Welcome to Wine Project's documentation!

For this project, we participate in a competition on Kaggle: https://www.kaggle.com/datasets/uciml/red-wine-quality-cortez-et-al-2009.

This competition aims at a classification task: try to understand whether a wine is good or not from chemical wine characteristics.

For the aim, three modules are built: "main," which contains the start of the project; "preprocessing," in which the dataset is prepared for the classification part; "analysis," which includes algorithms used for prediction. Every subsection is explained in them.

Contents:

Indices and tables

Contacts

s.aranciofebbo@studenti.unipi.it
m.secondo@studenti.unipi.it

Footnotes

Main module

This module contains the main function.

main.classification(*X_train*, *X_test*, *y_train*, *y_test*)

This function execute various learning models used for the classification task. Then take the results of each algorithm and call comparison().

Parameters::
- **X_train** (*non-target train*,) –
- **X_test** (*non-target for prediction test*,) –
- **y_train** (*target for prediction train*,) –
- **y_test** (*target for prediction test*) –

main.comparison(*models*, *scores*)

Parameters::
- **models** (*classification model analyzed*,) –
- **scores** (*It is a list that memorizes [f1score, recall, precision] for each model in the sequence obtaine using models listed in "models."*) –

Return type:: None

main.feature_Importance(*df*)

main.visualization(*df*)

Footnotes

preprocessing module

This module implements many functions to transform datasets with preprocessing tasks.

preprocessing.preprocessing(*db*)

In this function are done the significant tasks of preprocessing step are. Are defined the targets of interest. The dataset is divided into a training set and test set(20%), normalized with MinMaxScaler(). The training set is balanced with SMOTE(random_state=42) algorithm.

Other internal functions are dedicated to plotting some helpful representations of the dataset. If the users want them, they must write "plt. show()".

Parameters:: **db** (*DataFrame the competition DataFrame*) –

Yields::
- **df** (*DataFrame structured according to our needs*,
- **X_train** (*non-target train*,)
- **X_test** (*target for prediction test*,)
- **y_train** (*non-target for prediction train*,)
- **y_test** (*target for prediction test*,)
- **X** (*attributes*)

preprocessing.remove_outlier(*db*)

This function aims to remove all outliers using the Box Plot Diagram. The purpose of this diagram is to identify outliers and discard it from the data series.

Parameters:: **db** (*DataFrame*) – the competition DataFrame

Yields:: **df** (*Database cleaned of outliers*)

Footnotes

analysis module

This module contains all algorithms used for the classification task.

analysis.DecisionTC(*X_train*, *X_test*, *y_train*, *y_test*)

This function is used to visualize the results of the grid searches for the Decision Tree algorithm.

Parameters::
- **X_train** (*non-target train*,) –
- **X_test** (*non-target for prediction test*,) –
- **y_train** (*target for prediction train*,) –
- **y_test** (*target for prediction test*) –

Yields:: **score** (*it is the result of the csv_algo(model, grid, y_test, y_pred) function. A list with []*) – macro_f1, macro_precision, macro_recall] computed from report().

analysis.KNN_CV(*X_train*, *X_test*, *y_train*, *y_test*)

This function is used to visualize the results of the grid searches for the k nearest neighbor algorithm. We used GridSearchCV to test Hyper-parameters with cross-validation algorithms kFold.

In kFold, the data is shuffled once at the start, and then divided into the number of desired splits.

The only Hyper-parameters are used are the n_neighbors parameters which contains the interval from 2 to k neighbors, and they test among all these values which one gives the best results

Parameters::
- **X_train** (*non-target train*,) –
- **X_test** (*non-target for prediction test*,) –
- **y_train** (*target for prediction train*,) –
- **y_test** (*target for prediction test*) –

Yields::

score (*it is the result of the csv_algo(model, grid, y_test, y_pred) function. A list with []*) – macro_f1, macro_precision, macro_recall] computed from report().

analysis.NaiveBayes(*X_train*, *X_test*, *y_train*, *y_test*)

This function is used to visualize the results of the grid searches for the Naive Bayes algorithm. As hyper-parameters, we used var_smoothing.

It is a stability calculation to widen (or smooth) the curve and therefore account for more samples that are further away from the distribution mean.

Parameters::
- **X_train** (*non-target train,*) –
- **X_test** (*target for prediction test,*) –
- **y_train** (*target for prediction train,*) –
- **y_test** (*target for prediction test*) –

Yields:: **score** (*it is the result of the csv_algo(model, grid, y_test, y_pred) function. A list with []*) – macro_f1, macro_precision, macro_recall] computed from report().

analysis.RandomForestClassifier_CV(*X_train*, *X_test*, *y_train*, *y_test*)

This function is used to visualize the results of the grid searches for the random forest algorithm.

Parameters::
- **X_train** (*non-target train,*) –
- **X_test** (*non-target for prediction test,*) –
- **y_train** (*target for prediction train,*) –
- **y_test** (*target for prediction test*) –

Yields:: **score** (*it is the result of the csv_algo(model, grid, y_test, y_pred) function. A list with [macro_f1, macro_precision, macro_recall] computed from report()*)

analysis.csv_algo(*model*, *grid*, *y_test*, *y_pred*)

This function prints the algorithm’s results in a CSV file.

Parameters::
- **model** (*classification model analyzed*,) –
- **grid** (*results of GridsearchCV from the model*,) –
- **y_test** (*target for prediction train*,) –
- **y_pred** (*the target used for prediction*) –

Yields:: **Scores** (*It is a list that contains the f1 average, precision average, and recall average.*)

analysis.logisticRegression(*X_train*, *X_test*, *y_train*, *y_test*)

This function is used to visualize the results of the grid searches for the Logistic Regression algorithm. As hyper-parameter we used C .It's a penalty term, meant to disincentivize and regulate against Overfitting.

Parameters::
- **X_train** (*non-target train*,) –
- **X_test** (*target for prediction test*,) –
- **y_train** (*non-target for prediction train*,) –
- **y_test** (*target for prediction test*,) –

Returns:: **score** – macro_f1, macro_precision, macro_recall] computed from report().

Return type:: it is the result of the csv_algo(model, grid, y_test, y_pred) function. A list with [

Footnotes

Python Module Index

a | m | p

a

m

p

Index

M

N

P

R

V