model-selection

Library for the selection of machine learning models

Installation

```
pip install ml-model-selection
```

Usage

```
from sklearn.tree import DecisionTreeRegressor
from sklearn.neighbors import KNeighborsRegressor
import model_selection as ms


###### Single dataset X,y (numpy array)

# hyperparameter_validation
model = DecisionTreeRegressor()
hyperparameter = "max_leaf_nodes"
hyperparameter_values = [2,3,4]
(train_val_scores, best_index,
 test_score, ax) = ms.hyperparameter_validation(X, y, model,
hyperparameter, hyperparameter_values, plot=True, plot_train=True)

# hyperparameters_validation
model = DecisionTreeRegressor()
param_grid = {"max_leaf_nodes": [2,3,4],
              "max_features": [None,"sqrt"]}
(params, train_val_scores,
 best_index, test_score) = ms.hyperparameters_validation(X, y,
model, param_grid, time_series=True)

# models_validation
model_paramGrid_list = [
                         ("DT", DecisionTreeRegressor(),
                               {"max_leaf_nodes":[2,3,4],
                                "max_features":[None,"sqrt"]}
                         ),
                         ("kNN", KNeighborsRegressor(),
                                {"n_neighbors":[1,2,3],
                                "weights":["uniform","distance"]}
                         )
                       ]
(models_train_val_score, models_best_params, best_index,
test_score, ax) = ms.models_validation(X, y, model_paramGrid_list,
plot=True)


###### Multiple datasets, in dataset_list
dataset_list = [(X1,y1),(X2,y2),(X3,y3)]

# datasets_hyperparameter_validation
model = DecisionTreeRegressor()
hyperparameter = "max_leaf_nodes"
hyperparameter_values = [2,3,4]
(datasets_train_val_score, datasets_best_hyperparameter_value,
 best_index, test_score,
 axes) = ms.datasets_hyperparameter_validation(dataset_list,
model, hyperparameter, hyperparameter_values, plot=True,
xvalues=["D1","D2","D3"])

# datasets_hyperparameters_validation
model = DecisionTreeRegressor()
param_grid = {"max_leaf_nodes": [2,3,4],
              "max_features": [None,"sqrt"]}
(datasets_train_val_score, datasets_best_params,
 best_index, test_score,
 ax) = ms.datasets_hyperparameters_validation(dataset_list, model,
param_grid, plot=True, xvalues=["D1","D2","D3"])

# datasets_models_validation
model_paramGrid_list = [
                         ("DT", DecisionTreeRegressor(),
                               {"max_leaf_nodes":[2,3,4],
                                "max_features":[None,"sqrt"]}
                         ),
                         ("kNN", KNeighborsRegressor(),
                                {"n_neighbors":[1,2,3],
                                 "weights":["uniform","distance"]}
                          )
                        ]
(datasets_train_val_score, datasets_best_model,
 best_index, test_score,
 axes) = ms.datasets_models_validation(dataset_list,
model_paramGrid_list, time_series=True, plot=True,
xvalues=["D1","D2","D3"])
```

References

- NumPy, the fundamental package for scientific computing with Python.
- Matplotlib is a comprehensive library for creating static, animated, and interactive visualizations in Python.
- scikit-learn, machine Learning in Python.

License

MIT

DESCRIPTION

model-selection is a python library for the selection of machine learning models.

Purpose

The main purpose of this library is to perform the selection of the best machine learning model among several ones.
In other words, the purpose is to perform the validation of different machine learning models.

This library is meant to be simple and intuitive, but also rich. In this way, the user is able to personalize the model selection in a powerful and flexible way.

In addition, the library allows the user to plot the results of the validation, in order to graphically visualize the model selection.

There are several different model selection functions. Each of them is specifically designed for a particular scenario. Certain contexts are more specific, and others are more general: on the whole, the vast majority of the possible scenarios are covered.

This separation of the single model selection task into multiple functions has been made to keep the interface of the library both easy and flexible.

Functionalities

On the whole, there are six different model selection functions, divided into two groups.

1. In the first group there are the functions which perform the model selection with respect to a single dataset.
2. In the second group there are the functions which perform the model selection with respect to multiple datasets.

In particular, the functions in the first group are the following, sorted from the most specific to the most general.

- `hyperparameter_validation`. It performs the tuning of the hyperparameter of a certain model.
- `hyperparameters_validation`. It performs the tuning of some several hyperparameters of a certain model.
- `models_validation`. It performs the model selection among different models.

The functions in the second group are like the ones in the first group, but they perform the model selection among different datasets. This means that not only the best model is selected, but also the best dataset is selected.
The functions in the second group are the following, sorted from the most specific to the most general.

- `datasets_hyperparameter_validation`
- `datasets_hyperparameters_validation`
- `datasets_models_validation`

This library, besides the model selection functions, contains also some other secondary functionalities.
It contains the `PolynomialRegression` class, which is a machine learning model implementing the polynomial regression.
It also contains some functions which compute the training-validation-test scores and the bias^2-variance-error for a certain model on the given dataset.
Finally, it contains a function which is able to plot the predictions made by a certain model on the given dataset against the actual values. This is useful for graphically visualizing the goodness of a certain model.

Implementation details

The selection of the best model is performed computing the validation scores, which means that the best model is the one associated with the best validation score.
In particular, each dataset is split into training and test sets, and then the cross validation strategy is applied on the training set. In this way, the validation score is computed.

Additionally, the training and test scores are also computed. These are simply additional indicators: the selection is performed considering only the validation scores.
While the validation and training scores are computed for each model, the test score is computed only for the best one. The test score can be considered as a final measure of goodness for the best model.

As described above, this library has a rich interface, in order to allow the user to personalize the model selection in a powerful and flexible way.
Some examples are now presented.

- The user can specify whether the machine learning problem is a regression or a classification task. This choice influences the technique used for the selection.
- The user can specify whether the given datasets are time-series datasets or not. This choice influences the technique used for the selection.
- The user can specify whether the explanatory features have to be scaled or not.

This library is built on top of the scikit-learn library.
The machine learning models are indeed represented as scikit-learn models, i.e. they are compliant with the scikit-learn estimators interface.
In addition, under the hood, the selection is performed using the grid search cross validation provided by scikit-learn (i.e. `GridSearchCV`);
Moreover, several other scikit-learn utilities are used.

The datasets are instead represented as NumPy arrays, i.e. they have type `np.ndarray`.

Finally, the plots are made using the Matplotlib library.

DOCUMENTATION

`PolynomialRegression` model

```
class PolynomialRegression(degree=1)
```

Bases: `sklearn.base.BaseEstimator`

Polynomial regression model.

It's a sklearn model: it's compliant to the sklearn estimators interface.

Parameters:

- `degree: int`

Degree to apply for the polynomial transformation.

-[Notes]-
The polynomial transformation is performed using the sklearn `PolynomialFeatures`.

Utility functions

```
def compute_train_val_test(X, y, model, scale=False,
test_size=0.2, time_series=False, random_state=123, n_folds=5,
regr=True)
```

Compute the training-validation-test scores for the given model on the given dataset.

The training and test scores are simply computed by splitting the dataset into the training and test sets. The validation score is performed applying the cross validation on the training set.

Parameters:

- `X: np.array`
  Two-dimensional `np.array`, containing the explanatory features of the dataset.
- `y: np.array`
  Mono dimensional `np.array`, containing the response feature of the dataset.
- `model: sklearn.base.BaseEstimator`
  Model to evaluate.
- `scale: bool`
- Indicates whether to scale or not the features in `X`. (The scaling is performed using the sklearn `MinMaxScaler`).
- `test_size: float`
  Decimal number between 0 and 1, which indicates the proportion of the test set.
- `time_series: bool`
  Indicates if the given dataset is a time series dataset (i.e. datasets indexed by days). (This affects the computing of the scores).
- `random_state: int`
  Used in the training-test splitting of the dataset.
- `n_folds: int`
  Indicates how many folds are made in order to compute the k-fold cross validation. (It's used only if `time_series` is `False`).
- `regr: bool`
  Indicates if it's either a regression or a classification problem.

Returns:

- `train_score: float`
- `val_score: float`
- `test_score: float`

-[Notes]-

- If `regr` is `True`, the returned scores are errors, computed using the MSE formula (i.e. Mean Squared Error). Otherwise, the returned scores are accuracy measures.
- If `time_series` is `False`, the training-test splitting of the dataset is made randomly. In addition, the cross validation strategy performed is the classic k-fold cross validation: the number of folds is specified by `n_folds`. Otherwise, if `time_series` is `True`, the training-test sets are simply obtained by splitting the dataset into two contiguous parts. In addition, the cross validation strategy performed is the sklearn `TimeSeriesSplit`.

```
def compute_bias_variance_error(X, y, model, scale=False, N_TESTS
= 20, sample_size=0.67)
```

Compute the bias^2-variance-error scores for the given model on the given dataset.

These measures are computed in an approximate way, using N_TESTS random samples of size `sample_size` from the dataset.

Parameters:

- `X: np.array`
  Two-dimensional `np.array`, containing the explanatory features of the dataset.
- `y: np.array`
- Mono dimensional `np.array`, containing the response feature of the dataset.
- `model: sklearn.base.BaseEstimator`
  Model to evaluate.
- `scale: bool`
  Indicates whether to scale or not the features in `X`. (The scaling is performed using the sklearn `MinMaxScaler`).
- `N_TESTS: int`
  Number of samples that are made in order to compute the measures.
- `sample_size: float`
  Decimal number between 0 and 1, which indicates the proportion of the sample.

Returns:

- `bias: float`
- `variance: float`
- `error: float`

```
def plot_predictions(X, y, model, scale=False, test_size=0.2,
plot_type=0, xvalues=None, xlabel="Index",
title="Actual vs Predicted values", figsize=(6,6))
```

Plot the predictions made by the given model on the given dataset, versus its actual values.

The dataset is split into the training-test sets: the former is used to train the `model`, on the latter the predictions are made.

