prediction scores.

Parameters

- **test_data** (*pandas.DataFrame*) – A Pandas' DataFrame with with target and prediction scores.
- **threshold** (*float*) –

  A threshold for the prediction column above which samples will be classified as 1
- **prediction_column** (*str*) – The name of the column in *test_data* with the prediction scores.
- **target_column** (*str*) – The name of the column in *test_data* with the binary target.
- **eval_name** (*str, optional (default=None)*) – the name of the evaluator as it will appear in the logs.

Returns **log** – A log-like dictionary with the Precision Score

Return type dict

fklearn.validation.evaluators.**roc_auc_evaluator**

Computes the ROC AUC score, given true label and prediction scores.

Parameters

- **test_data** (*Pandas' DataFrame*) – A Pandas' DataFrame with target and prediction scores.
- **prediction_column** (*Strings*) – The name of the column in *test_data* with the prediction scores.
- **target_column** (*String*) – The name of the column in *test_data* with the binary target.
- **eval_name** (*String, optional (default=None)*) – the name of the evaluator as it will appear in the logs.

Returns **log** – A log-like dictionary with the ROC AUC Score

Return type dict

fklearn.validation.evaluators.**spearman_evaluator**

Computes the Spearman correlation between prediction and target. The Spearman correlation evaluates the rank order between two variables: https://en.wikipedia.org/wiki/Spearman%27s_rank_correlation_coefficient

Parameters

- **test_data** (*Pandas' DataFrame*) – A Pandas' DataFrame with with target and prediction.
- **prediction_column** (*Strings*) – The name of the column in *test_data* with the prediction.
- **target_column** (*String*) – The name of the column in *test_data* with the continuous target.
- **eval_name** (*String, optional (default=None)*) – the name of the evaluator as it will appear in the logs.

Returns **log** – A log-like dictionary with the Spearman correlation

Return type dict

fklearn.validation.evaluators.**split_evaluator**

Splits the dataset into the categories in *split_col* and evaluate model performance in each split. Useful when you belive the model performs differs in a sub population defined by *split_col*.

Parameters

- **test_data** (*Pandas' DataFrame*) – A Pandas' DataFrame with with target and predictions.
- **eval_fn** (*function DataFrame -> Log Dict*) – A partially applied evaluation function.
- **split_col** (*String*) – The name of the column in *test_data* to split by.
- **split_values** (*Array, optional (default=None)*) – An Array to split by. If not provided, *test_data[split_col].unique()* will be used.
- **eval_name** (*String, optional (default=None)*) – the name of the evaluator as it will appear in the logs.

Returns **log** – A log-like dictionary with evaluation results by split.

Return type dict

fklearn.validation.evaluators.**temporal_split_evaluator**

Splits the dataset into the temporal categories by *time_col* and evaluate model performance in each split.

The splits are implicitly defined by the *time_format*. For example, for the default time format ("%Y-%m"), we will split by year and month.

Parameters

- **test_data** (*Pandas' DataFrame*) – A Pandas' DataFrame with with target and predictions.
- **eval_fn** (*function DataFrame -> Log Dict*) – A partially applied evaluation function.
- **time_col** (*string*) – The name of the column in *test_data* to split by.
- **time_format** (*string*) – The way to format the *time_col* into temporal categories.
- **split_values** (*Array of string, optional (default=None)*) – An array of date formatted strings to split the evaluation by. If not provided, all unique formatted dates will be used.
- **eval_name** (*String, optional (default=None)*) – the name of the evaluator as it will appear in the logs.

Returns **log** – A log-like dictionary with evaluation results by split.

Return type dict

fklearn.validation.splitters.**forward_stability_curve_time_splitter**

Splits the data into temporal buckets with both the training and testing folds both moving forward. The folds move forward by a fixed timedelta step. Optionally, there can be a gap between the end of the training period and the start of the holdout period.

Similar to the stability curve time splitter, with the difference that the training period also moves forward with each fold.

The clearest use case is to evaluate a periodic re-training framework.

Parameters

- **train_data** (*pandas.DataFrame*) – A Pandas' DataFrame that will be split for stability curve estimation.
- **training_time_start** (*datetime.datetime or str*) – Date for the start of the training period. If *move_training_start_with_steps* is *True*, each step will increase this date by *step*.
- **training_time_end** (*datetime.datetime or str*) – Date for the end of the training period. Each step increases this date by *step*.
- **time_column** (*str*) – The name of the Date column of *train_data*.
- **holdout_gap** (*datetime.timedelta*) – Timedelta of the gap between the end of the training period and the start of the validation period.
- **holdout_size** (*datetime.timedelta*) – Timedelta of the range between the start and the end of the holdout period.
- **step** (*datetime.timedelta*) – Timedelta that shifts both the training period and the holdout period by this value.
- **move_training_start_with_steps** (*bool*) – If True, the training start date will increase by *step* for each fold. If False, the training start date remains fixed at the *training_time_start* value.