Parameters:

- `X: np.array`
  Two-dimensional `np.array`, containing the explanatory features of the dataset.
- `y: np.array`
  Mono dimensional `np.array`, containing the response feature of the dataset.
- `model: sklearn.base.BaseEstimator`
  Model used to make the predictions.
- `scale: bool`
  Indicates whether to scale or not the features in `X`. (The scaling is performed using the sklearn `MinMaxScaler`).
- `test_size: float`
  Decimal number between 0 and 1, which indicates the proportion of the test set.
- `plot_type: int`
  Indicates the type of the plot.
  0 -> In the same plot two different curves are drawn: the first has on the x axis `xvalues` and on the y axis the actual values (i.e. `y`); the second has on the x axis `xvalues` and on the y axis the computed predicted values.
  1 -> On the x axis the actual values are put, on the y axis the predicted ones.
- `xvalues: list` (in general, `iterable`)
  Values that have to be put in the x axis of the plot. (It's used only if `plot_type` is 0).
- `xlabel: str`
  Label of the x axis of the plot. (It's used only if `plot_type` is 0).
- `title: str`
  Title of the plot.
- `figsize: tuple`
  Two dimensions of the plot.

Returns: `matplotlib.axes.Axes`
The matplotlib Axes where the plot has been made.

-[Notes]-
The splitting of the datasets into the training-test sets is simply made by dividing the dataset into two contiguous sequences. I.e. it is the same technique used usually when the dataset is a time series dataset. (This is done in order to simplify the visualization). For this reason, typically this function is applied on time series datasets.

Functions that perform the model selection with respect to a single dataset

```
def hyperparameter_validation(X, y, model, hyperparameter,
hyperparameter_values, scale=False, test_size=0.2,
time_series=False, random_state=123, n_folds=5, regr=True,
plot=False, plot_train=False,  xvalues=None, xlabel=None,
title="Hyperparameter validation", figsize=(6,6))
```

Select the best value for the specified hyperparameter of the specified model on the given dataset.

In other words, perform the tuning of the `hyperparameter` among the values in `hyperparameter_values`.

This selection is made using the validation score (i.e. the best hyperparameter value is the one with the best validation score). The validation score is computed by splitting the dataset into the training-test sets and then by applying the cross validation on the training set. Additionally, the training and test scores are also computed.

Optionally, the validation scores of the `hyperparameter_values` can be plotted, making a graphical visualization of the selection.

Parameters:

- `X: np.array`
  Two-dimensional `np.array`, containing the explanatory features of the dataset.
- `y: np.array`
  Mono dimensional `np.array`, containing the response feature of the dataset.
- `model: sklearn.base.BaseEstimator`
  Model which has the specified `hyperparameter`.
- `hyperparameter: str`
  The name of the hyperparameter that has to be validated.
- `hyperparameter_values: list`
  List of values for `hyperparameter` that have to be taken into account in the selection.
- `scale: bool`
  Indicates whether to scale or not the features in `X`. (The scaling is performed using the sklearn `MinMaxScaler`).
- `test_size: float`

Decimal number between 0 and 1, which indicates the proportion of the test set.

- `time_series: bool`
  Indicates if the given dataset is a time series dataset (i.e. dataset indexed by days). (This affects the computing of the validation score).
- `random_state: int`
  Used in the training-test splitting of the dataset.
- `n_folds: int`
  Indicates how many folds are made in order to compute the k-fold cross validation. (It's used only if `time_series` is `False`).
- `regr: bool`
  Indicates if it's either a regression or a classification problem.
- `plot: bool`
  Indicates whether to plot or not the validation score values.
- `plot_train: bool`
  Indicates whether to plot also the training scores. (It's considered only if `plot` is `True`).
- `xvalues: list` (in general, `iterable`)
  Values that have to be put in the x axis of the plot.
- `xlabel: str`
  Label of the x axis of the plot.
- `title: str`
  Title of the plot.
- `figsize: tuple`
  Two dimensions of the plot.

Returns:

- `train_val_scores: np.array`
  Two dimensional np.array, containing two columns: the first contains the training scores, the second the validation scores. It has as many rows as the number of values in `hyperparameter_values` (i.e. number of values to be tested).
- `best_index: int`
  Index of `hyperparameter_values` that indicates which is the best hyperparameter value.
- `test_score: float`
  Test score associated with the best hyperparameter value.
- `ax: matplotlib.axes.Axes`
  The matplotlib Axes where the plot has been made. If `plot` is `False`, then it is `None`.

-[Notes]-

- If `regr` is `True`, the validation scores are errors (MSE, i.e. Mean Squared Errors): this means that the best hyperparameter value is the one associated with the minimum validation score. Otherwise, the validation scores are accuracies: this

means that the best hyperparameter value is the one associated with the maximum validation score.
- If `time_series` is `False`, the training-test splitting of the dataset is made randomly. In addition, the cross validation strategy performed is the classic k-fold cross validation: the number of folds is specified by `n_folds`. Otherwise, if `time_series` is `True`, the training-test sets are simply obtained by splitting the dataset into two contiguous parts. In addition, the cross validation strategy performed is the sklearn `TimeSeriesSplit`.

```
def hyperparameters_validation(X, y, model, param_grid,
scale=False, test_size=0.2, time_series=False, random_state=123,
n_folds=5, regr=True)
```

Select the best combination of values for the specified hyperparameters of the specified model on the given dataset.

In other words, perform the tuning of multiple hyperparameters. The parameter `param_grid` is a dictionary that indicates which are the specified hyperparameters and what are the associated values to test.

All the possible combinations of values are tested, in an exhaustive way (i.e. grid search).

This selection is made using the validation score (i.e. the best combination of hyperparameters values is the one with the best validation score). The validation score is computed by splitting the dataset into the training-test sets and then by applying the cross validation on the training set. Additionally, the training and test scores are also computed.

Parameters:
- `X: np.array`
  Two-dimensional `np.array`, containing the explanatory features of the dataset.
- `y: np.array`
  Mono dimensional `np.array`, containing the response feature of the dataset.
- `model: sklearn.base.BaseEstimator`
  Model which has the specified hyperparameters.
- `param_grid: dict`
  Dictionary which has as keys the names of the specified hyperparameters and as values the associated list of values to test.
- `scale: bool`
  Indicates whether to scale or not the features in `X`. (The scaling is performed using the sklearn `MinMaxScaler`).
- `test_size: float`
  Decimal number between 0 and 1, which indicates the proportion of the test set.

- `time_series: bool`
  Indicates if the given dataset is a time series dataset (i.e. dataframe indexed by days). (This affects the computing of the validation score).
- `random_state: int`
  Used in the training-test splitting of the dataset.
- `n_folds: int`
  Indicates how many folds are made in order to compute the k-fold cross validation. (It's used only if `time_series` is `False`).
- `regr: bool`
  Indicates if it's either a regression or a classification problem.

Returns:
- `params: list`
  List which enumerates all the possible combinations of hyperparameters values. It's a list of dictionaries: each dictionary represents a specific combination of hyperparameters values. (It's a dictionary which has as keys the hyperparameters names and as values the specific associated values of that combination).
- `train_val_scores: np.array`
  Two dimensional `np.array`, containing two columns: the first contains the training scores, the second the validation scores. It has as many rows as the number of possible combinations of the hyperparameters values. (It has as many rows as the elements of `params`).
- `best_index: int`
  Index of `params` that indicates which is the best combination of hyperparameters values.
- `test_score: float`
  Test score associated with the best combination of hyperparameters values.

-[Notes]-
- If `regr` is `True`, the validation scores are errors (MSE, i.e. Mean Squared Errors): this means that the best combination of hyperparameters values is the one associated with the minimum validation score. Otherwise, the validation scores are accuracies: this means that the best combination of hyperparameters values is the one associated with the maximum validation score.
- If `time_series` is `False`, the training-test splitting of the dataset is made randomly. In addition, the cross validation strategy performed is the classic k-fold cross validation: the number of folds is specified by `n_folds`. Otherwise, if `time_series` is `True`, the training-test sets are simply obtained by splitting the dataset into two contiguous parts. In addition, the cross validation strategy performed is the sklearn `TimeSeriesSplit`.

```
def models_validation(X, y, model_paramGrid_list, scale_list=None,
test_size=0.2, time_series=False, random_state=123,  n_folds=5,
regr=True, plot=False, plot_train=False, xvalues=None,
xlabel="Models",  title="Models validation", figsize=(6,6))
```
Select the best model on the given dataset.

The parameter `model_paramGrid_list` is the list of the models to test. It also contains, for each model, the grid of hyperparameters that have to be tested on that model (i.e. the grid which contains the values to test for each specified hyperparameter of the model). (That grid has the same structure as the `param_grid` parameter of the function `hyperparameters_validation`. See `hyperparameters_validation`).

For each specified model, the best combination of hyperparameters values is selected in an exhaustive way (i.e. grid search). Actually, the function `hyperparameters_validation` is used. (See `hyperparameters_validation`).

The selection of the best model is made using the validation score (i.e. the best model is the one with the best validation score). The validation score is computed by splitting the dataset into the training-test sets and then by applying the cross validation on the training set. Additionally, the training and test scores are also computed.

Optionally, the validation scores of the different models can be plotted, making a graphical visualization of the selection.

Parameters:

- `X: np.array`
  Two-dimensional `np.array`, containing the explanatory features of the dataset.
- `y: np.array`
  Mono dimensional `np.array`, containing the response feature of the dataset.
- `model_paramGrid_list: list`
  List that specifies the models and the relative grids of hyperparameters to be tested. It's a list of triples (i.e. tuples), where each triple represents a model:
  1. the first element is a string, which is a mnemonic name of that model;
  2. the second element is the sklearn model;
  3. the third element is the grid of hyperparameters to test for that model. It's a dictionary, with the same structure of the parameter `param_grid` of the function `hyperparameters_validation`.
- `scale_list: list` or `bool`
  List of booleans, which has as many elements as the number of models to test (i.e. number of elements in the `model_paramGrid_list` list). This list indicates, for each different model, if the features in X have to be scaled or not.
  `scale_list` can be `None` or `False`: in this case the X features aren't scaled for any model. `scale_list` can be `True`: in this case the X features are scaled for all the models.