Returns

- **Folds** (*list of tuples*) – A list of folds. Each fold is a Tuple of arrays. The fist array in each tuple contains training indexes while the second array contains validation indexes.
- **logs** (*list of dict*) – A list of logs, one for each fold

fklearn.validation.splitters.**k_fold_splitter**

Makes K random train/test split folds for cross validation. The folds are made so that every sample is used at least once for evaluating and K-1 times for training.

If stratified is set to True, the split preserves the distribution of stratify_column

Parameters

- **train_data** (*pandas.DataFrame*) – A Pandas' DataFrame that will be split into K-Folds for cross validation.
- **n_splits** (*int*) – The number of folds K for the K-Fold cross validation strategy.
- **random_state** (*int*) – Seed to be used by the random number generator.
- **stratify_column** (*string*) – Column name in train_data to be used for stratified split.

Returns

- **Folds** (*list of tuples*) – A list of folds. Each fold is a Tuple of arrays. The fist array in each tuple contains training indexes while the second array contains validation indexes.
- **logs** (*list of dict*) – A list of logs, one for each fold

fklearn.validation.splitters.**out_of_time_and_space_splitter**

Makes K grouped train/test split folds for cross validation. The folds are made so that every ID is used at least once for evaluating and K-1 times for training. Also, for each fold, evaluation will always be out-of-ID and out-of-time.

Parameters

- **train_data** (*pandas.DataFrame*) – A Pandas' DataFrame that will be split into K out-of-time and ID folds for cross validation.
- **n_splits** (*int*) – The number of folds K for the K-Fold cross validation strategy.
- **in_time_limit** (*str or datetime.datetime*) – A String representing the end time of the training data. It should be in the same format as the Date column in *train_data*.
- **time_column** (*str*) – The name of the Date column of *train_data*.
- **space_column** (*str*) – The name of the ID column of *train_data*.
- **holdout_gap** (*datetime.timedelta*) – Timedelta of the gap between the end of the training period and the start of the validation period.

Returns

- **Folds** (*list of tuples*) – A list of folds. Each fold is a Tuple of arrays. The fist array in each tuple contains training indexes while the second array contains validation indexes.
- **logs** (*list of dict*) – A list of logs, one for each fold

fklearn.validation.splitters.**reverse_time_learning_curve_splitter**

Splits the data into temporal buckets given by the specified frequency. Uses a fixed out-of-ID and time hold out set for every fold. Training size increases per fold, with less recent data being added in each fold. Useful for inverse learning curve validation, that is, for seeing how hold out performance increases as the training size increases with less recent data.

Parameters

- **train_data** (*pandas.DataFrame*) – A Pandas' DataFrame that will be split inverse learning curve estimation.
- **time_column** (*str*) – The name of the Date column of *train_data*.
- **training_time_limit** (*str*) – The Date String for the end of the training period. Should be of the same format as *time_column*.
- **lower_time_limit** (*str*) – A Date String for the begining of the training period. This allows limiting the learning curve from bellow, avoiding heavy computation with very old data.
- **freq** (*str*) – The temporal frequency. See: http://pandas.pydata.org/pandas-docs/stable/timeseries.html#offset-aliases
- **holdout_gap** (*datetime.timedelta*) – Timedelta of the gap between the end of the training period and the start of the validation period.
- **min_samples** (*int*) – The minimum number of samples required in the split to keep the split.

Returns

- **Folds** (*list of tuples*) – A list of folds. Each fold is a Tuple of arrays. The fist array in each tuple contains training indexes while the second array contains validation indexes.
- **logs** (*list of dict*) – A list of logs, one for each fold

fklearn.validation.splitters.**spatial_learning_curve_splitter**

Splits the data for a spatial learning curve. Progressively adds more and more examples to the training in order to verify the impact of having more data available on a validation set.

The validation set starts after the training set, with an optional time gap.

Similar to the temporal learning curves, but with spatial increases in the training set.

Parameters

- **train_data** (*pandas.DataFrame*) – A Pandas' DataFrame that will be split for learning curve estimation.
- **space_column** (*str*) – The name of the ID column of *train_data*.
- **time_column** (*str*) – The name of the temporal column of *train_data*.
- **training_limit** (*datetime or str*) – The date limiting the training (after which the holdout begins).
- **holdout_gap** (*timedelta*) – The gap between the end of training and the start of the holdout. If you have censored data, use a gap similar to the censor time.
- **train_percentages** (*list or tuple of floats*) – A list containing the percentages of IDs to use in the training. Defaults to (0.25, 0.5, 0.75, 1.0). For example: For the default value, there would be four model trainings, containing respectively 25%, 50%, 75%, and 100% of the IDs that are not part of the held out set.
- **random_state** (*int*) – A seed for the random number generator that shuffles the IDs.

Returns

- **Folds** (*list of tuples*) – A list of folds. Each fold is a Tuple of arrays. The fist array in each tuple contains training indexes while the second array contains validation indexes.
- **logs** (*list of dict*) – A list of logs, one for each fold

fklearn.validation.splitters.**stability_curve_time_in_space_splitter**

Splits the data into temporal buckets given by the specified frequency. Training set is fixed before hold out and uses a rolling window hold out set. Each fold moves the hold out further into the future. Useful to see how model performance degrades as the training data gets more outdated. Folds are made so that ALL IDs in the holdout also appear in the training set.

Parameters

- **train_data** (*pandas.DataFrame*) – A Pandas' DataFrame that will be split for stability curve estimation.
- **training_time_limit** (*str*) – The Date String for the end of the testing period. Should be of the same format as *time_column*.
- **space_column** (*str*) – The name of the ID column of *train_data*.
- **time_column** (*str*) – The name of the Date column of *train_data*.
- **freq** (*str*) – The temporal frequency. See: http://pandas.pydata.org/pandas-docs/stable/timeseries.html#offset-aliases
- **space_hold_percentage** (*float (default=0.5)*) – The proportion of hold out IDs.

- **random_state** (*int*) – A seed for the random number generator for ID sampling across train and hold out sets.
- **min_samples** (*int*) – The minimum number of samples required in the split to keep the split.

Returns

- **Folds** (*list of tuples*) – A list of folds. Each fold is a Tuple of arrays. The fist array in each tuple contains training indexes while the second array contains validation indexes.
- **logs** (*list of dict*) – A list of logs, one for each fold

fklearn.validation.splitters.**stability_curve_time_space_splitter**

Splits the data into temporal buckets given by the specified frequency. Training set is fixed before hold out and uses a rolling window hold out set. Each fold moves the hold out further into the future. Useful to see how model performance degrades as the training data gets more outdated. Folds are made so that NONE of the IDs in the holdout appears in the training set.

Parameters

- **train_data** (*pandas.DataFrame*) – A Pandas' DataFrame that will be split for stability curve estimation.
- **training_time_limit** (*str*) – The Date String for the end of the testing period. Should be of the same format as *time_column*
- **space_column** (*str*) – The name of the ID column of *train_data*
- **time_column** (*str*) – The name of the Date column of *train_data*
- **freq** (*str*) – The temporal frequency. See: http://pandas.pydata.org/pandas-docs/stable/timeseries.html#offset-aliases
- **space_hold_percentage** (*float*) – The proportion of hold out IDs
- **random_state** (*int*) – A seed for the random number generator for ID sampling across train and hold out sets.
- **min_samples** (*int*) – The minimum number of samples required in the split to keep the split.

Returns

- **Folds** (*list of tuples*) – A list of folds. Each fold is a Tuple of arrays. The fist array in each tuple contains training indexes while the second array contains validation indexes.
- **logs** (*list of dict*) – A list of logs, one for each fold

fklearn.validation.splitters.**stability_curve_time_splitter**

Splits the data into temporal buckets given by the specified frequency. Training set is fixed before hold out and uses a rolling window hold out set. Each fold moves the hold out further into the future. Useful to see how model performance degrades as the training data gets more outdated. Training and holdout sets can have same IDs

Parameters

- **train_data** (*pandas.DataFrame*) – A Pandas' DataFrame that will be split for stability curve estimation.
- **training_time_limit** (*str*) – The Date String for the end of the testing period. Should be of the same format as *time_column*.
- **time_column** (*str*) – The name of the Date column of *train_data*.

- **freq** (*str*) – The temporal frequency. See: http://pandas.pydata.org/pandas-docs/stable/timeseries.html#offset-aliases
- **min_samples** (*int*) – The minimum number of samples required in a split to keep it.