- `test_size: float`
  Decimal number between 0 and 1, which indicates the proportion of the test set.
- `time_series: bool`
  Indicates if the given dataset is a time series dataset (i.e. dataset indexed by days). (This affects the computing of the validation score).
- `random_state: int`
  Used in the training-test splitting of the dataset.
- `n_folds: int`
  Indicates how many folds are made in order to compute the k-fold cross validation. (It's used only if `time_series` is `False`).
- `regr: bool`
  Indicates if it's either a regression or a classification problem.
- `plot: bool`
  Indicates whether to plot or not the validation score values.
- `plot_train: bool`
  Indicates whether to plot also the training scores. (It's considered only if `plot` is `True`).
- `xvalues: list` (in general, `iterable`)
  Values that have to be put in the x axis of the plot.
- `xlabel: str`
  Label of the x axis of the plot.
- `title: str`
  Title of the plot.
- `figsize: tuple`
  Two dimensions of the plot.

Returns:

- `models_train_val_score: np.array`
  Two dimensional `np.array`, containing two columns: the first contains the training scores, the second the validation scores. It has as many rows as the number of models to test (i.e. number of elements in the `model_paramGrid_list` list).
- `models_best_params: list`
  List which indicates, for each model, the best combination of the hyperparameters values for that model. It has as many elements as the number of models to test (i.e. number of elements in the `model_paramGrid_list` list), and it contains dictionaries: each dictionary represents the best combination of the hyperparameters values for the associated model.
- `best_index: int`
  Index of `model_paramGrid_list` that indicates which is the best model.
- `test_score: float`
  Test score associated with the best model.
- `ax: matplotlib.axes.Axes`
  The matplotlib Axes where the plot has been made. If `plot` is False, then it is `None`.

-[Notes]-

- If `regr` is `True`, the validation scores are errors (MSE, i.e. Mean Squared Errors): this means that the best model is the one associated with the minimum validation score. Otherwise, the validation scores are accuracies: this means that the best model is the one associated with the maximum validation score.
- If `time_series` is `False`, the training-test splitting of the dataset is made randomly. In addition, the cross validation strategy performed is the classic k-fold cross validation: the number of folds is specified by `n_folds`. Otherwise, if `time_series` is `True`, the training-test sets are simply obtained by splitting the dataset into two contiguous parts. In addition, the cross validation strategy performed is the sklearn `TimeSeriesSplit`.

Functions that perform the model selection with respect to multiple datasets

```
def datasets_hyperparameter_validation(dataset_list, model,
hyperparameter, hyperparameter_values, scale=False,
test_size=0.2, time_series=False, random_state=123, n_folds=5,
regr=True, plot=False, plot_train=False, xvalues=None,
xlabel="Datasets", title="Datasets validation", figsize=(6,6)
,verbose=False, figsize_verbose=(6,6))
```

Select the best dataset and the best value for the specified hyperparameter of the specified model (i.e. select the best couple dataset-hyperparameter value).

For each dataset in `dataset_list`, all the specified values `hyperparameter_values` are tested for the specified `hyperparameter` of `model`. In other words, on each dataset the tuning of `hyperparameter` is performed: in fact, on each dataset, the function `hyperparameter_validation` is applied. (See `hyperparameter_validation`). In the end, the best couple dataset-hyperparameter value is selected.

Despite the fact that a couple dataset-hyperparameter value is selected, the main viewpoint is focused with respect to the datasets. It's a validation focused on the datasets. In fact, first of all, for each dataset the hyperparameter tuning is performed: in this way the best value is selected and its relative score is associated with the dataset (i.e. it's the score of the dataset). (In other words, on each dataset the function `hyperparameter_validation` is applied).
Finally, after that, the best dataset is selected. It's a two-levels selection.

This selection is made using the validation score (i.e. the best couple dataset-hyperparameter value is the one with the best validation score). The validation score is computed by splitting each dataset into the training-test sets and then by applying the cross validation on the training set.
Additionally, the training and test scores are also computed.

Optionally, the validation scores of the datasets can be plotted, making a graphical visualization of the dataset selection. This is the 'main' plot. Moreover, still optionally, the 'secondary' plots can be done: for each dataset, the validation scores of the `hyperparameter_values` are plotted, making a graphical visualization of the hyperparameter tuning on that dataset. (As the plot made by the `hyperparameter_validation` function).

Parameters:

- `dataset_list: list`
  List of couples, where each couple is a dataset.
  1. The first element is `X`, the two-dimensional `np.array` containing the explanatory features of the dataset.
  2. The second element is `y`, the mono dimensional `np.array` containing the response feature of the dataset.
- `model: sklearn.base.BaseEstimator`
  Model which has the specified `hyperparameter`.
- `hyperparameter: str`
  The name of the hyperparameter that has to be validated.
- `hyperparameter_values: list`
  List of values for `hyperparameter` that have to be taken into account in the selection.
- `scale: bool`
  Indicates whether to scale or not the features in `X` (for all the datasets). (The scaling is performed using the sklearn `MinMaxScaler`).
- `test_size: float`
  Decimal number between 0 and 1, which indicates the proportion of the test set (for each dataset).
- `time_series: bool`
  Indicates if the given datasets are time series dataset (i.e. datasets indexed by days). (This affects the computing of the validation scores).
- `random_state: int`
  Used in the training-test splitting of the datasets.
- `n_folds: int`
  Indicates how many folds are made in order to compute the k-fold cross validation. (It's used only if `time_series` is `False`).
- `regr: bool`
  Indicates if it's either a regression or a classification problem.
- `plot: bool`

Indicates whether to plot or not the validation score values of the datasets (this is the 'main' plot).

- `plot_train: bool`
  Indicates whether to plot also the training scores (both in the 'main' and 'secondary' plots).
- `xvalues: list` (in general, `iterable`)
  Values that have to be put in the x axis of the 'main' plot.
- `xlabel: str`
  Label of the x axis of the 'main' plot.
- `title: str`
  Title of the 'main' plot.
- `figsize: tuple`
  Two dimensions of the 'main' plot.
- `verbose: bool`
  If `True`, for each dataset the validation scores of the hyperparameter tuning are plotted (these are the 'secondary' plots). (See `hyperparameter_validation`).
- `figsize_verbose: tuple`
  Two dimensions of the 'secondary' plots.

Returns:

- `datasets_train_val_score: np.array`
  Two dimensional `np.array`, containing two columns: the first contains the training scores, the second the validation scores. It has as many rows as the number of datasets to test, i.e. as the number of elements in `dataset_list`.
- `datasets_best_hyperparameter_value: list`
  List which has as many elements as the number of the datasets (i.e. as the number of elements in `dataset_list`). For each dataset, it contains the best `hyperparameter` value on that dataset.
- `best_index: int`
  Index of `dataset_list` that indicates which is the best dataset.
- `test_score: float`
  Test score associated with the best couple dataset-hyperparameter value.
- `axes: list`
  List of the matplotlib Axes where the plots have been made.
  Firstly, the 'secondary' plots are put (if any). And, as last, the 'main' plot is put (if any).
  If no plot has been made, `axes` is an empty list.

-[Notes]-

- If `regr` is `True`, the validation scores are errors (MSE, i.e. Mean Squared Errors): this means that the best couple dataset-hyperparameter value is the one associated with the minimum validation score. Otherwise, the validation scores are accuracies: this means that the best couple is the one associated with the maximum validation score.

- If `time_series` is `False`, the training-test splitting of each dataset is made randomly. In addition, the cross validation strategy performed is the classic k-fold cross validation: the number of folds is specified by `n_folds`. Otherwise, if `time_series` is `True`, the training-test sets are simply obtained by splitting each dataset into two contiguous parts. In addition, the cross validation strategy performed is the sklearn `TimeSeriesSplit`.

```
def datasets_hyperparameters_validation(dataset_list, model,
param_grid, scale=False, test_size=0.2, time_series=False,
random_state=123, n_folds=5, regr=True, plot=False,
plot_train=False, xvalues=None, xlabel="Datasets", title="Datasets
validation",figsize=(6,6))
```

Select the best dataset and the best combination of values for the specified hyperparameters of the specified model (i.e. select the best couple dataset-combination of hyperparameters values).

For each dataset in `dataset_list`, all the possible combinations of the hyperparameters values for `model` (specified with `param_grid`) are tested. In other words, on each dataset the tuning of the specified hyperparameters is performed in an exhaustive way: in fact, on each dataset, the function `hyperparameters_validation` is applied. (See `hyperparameters_validation`).
In the end, the best couple dataset-combination of hyperparameters values is selected.

Despite the fact that a couple dataset-combination of hyperparameters values is selected, the main viewpoint is focused with respect to the datasets. It's a validation focused on the datasets. In fact, first of all, for each dataset the hyperparameters tuning is performed: in this way the best combination of values is selected and its relative score is associated with the dataset (i.e. it's the score of the dataset). (In other words, on each dataset the function `hyperparameters_validation` is applied). Finally, after that, the best dataset is selected. It's a two-levels selection.

This selection is made using the validation score (i.e. the best couple dataset-combination of hyperparameters values, is the one with best validation score). The validation score is computed by splitting each dataset into the training-test sets and then by applying the cross validation on the training set.
Additionally, the training and test scores are also computed.

Optionally, the validation scores of the datasets can be plotted, making a graphical visualization of the dataset selection.

Parameters:

- `dataset_list: list`
  List of couples, where each couple is a dataset.
  1. The first element is `X`, the two-dimensional `np.array` containing the explanatory features of the dataset.

2. The second element is `y`, the mono dimensional `np.array` containing the response feature of the dataset.

- `model: sklearn.base.BaseEstimator`
  Model which has the specified hyperparameters.
- `param_grid: dict`
  Dictionary which has as keys the names of the specified hyperparameters and as values the associated list of values to test.
- `scale: bool`
  Indicates whether to scale or not the features in `X` (for all the datasets). (The scaling is performed using the sklearn `MinMaxScaler`).
- `test_size: float`
  Decimal number between 0 and 1, which indicates the proportion of the test set (for each dataset).
- `time_series: bool`
  Indicates if the given datasets are time series datasets (i.e. datasets indexed by days). (This affects the computing of the validation score).
- `random_state: int`
  Used in the training-test splitting of the datasets.
- `n_folds: int`
  Indicates how many folds are made in order to compute the k-fold cross validation. (It's used only if `time_series` is `False`).
- `regr: bool`
  Indicates if it's either a regression or a classification problem.
- `plot: bool`
  Indicates whether to plot or not the validation score values of the datasets.
- `plot_train: bool`
  Indicates whether to plot also the training scores. (It's considered only if `plot` is `True`).
- `xvalues: list` (in general, `iterable`)
  Values that have to be put in the x axis of the plot.
- `xlabel: str`
  Label of the x axis of the plot.
- `title: str`
  Title of the plot.
- `figsize: tuple`
  Two dimensions of the plot.

Returns:

- `datasets_train_val_score: np.array`
  Two dimensional np.array, containing two columns: the first contains the training scores, the second the validation scores. It has as many rows as the number of datasets to test, i.e. as the number of elements in `dataset_list`.
- `datasets_best_params: list`

List which has as many elements as the number of datasets (i.e. as number of elements in `dataset_list`). For each dataset, it contains the best combination of hyperparameters values on that dataset. Each combination is represented as a dictionary, with keys the hyperparameters names and values the associated values.

- `best_index: int`
  Index of `dataset_list` that indicates which is the best dataset.
- `test_score: float`
  Test score associated with the best couple dataset-combination of hyperparameters values.
- `ax: matplotlib.axes.Axes`
  The matplotlib Axes where the plot has been made.

-[Notes]-

- If `regr` is `True`, the validation scores are errors (MSE, i.e. Mean Squared Errors): this means that the best couple dataset-combination of hyperparameters values is the one associated with the minimum validation score. Otherwise, the validation scores are accuracies: this means that the best couple is the one associated with the maximum validation score.
- If `time_series` is `False`, the training-test splitting of each dataset is made randomly. In addition, the cross validation strategy performed is the classic k-fold cross validation: the number of folds is specified by `n_folds`. Otherwise, if `time_series` is `True`, the training-test sets are simply obtained by splitting each dataset into two contiguous parts. In addition, the cross validation strategy performed is the sklearn `TimeSeriesSplit`.

```
def datasets_models_validation(dataset_list, model_paramGrid_list,
scale_list=None, test_size=0.2, time_series=False,
random_state=123, n_folds=5, regr=True, plot=False,
plot_train=False, xvalues=None,  xlabel="Datasets",
title="Datasets validation", figsize=(6,6) ,verbose=False,
figsize_verbose=(6,6))
```

Select the best dataset and the best model (i.e. select the best couple dataset-model).

For each dataset in `dataset_list`, all the models in `model_paramGrid_list` are tested: each model is tested performing an exhaustive tuning of the specified hyperparameters. In fact, `model_paramGrid_list` also contains, for each model, the grid of the hyperparameters that have to be tested on that model (i.e. the grid which contains the values to test for each specified hyperparameter of the model). In other words, on each dataset the selection of the best model is performed: in fact, on each dataset, the function `models_validation` is applied. (See `models_validation`). In the end, the best couple dataset-model is selected.

Despite the fact that a couple dataset-model is selected, the main viewpoint is focused with respect to the datasets. It's a validation focused on the datasets. In fact, first of all, for each dataset the model selection is performed: in this way the best model is selected and its relative score is associated with the dataset (i.e. it's the score of the dataset). (In other words, on each dataset the function `models_validation` is applied). Finally, after that, the best dataset is selected. It's a two-levels selection.

This selection is made using the validation score (i.e. the best couple dataset-model is the one with best validation score). The validation score is computed by splitting each dataset into the training-test sets and then by applying the cross validation on the training set. Additionally, the training and test scores are also computed.

Optionally, the validation scores of the datasets can be plotted, making a graphical visualization of the dataset selection. This is the 'main' plot. Moreover, still optionally, the 'secondary' plots can be done: for each dataset, the validation scores of the models are plotted, making a graphical visualization of the models selection on that dataset. (As the plot made by the `models_validation` function).

Parameters:

- `dataset_list: list`
  List of couples, where each couple is a dataset.
  1. The first element is `X`, the two-dimensional `np.array` containing the explanatory features of the dataset.
  2. The second element is `y`, the mono dimensional `np.array` containing the response feature of the dataset.
- `model_paramGrid_list: list`
  List that specifies the models and the relative grid of hyperparameters to be tested. It's a list of triples (i.e. tuples), where each triple represents a model:
  1. the first element is a string, which is a mnemonic name of that model;
  2. the second element is the sklearn model;
  3. the third element is the grid of hyperparameters to test for that model. It's a dictionary, with the same structure of parameter `param_grid` of the function `hyperparameters_validation`.
- `scale_list: list` or `bool`
  List of booleans, which has as many elements as the number of models to test (i.e. number of elements in the `model_paramGrid_list` list). This list indicates, for each different model, if the features in `X` have to be scaled or not (for all the datasets).
  `scale_list` can be `None` or `False`: in this case the `X` features aren't scaled for any model. `scale_list` can be `True`: in this case the `X` features are scaled for all the models.
- `test_size: float`
  Decimal number between 0 and 1, which indicates the proportion of the test set (for each dataset).

- `time_series: bool`
  Indicates if the given datasets are time series dataset (i.e. datasets indexed by days). (This affects the computing of the validation score).
- `random_state: int`
  Used in the training-test splitting of the datasets.
- `n_folds: int`
  Indicates how many folds are made in order to compute the k-fold cross validation. (It's used only if `time_series` is `False`).
- `regr: bool`
  Indicates if it's either a regression or a classification problem.
- `plot: bool`
  Indicates whether to plot or not the validation score values of the datasets (i.e. this is the 'main' plot).
- `plot_train: bool`
  Indicates whether to plot also the training scores (both in the 'main' and 'secondary' plots).
- `xvalues: list` (in general, `iterable`)
  Values that have to be put in the x axis of the 'main' plot.
- `xlabel: str`
  Label of the x axis of the 'main' plot.
- `title: str`
  Title of the 'main' plot.
- `figsize: tuple`
  Two dimensions of the 'main' plot.
- `verbose: bool`
  If `True`, for each dataset the validation scores of the models are plotted (i.e. these are the 'secondary' plots). (See `models_validation`).
- `figsize_verbose: tuple`
  Two dimensions of the 'secondary' plots.

Returns:
- `datasets_train_val_score: np.array`
  Two dimensional np.array, containing two columns: the first contains the training scores, the second the validation scores. It has as many rows as the number of datasets to test, i.e. as the number of elements in `dataset_list`.
- `datasets_best_model: list`
  List which has as many elements as the number of the datasets (i.e. number of elements in `dataset_list`). For each dataset, it contains the best model for that dataset.
  More precisely, it is a list of triple:
  1. the first element is the index of `model_paramGrid_list` which indicates the best model;
  2. the second element is the mnemonic name of the best model;

3. the third element is the best combination of hyperparameters values on that best model (i.e. it's a dictionary which has as keys the hyperparameters names and as values their associated values).

- `best_index: int`
  Index of `dataset_list` that indicates which is the best dataset.
- `test_score: float`
  Test score associated with the best couple dataset-model.
- `axes: list`
  List of the matplotlib Axes where the plots have been made.
  Firstly, the 'secondary' plots are put (if any). And, as last, the 'main' plot is put (if any).
  If no plot has been made, `axes` is an empty list.

-[Notes]-

- If `regr` is `True`, the validation scores are errors (MSE, i.e. Mean Squared Errors): this means that the best couple dataset-model is the one associated with the minimum validation score. Otherwise, the validation scores are accuracies: this means that the best couple is the one associated with the maximum validation score.
- If `time_series` is `False`, the training-test splitting of each dataset is made randomly. In addition, the cross validation strategy performed is the classic k-fold cross validation: the number of folds is specified by `n_folds`. Otherwise, if `time_series` is `True`, the training-test sets are simply obtained by splitting each dataset into two contiguous parts. In addition, the cross validation strategy performed is the sklearn `TimeSeriesSplit`.

EXAMPLES

```
>>> import model_selection as ms
```

Prerequisites

In the examples shown in this document, the air pollution datasets of EEA will be used
For this reason, the EEA-datasets-handler library will be used.
In particular,the PM10 mean concentrations in Italy are considered, with respect to 2020.

```
>>> import EEA_datasets_handler as eea
```

```
# Download the datasets
# IT'S NECESSARY ONLY IF THEY HAVEN'T BEEN DOWNLOADED YET
>>> dest_path = "C:\\Datasets"
>>> countries_cities_dict = {"IT": "all"}
>>> pollutants = ["PM10"]
>>> years = [2020]
>>> eea.download_datasets(dest_path, countries_cities_dict,
pollutants, years)

# Load the datasets
>>> source_path = "C:\\Datasets\\EEA"
>>> countries_cities_dict = {"IT":"all"}
>>> pollutants = ["PM10"]
>>> years = [2020]
>>> df = eea.load_datasets(source_path, countries_cities_dict,
pollutants, years)
```

In addition, the timeSeries-processing library will be used in order to process the time series datasets.

```
>>> import timeSeries_processing as tsp
```