Returns

- **Folds** (*list of tuples*) – A list of folds. Each fold is a Tuple of arrays. The fist array in each tuple contains training indexes while the second array contains validation indexes.
- **logs** (*list of dict*) – A list of logs, one for each fold

fklearn.validation.splitters.**time_and_space_learning_curve_splitter**

Splits the data into temporal buckets given by the specified frequency. Uses a fixed out-of-ID and time hold out set for every fold. Training size increases per fold, with more recent data being added in each fold. Useful for learning curve validation, that is, for seeing how hold out performance increases as the training size increases with more recent data.

Parameters

- **train_data** (*pandas.DataFrame*) – A Pandas' DataFrame that will be split for learning curve estimation.
- **training_time_limit** (*str*) – The Date String for the end of the testing period. Should be of the same format as *time_column*.
- **space_column** (*str*) – The name of the ID column of *train_data*.
- **time_column** (*str*) – The name of the Date column of *train_data*.
- **freq** (*str*) – The temporal frequency. See: http://pandas.pydata.org/pandas-docs/stable/timeseries.html#offset-aliases
- **space_hold_percentage** (*float*) – The proportion of hold out IDs.
- **holdout_gap** (*datetime.timedelta*) – Timedelta of the gap between the end of the training period and the start of the validation period.
- **random_state** (*int*) – A seed for the random number generator for ID sampling across train and hold out sets.
- **min_samples** (*int*) – The minimum number of samples required in the split to keep the split.

Returns

- **Folds** (*list of tuples*) – A list of folds. Each fold is a Tuple of arrays. The fist array in each tuple contains training indexes while the second array contains validation indexes.
- **logs** (*list of dict*) – A list of logs, one for each fold

fklearn.validation.splitters.**time_learning_curve_splitter**

Splits the data into temporal buckets given by the specified frequency.

Uses a fixed out-of-ID and time hold out set for every fold. Training size increases per fold, with more recent data being added in each fold. Useful for learning curve validation, that is, for seeing how hold out performance increases as the training size increases with more recent data.

Parameters

- **train_data** (*pandas.DataFrame*) – A Pandas' DataFrame that will be split for learning curve estimation.
- **training_time_limit** (*str*) – The Date String for the end of the testing period. Should be of the same format as *time_column*.

- **time_column** (*str*) – The name of the Date column of *train_data*.
- **freq** (*str*) – The temporal frequency. See: http://pandas.pydata.org/pandas-docs/stable/timeseries.html#offset-aliases
- **holdout_gap** (*datetime.timedelta*) – Timedelta of the gap between the end of the training period and the start of the validation period.
- **min_samples** (*int*) – The minimum number of samples required in the split to keep the split.

Returns

- **Folds** (*list of tuples*) – A list of folds. Each fold is a Tuple of arrays. The fist array in each tuple contains training indexes while the second array contains validation indexes.
- **logs** (*list of dict*) – A list of logs, one for each fold

1.4 Contributing

Table of contents:

1.4.1 Where to start?

We love pull requests(and issues) from everyone. We recommend you to take a look at the project, follow the examples before contribute with code.

By participating in this project, you agree to abide by our code of conduct.

1.4.2 Getting Help

If you found a bug or need a new feature, you can submit an issue.

If you would like to chat with other contributors to fklearn, consider joining the Gitter.

1.4.3 Working with the code

Now that that you already understand how the project works, maybe it's time to fix something, add and enhancement, or write new documentation. It's time to understand how we send contributions.

Version control

This project is hosted in Github, this way, to contribute with it you need an account, you sign up *here <https://github.com/signup/free>*_ We use git as version control, so it's good to understand basic git flows before send new code. You can follow Github Help to understand how to work with git.

Fork

To write new code, you will iteract with your own fork, so go to fklearn repo page, and hit the `Fork` button. This will create a copy of our repository in your account. To clone the repository in your machine:

```
git clone git@github.com:your-username/fklearn.git
git remote add upstream https://github.com/nubank/fklearn.git
```

This will create a folder called `fklearn` and will connect to the upstream(main repo).

Development env

We recommend you to create a virtualenv before starts to work with the code. And be able to run all tests locally before start to write new code.

Creating the virtualenv

```
# Use an ENV_DIR of you choice. We are using ~/venvs
python3.6 -m venv ~/venvs/fklearn-dev
source ~/venvs/fklearn-dev/activate
```

Install the requirements

This command will install all the test dependencies. To install the package itself follow install instruction.

```
pip install -qe .[test_deps]
```

Run tests

The following command should run all tests, if every test pass, you should be ready to start develop new stuff

```
python -m pytest tests/
```

Creating a development branch

First we should check if you master is up to date with the latest version of the repo

```
git checkout master
git pull upstream master --ff-only
```

```
git checkout -b name-of-your-bugfix-or-feature
```

If you already have a branch, and you want to update with the upstream master

```
git checkout name-of-your-bugfix-or-feature
git fetch upstream
git merge upstream/master
```

1.4.4 Contribute with code

In this session we'll guide you on how to contribute with the code. This is a guide if you want to fix or implement a new feature.