Utilities

PolynomialRegressor

It's a simple class, compliant with the sklearn estimators interface, which implements the polynomial regression.

```
>>> model = ms.PolynomialRegression(degree=3)
```

compute_train_val_test

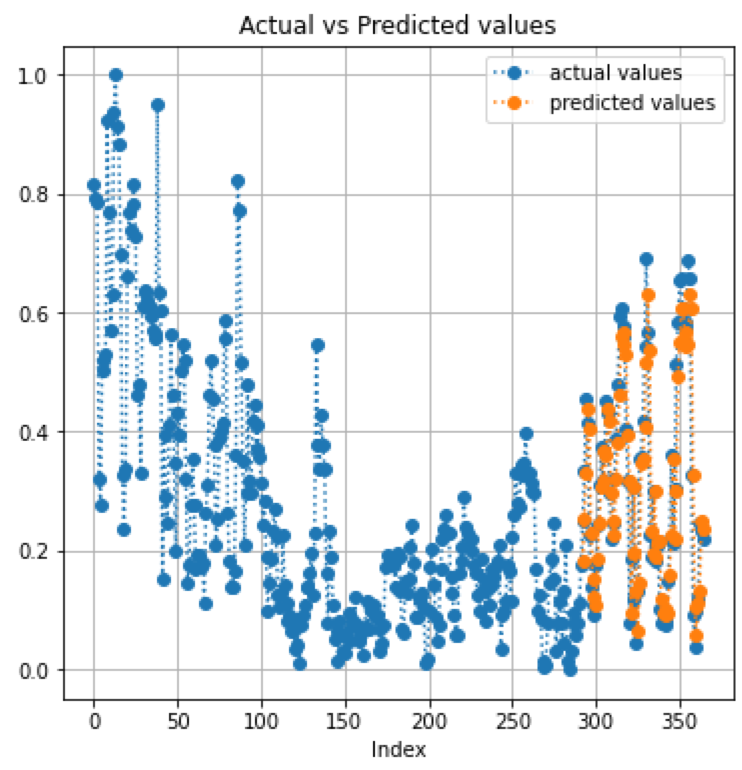
```
>>> df_mean,df_min,df_max = eea.preprocessing(df, fill=True,
fill_n_days=10 ,fill_aggr="mean")
UserWarning: Missing days: ['2020-01-30', '2020-01-31',
'2020-02-01', '2020-02-02', '2020-02-03', '2020-02-04',
'2020-02-05', '2020-02-06', '2020-02-07', '2020-02-08',
'2020-02-09', '2020-02-10', '2020-02-11']
>>> df_mean_supp, X, y = tsp.add_k_previous_days(df_mean,
col_name="mean", k=1, y_col="mean", scale_y=True)

>>> (train_score, val_score,
test_score) = ms.compute_train_val_test(X, y, model)
>>> train_score, val_score, test_score
(0.010590308174761493, 0.011154041214801094, 0.011147349084587836)
```

compute_bias_variance_error

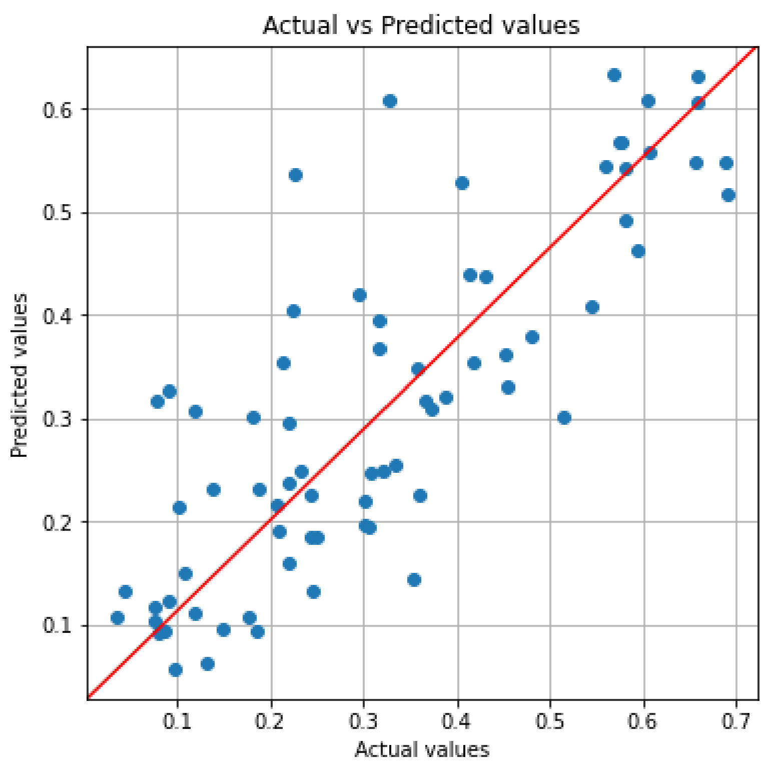
```
>>> bias2, variance, error = ms.compute_bias_variance_error(X, y,
model)
>>> bias2, variance, error
(0.010697516686507456, 0.0002482539041854526, 0.01094577059069291)
```

plot_predictions

```
>>> ms.plot_predictions(X, y, model)
<AxesSubplot:title={'center':'Actual vs Predicted values'},
xlabel='Index'>
```



```
>>> ms.plot_predictions(X, y, model, plot_type=1)
<AxesSubplot:title={'center':'Actual vs Predicted values'},
xlabel='Actual values', ylabel='Predicted values'>
```



Functions that perform the model selection with respect to a single dataset

Create the single dataset.

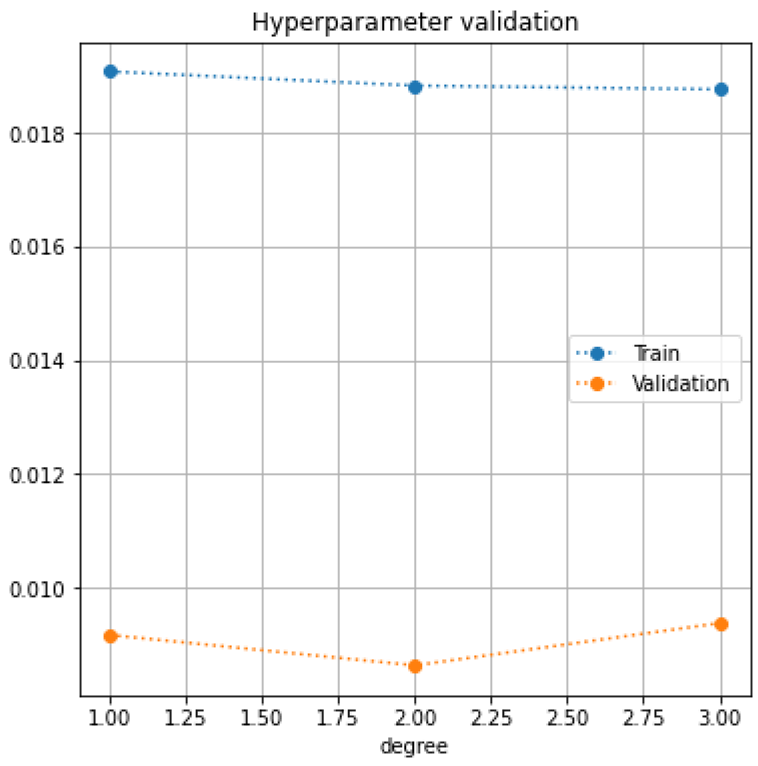
```
>>> df_mean,df_min,df_max = eea.preprocessing(df, fill=True,
fill_n_days=10 ,fill_aggr="mean")
```

```
UserWarning: Missing days: ['2020-01-30', '2020-01-31',
'2020-02-01', '2020-02-02', '2020-02-03', '2020-02-04',
'2020-02-05', '2020-02-06', '2020-02-07', '2020-02-08',
'2020-02-09', '2020-02-10', '2020-02-11']
>>> df_mean_supp, X, y = tsp.add_k_previous_days(df_mean,
col_name="mean", k=1, y_col="mean", scale_y=True)
```

hyperparameter_validation

It selects the best value for the specified hyperparameter of the specified model on the given dataset.

```
>>> model = ms.PolynomialRegression()
>>> hyperparameter = "degree"
>>> hyperparameter_values = [1,2,3]

>>> (train_val_scores, best_index,
test_score, ax) = ms.hyperparameter_validation(X, y, model,
hyperparameter, hyperparameter_values, time_series=True,
plot=True, plot_train=True)
```



```
>>> train_val_scores
```

```
[[0.01907515 0.00917335]
 [0.01882686 0.00863946]
 [0.01876097 0.0093821 ]]

>>> best_index
1

>>> test_score
0.011503918056149346
```

hyperparameters_validation

It selects the best combination of values for the specified hyperparameters of the specified model on the given dataset.

```
>>> from sklearn.tree import DecisionTreeRegressor
>>> model = DecisionTreeRegressor()
>>> param_grid = {"max_leaf_nodes": [2,3,4],
                  ”max_features": [None,"sqrt"]}

>>> (params, train_val_scores,
best_index, test_score) = ms.hyperparameters_validation(X, y,
model, param_grid, time_series=True)
```

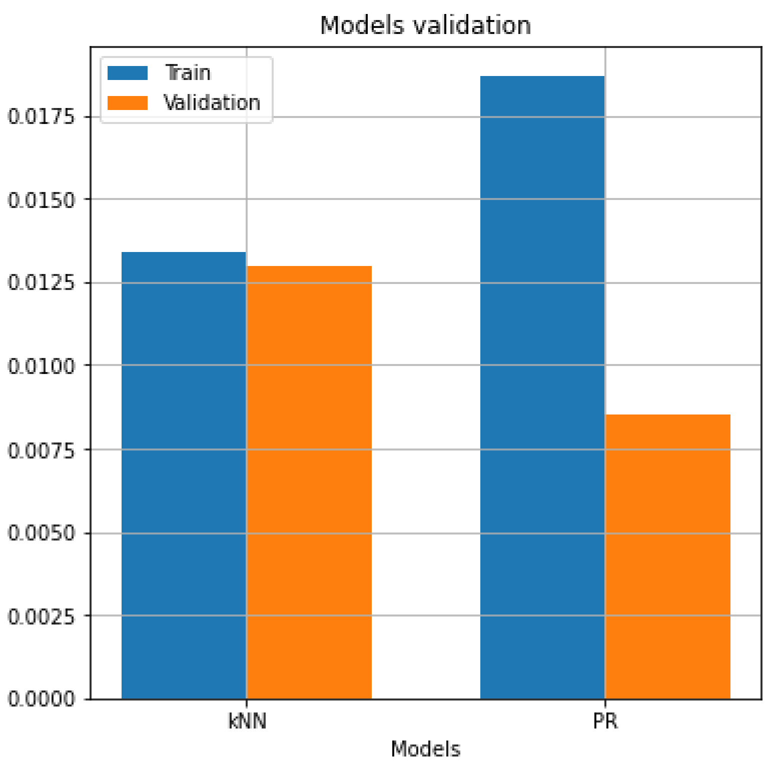
```
>>> params
[{'max_features': None, 'max_leaf_nodes': 2}, {'max_features':
None, 'max_leaf_nodes': 3}, {'max_features': None,
'max_leaf_nodes': 4}, {'max_features': 'sqrt', 'max_leaf_nodes':
2}, {'max_features': 'sqrt', 'max_leaf_nodes': 3},
{'max_features': 'sqrt', 'max_leaf_nodes': 4}]

>>> train_val_scores
[[0.02625313 0.02248355]
 [0.02159088 0.02162117]
 [0.01829289 0.01602048]
 [0.02625313 0.02248355]
 [0.02159088 0.02162117]
 [0.01829289 0.01602048]]
```

```
>>> best_index
2

>>> test_score
0.015266789104509664
```

models_validation

It selects the best model on the given dataset.

```
>>> from sklearn.neighbors import KNeighborsRegressor

>>> model_paramGrid_list = [
                         ("kNN", KNeighborsRegressor(),
                             {"n_neighbors":[1,2,3],
                              "weights":["uniform","distance"]}
                          ),
                         ("PR", ms.PolynomialRegression(),
                             {"degree":[1,2,3]}),
                       ]

>>> (models_train_val_score, models_best_params, best_index,
test_score, ax) = ms.models_validation(X, y, model_paramGrid_list,
scale_list=[True,False], time_series=True, plot=True,
plot_train=True)
```



```
>>> models_train_val_score
[[0.01414521 0.01280187]
 [0.01882686 0.00863946]]

>>> models_best_params
[{'n_neighbors': 3, 'weights': 'uniform'}, {'degree': 2}]

>>> best_index
1

>>> test_score
0.011503918056149346
```

Functions that perform the model selection with respect to multiple datasets

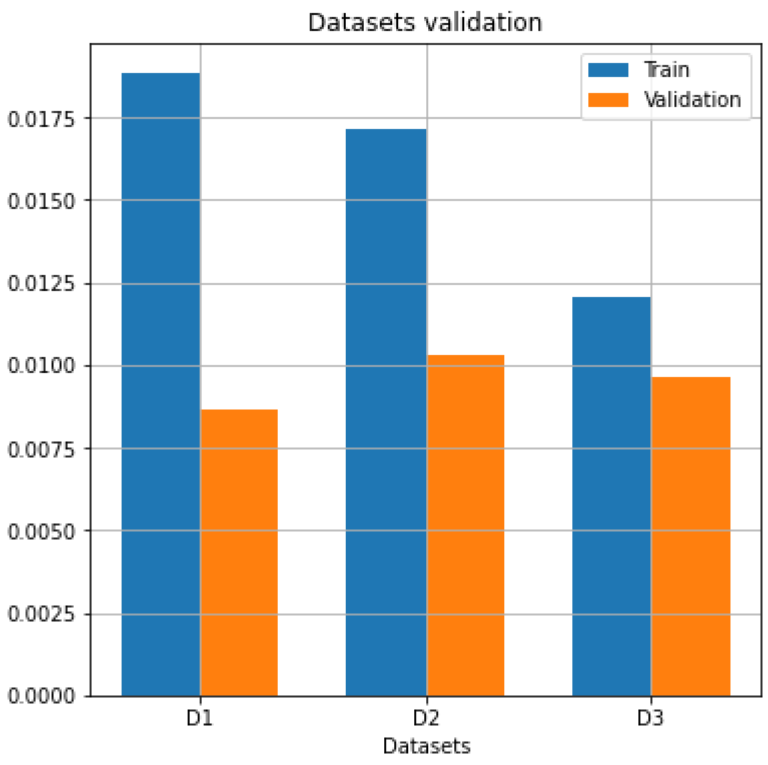
```
# First dataset
```

```
>>> _, X1, y1 = tsp.add_k_previous_days(df_mean, col_name="mean",
k=1, y_col="mean", scale_y=True)
# Second dataset
>>> _, X1, y1 = tsp.add_k_previous_days(df_mean, col_name="mean",
k=1, y_col="mean", scale_y=True)
# Third dataset
>>> _, X3, y3 = tsp.add_k_previous_days(df_mean, col_name="mean",
k=15, y_col="mean", scale_y=True)
```

datasets_hyperparameter_validation

It selects the best dataset and the best value for the specified hyperparameter of the specified model (i.e. it selects the best couple dataset-hyperparameter value).

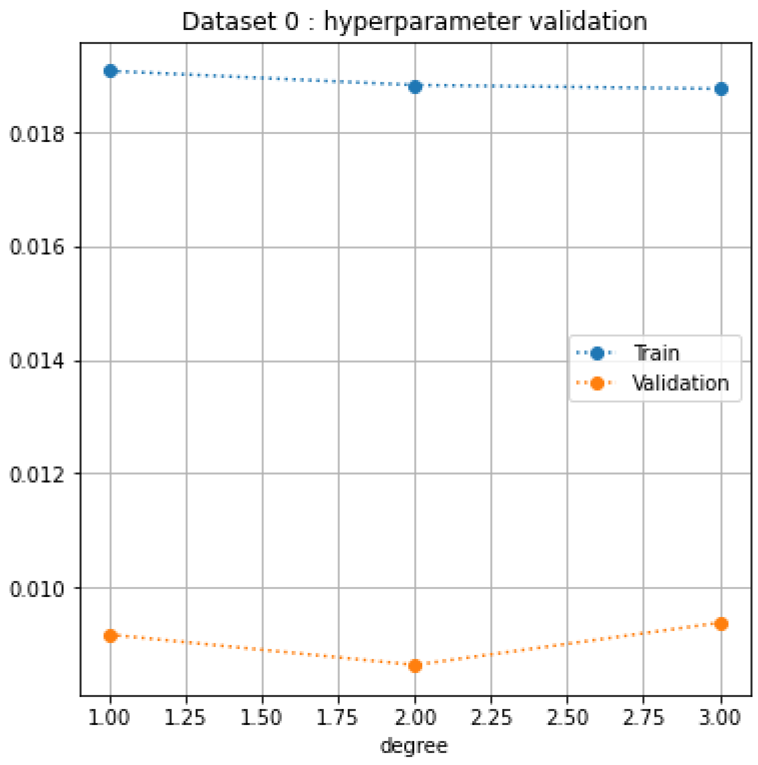
```
>>> dataset_list = [(X1,y1),(X2,y2),(X3,y3)]

>>> model = ms.PolynomialRegression()
>>> hyperparameter = "degree"
>>> hyperparameter_values = [1,2,3]

>>> (datasets_train_val_score, datasets_best_hyperparameter_value,
 best_index, test_score,
 axes) = ms.datasets_hyperparameter_validation(dataset_list,
model, hyperparameter,  hyperparameter_values, time_series=True,
plot=True, plot_train=True, xvalues=["D1","D2","D3"])
```