Code standards

This project is compatible only with python3.6 and follows the pep8 style And we use this import formatting

In order to check if your code follow our style, you can run from the repo root dir:

```
python -m pip install -q flake8
python -m flake8 \
--ignore=E731,W503 \
--filename=\*.py \
--exclude=__init__.py \
--show-source \
--statistics \
```

(continues on next page)

(continued from previous page)

```
--max-line-length=120 \
src/ tests/
```

Run tests

After you finish your feature development or bug fix, you should run your tests, using:

```
python -m pytest tests/
```

Or if want to run only one test:

```
python -m pytest tests/test-file-name.py::test_method_name
```

You should **always** write tests for your features, you can look at the other tests to have a better idea how we implement them. As test framework we use pytest

Document your code

All methods should have type annotations, this allow us to know what that method expect as parameters, and what is the output. You can learn more about it in typing docs

To document your code you should add docstrings, all methods with docstring will appear in this documentation's api file. If you created a new file, you may need to add it to the `api.rst` following the structure

```
 Folder Name
 -----------

 File name (fklearn.folder_name.file_name)
 #########################################

 ..currentmodule:: fklearn.folder_name.file_name

 .. autosummary::
   method_name

The docstrings should follow this format

.. code-block:: none

 """
 Brief introduction of method

 More info about it

 Parameters
 ----------

 parameter_1 : type
     Parameter description

 Returns
 -------

 value_1 : type
```

(continues on next page)

(continued from previous page)

```
    Value description
"""
```

1.4.5 Contribute with documentation

You can add, fix documenation of: code(docstrings) or this documentation files.

Docstrings

Follow the same structure we explained in code contribution

Documentation

This documentation is written using rst(`reStructuredText`) you can learn more about it in rst docs When you make changes in the docs, please make sure, we still be able to build it without any issue.

Build documentation

From `docs/` folder, install *requirments.txt* and run

```
make html
```

This command will build the documentation inside `docs/build/html` and you can check locally how it looks, and if everything worked.

1.4.6 Send you changes to Fklearn repo

Commit your changes

You should think about a commit as a unit of change. So it should describe a small change you did in the project.

The following command will list all files you changed:

```
git status
```

To choose which files will be added to the commit:

```
git add path/to/the/file/name.extension
```

And to write a commit message:

This command will open your text editor to write commit messages

```
git commit
```

This will add a commit only with subject

```
git commit -m "My commit message"
```

We recommend this guide to write better commit messages

Push the changes

After you write all your commit messages, decribing what you did, it's time to send to your remote repo.

```
git push origin name-of-your-bugfix-or-feature
```

Create a pull request

Now that you already finished your job, you should: - Go to your repo's Github page - Click `New pull request` - Choose the branch you want to merge - Review the files that will be merged - Click `Create pull request` - Fill the template - Tag your PR, add the category(bug, enhancement, documentation...) and a review-request label

When my code will be merged?

All code will be reviewed, we require at least one code owner review, and any other person review. We will usually do weekly releases of the package if we have any new features, that are already reviewed.

1.4.7 Versioning

Use Semantic versioning to set library versions, more info: semver.org But basically this means:

1. MAJOR version when you make incompatible API changes,
2. MINOR version when you add functionality in a backwards-compatible manner, and
3. PATCH version when you make backwards-compatible bug fixes.

(from semver.org summary)

You don't need to set the version in your PR, we'll take care of this when we decide to release a new version. Today the process is:

- Create a new `milestone` X.Y.Z (maintainers only)
- Some PR/issues are attibuted to this new milestone
- Merge all the related PRs (maintainers only)
- Create a new PR: `Bump package to X.Y.Z` This PR update the version and the change log (maintainers only)
- Create a tag `X.Y.Z` (maintainers only)

This last step will trigger the CI to build the package and send the version to pypi

When we add new functionality, the past version will be moved to another branch. For example, if we're at version *1.13.7* and a new functionality is implemented, we create a new branch *1.13.x*, and protect it(this way we can't delete it), the new code is merged to master branch, and them we create the tag *1.14.0*

This way we can always fix a past version, opening PRs from *1.13.x* branch.

Python Module Index

f

Index

G

H

I

K

L

M

N

O

P

Q

R

S

T

V

X