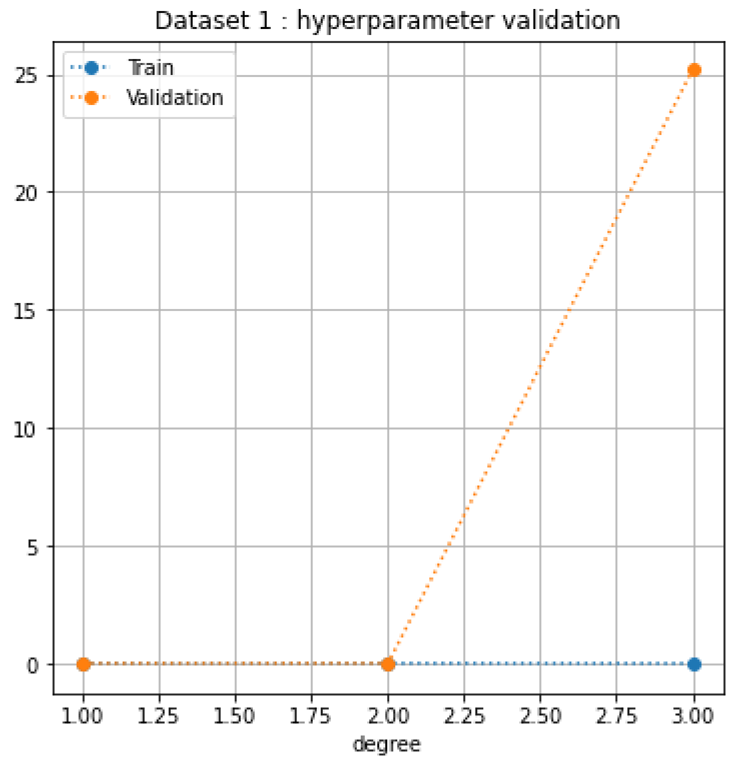


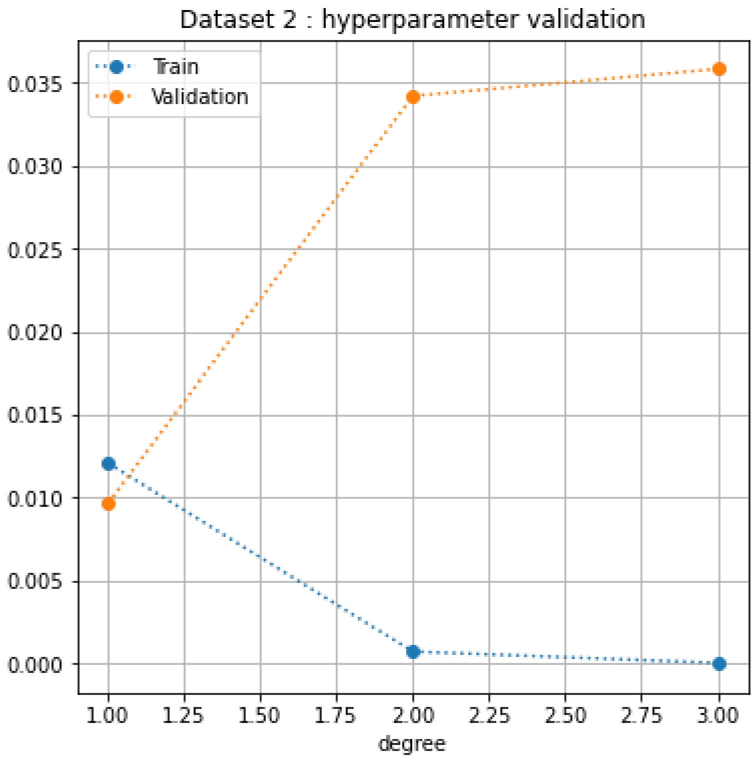
```
>>> datasets_train_val_score
[[0.01882686 0.00863946]
 [0.01713774 0.01030532]
 [0.012072   0.00963146]]

>>> datasets_best_hyperparameter_value
[2, 1, 1]

>>> best_index
0

>>> test_score
0.011503918056149346
```

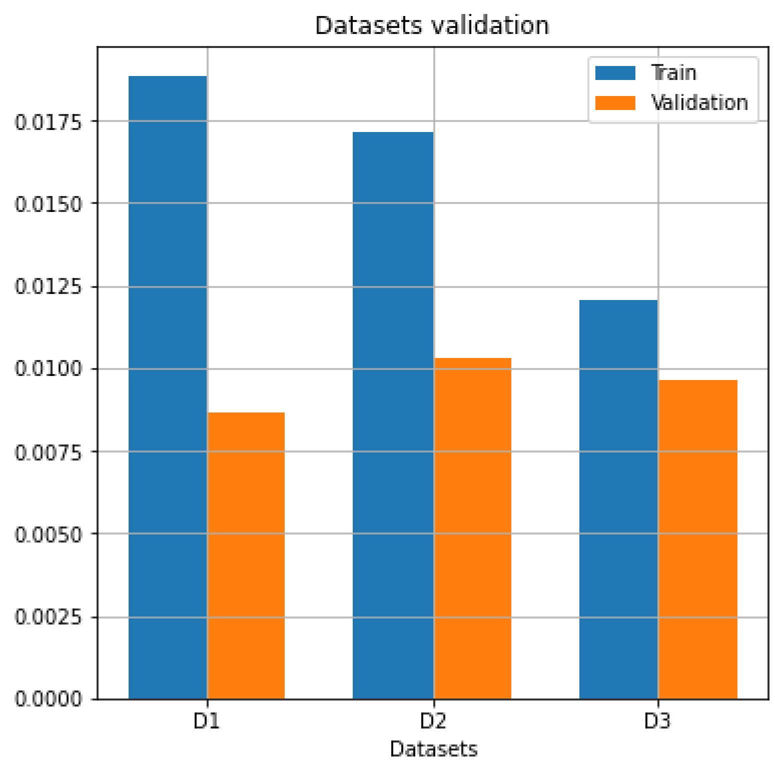
verbose = True

```
>>> dataset_list = [(X1,y1),(X2,y2),(X3,y3)]

>>> model = ms.PolynomialRegression()
>>> hyperparameter = "degree"
>>> hyperparameter_values = [1,2,3]
>>> (datasets_train_val_score, datasets_best_hyperparameter_value,
 best_index, test_score,
 axes) = ms.datasets_hyperparameter_validation(dataset_list,
model, hyperparameter, hyperparameter_values, time_series=True,
plot=True, plot_train=True, xvalues=["D1","D2","D3"],
verbose=True)
```




Dataset 1 : hyperparameter validation
Train
Validation
25
20
15
10
5
0
1.00
1.25
1.50
1.75
2.00
2.25
2.50
2.75
3.00
degree

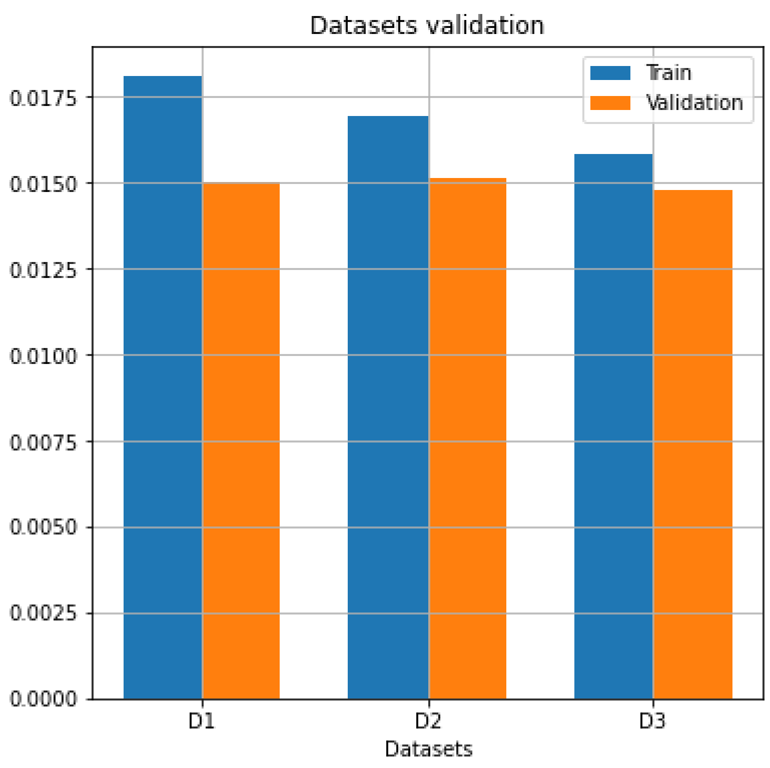

Dataset 2 : hyperparameter validation
Train
Validation
0.035
0.030
0.025
0.020
0.015
0.010
0.005
0.000
1.00
1.25
1.50
1.75
2.00
2.25
2.50
2.75
3.00
degree



datasets_hyperparameters_validation

It selects the best dataset and the best combination of values for the specified hyperparameters of the specified model (i.e. it selects the best couple dataset-combination of hyperparameters values).

```
>>> from sklearn.tree import DecisionTreeRegressor

>>> dataset_list = [(X1,y1),(X2,y2),(X3,y3)]

>>> model = DecisionTreeRegressor()
>>> param_grid = {"max_leaf_nodes": [2,3,4],
                  "max_features": [None,"sqrt"]}

>>> (datasets_train_val_score, datasets_best_params,
 best_index, test_score,
 ax) = ms.datasets_hyperparameters_validation(dataset_list, model,
param_grid, time_series=True, plot=True, plot_train=True,
xvalues=["D1","D2","D3"])
```

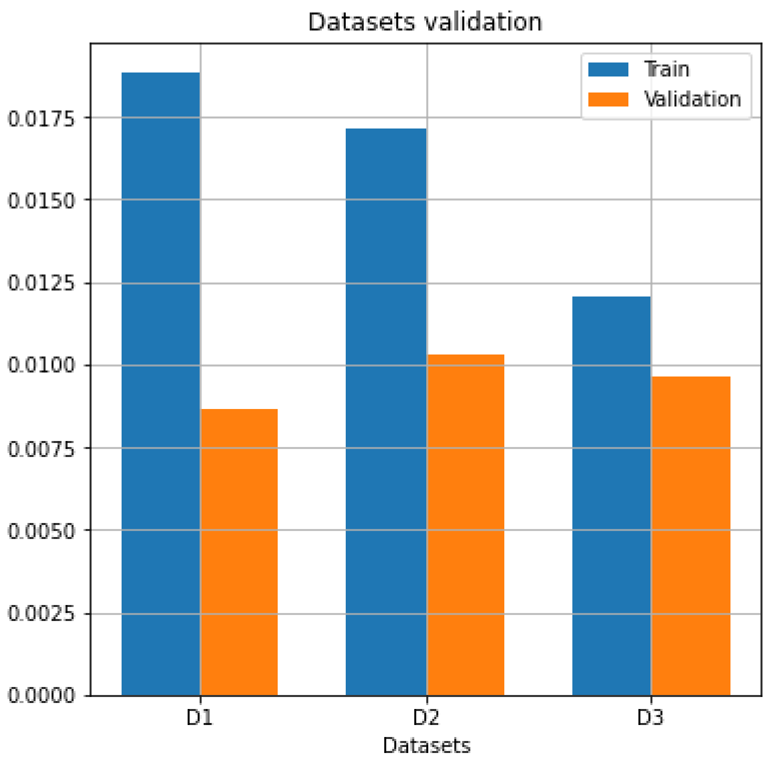


```
>>> datasets_train_val_score
[[0.01808403 0.0150039 ]
 [0.01692864 0.0151333 ]
 [0.01580857 0.0147613 ]]

>>> datasets_best_params
[{'max_features': None, 'max_leaf_nodes': 4}, {'max_features':
None, 'max_leaf_nodes': 4}, {'max_features': None,
'max_leaf_nodes': 4}]

>>> best_index
2

>>> test_score
0.01711360531135875
```

datasets_models_validation

It selects the best dataset and the best model (i.e. it selects the best couple dataset-model). *It's the most general function.*

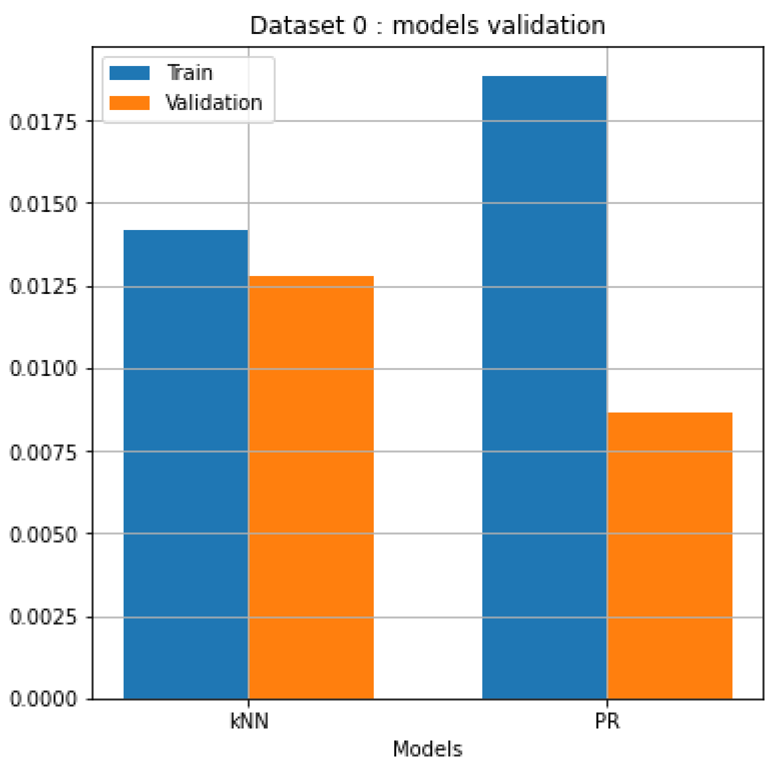
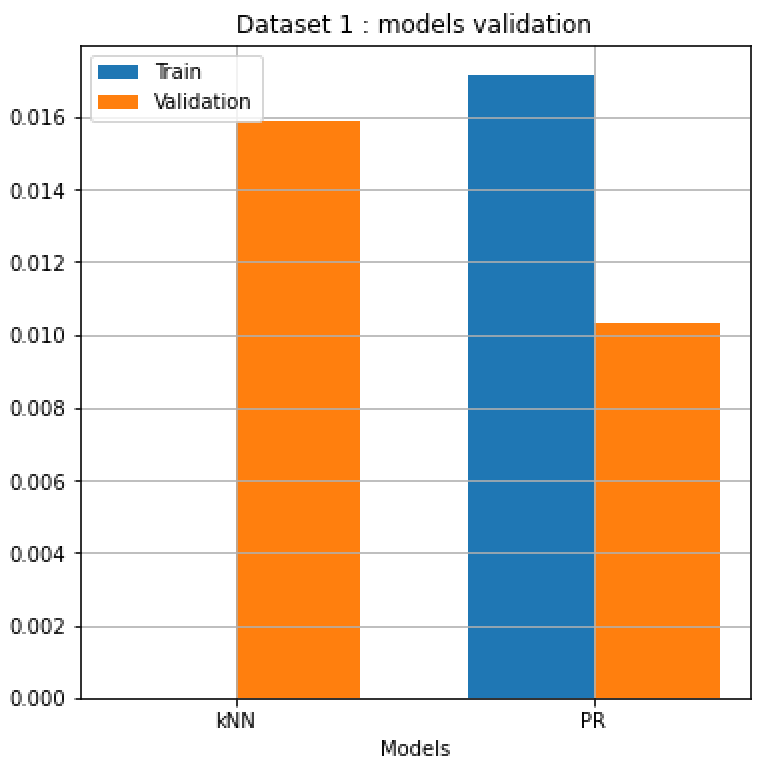
```
>>> from sklearn.neighbors import KNeighborsRegressor

>>> dataset_list = [(X1,y1),(X2,y2),(X3,y3)]

>>> model_paramGrid_list = [
                          ("kNN", KNeighborsRegressor(),
                              {"n_neighbors":[1,2,3],
                               "weights":["uniform","distance"]}
                           ),
                          ("PR", ms.PolynomialRegression(),
                              {"degree":[1,2,3]}),
]

>>> (datasets_train_val_score, datasets_best_model,
 best_index, test_score,
 axes) = ms.datasets_models_validation(dataset_list,
model_paramGrid_list, time_series=True, plot=True,
plot_train=True, xvalues=["D1","D2","D3"])
```

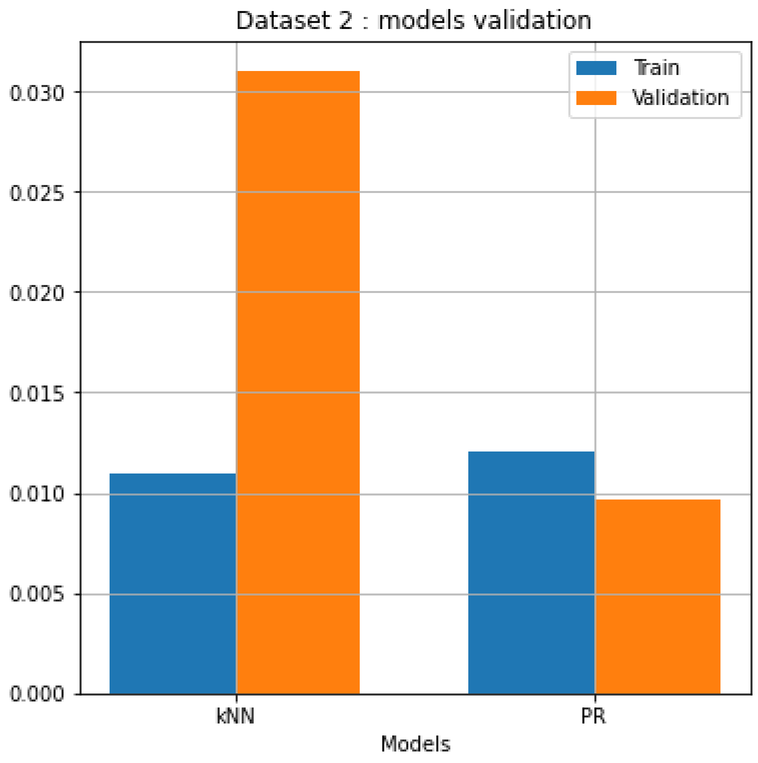


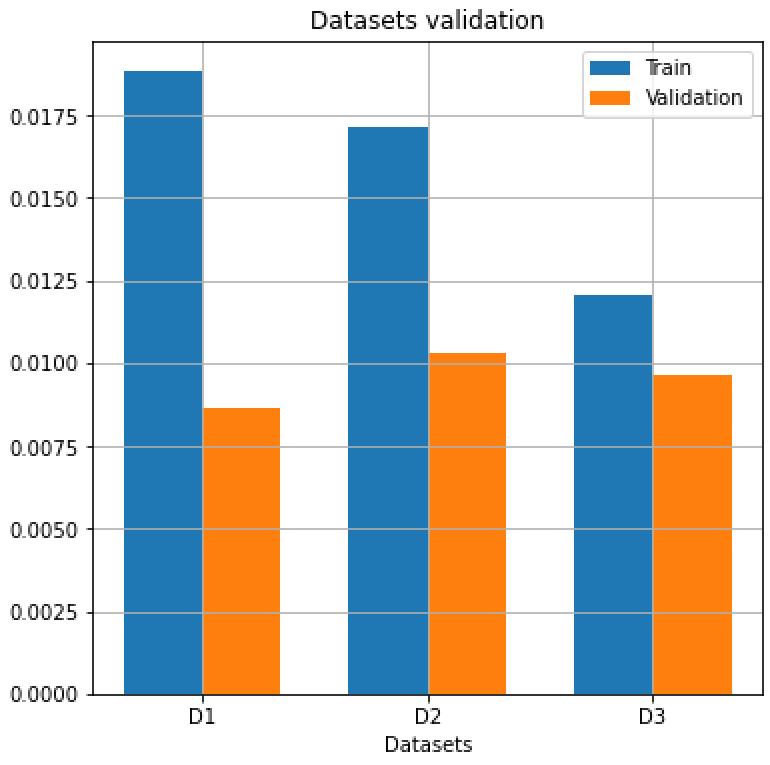
```
>>> datasets_train_val_score
[[0.01882686 0.00863946]
 [0.01713774 0.01030532]
 [0.012072   0.00963146]]

>>> datasets_best_model
[(1, 'PR', {'degree': 2}), (1, 'PR', {'degree': 1}), (1, 'PR',
{'degree': 1})]

>>> best_index
0

>>> test_score
0.011503918056149346
```

verbose = True

```
>>> from sklearn.neighbors import KNeighborsRegressor

>>> dataset_list = [(X1,y1),(X2,y2),(X3,y3)]

>>> model_paramGrid_list = [
                         ("kNN", KNeighborsRegressor(),
                             {"n_neighbors":[1,2,3],
                              "weights":["uniform","distance"]}
                          ),
                         ("PR", ms.PolynomialRegression(),
                             {"degree":[1,2,3]}),
]

>>> (datasets_train_val_score, datasets_best_model,
 best_index, test_score,
 axes) = ms.datasets_models_validation(dataset_list,
model_paramGrid_list, time_series=True, plot=True,
plot_train=True,  xvalues=["D1","D2","D3"], verbose=True)
```


Dataset 0 : models validation
Train
Validation
0.0175
0.0150
0.0125
0.0100
0.0075
0.0050
0.0025
0.0000
kNN
PR
Models

Dataset 1 : models validation
Train
Validation
0.016
0.014
0.012
0.010
0.008
0.006
0.004
0.002
0.000
kNN
PR
Models


Dataset 2 : models validation
Train
Validation
0.030
0.025
0.020
0.015
0.010
0.005
0.000
kNN
PR
Models


Datasets validation
Train
Validation
0.0175
0.0150
0.0125
0.0100
0.0075
0.0050
0.0025
0.0000
D1
D2
D3
Datasets