src

utils

`src.utils.center_encode`(*data*, *num_variables*, *categ_variables*) [source]

Centring and normalizing the data. Transforming the categorical variables.

Args:

param data: The dataset.
type data: pandas.DataFrame
param num_variables: List of columns with numerical values.
type num_variables: list
param categ_variables: List of columns with categorical values.
type categ_variables: list

Returns:

param data_tr_table: A centered and encoded dataset.
type data_tr_table: pandas.DataFrame

`src.utils.clean_noisy_data`(*dataset*, *classes=2*) [source]

Clean the data with replacing the miss-filled values by the correct ones and defining the right types for the dataset variables.

Args:

param dataset: The dataset to clean.
type dataset: pandas.DataFrame

Returns:

param dataset: Cleaned dataset.
type dataset: pandas.DataFrame

`src.utils.detect_type`(*data*) [source]

Detecting the type of the variables numerical or categorical

Args:

param data: The dataset.
type data: pandas.DataFrame

Returns:

param num_variables: List of columns with numerical values.
type num_variables: list
param categ_variables: List of columns with categorical values.
type categ_variables: list

`src.utils.determine_combinations`(*parameters*) [source]

Determine all possible combinaisons to have from a defined set of parameters of a model

Args:

param parameters: Dictionary of parameters.
type parameters: dict

Returns:

param comb_parameters: List of all combinaisons, each combinaison of parameters defined as a dictionary.
type comb_parameters: list

`src.utils.feature_selection`(*dataset*, *cut_off_variance=0.95*) [source]

Applies Feature selection using PCA. We fix the number of the remaining vectors based on the number of components which garantees 95% of the original variance of the dataset.

Args:

param dataset: The dataset.
type dataset: pandas.DataFrame
param cut_off_variance: The threshold ratio of the explained variance, takes values between 0 and 1, default is 0.95.
type cut_off_variance: float

Returns:

param data_compressed: The compressed dataset.
type data_compressed: pandas.DataFrame

`src.utils.`**`replace_missing`**(*data*, *num_variables*, *categ_variables*, *num_strategy='mean'*, *categ_strategy='most_frequent'*) [source]

Replacing the missing values in categorical variables and numerical variables by 2 corresponding strategies (mean for numerical variables and the most frequent value for categorical varibles for example)

Args:

param data: The dataset.
type data: pandas.DataFrame
param num_variables: List of columns with numerical values.
type num_variables: list
param categ_variables: List of columns with categorical values.
type categ_variables: list
param num_strategy: The defined strategy to replace the numerical missing values, default is mean.
type num_strategy: str
param categ_strategy: The defined strategy to replace the categorical missing values, default is most_frequent.
type categ_strategy: str

Returns:

param data_tr_table: A transformed dataset with missing values filled.
type data_tr_table: pandas.DataFrame

`src.utils.`**`select_params`**(*model*, *parameters*, *X*, *y*) [source]

Selecting the best parameters to take for the model

Input : Model, dictionary of possible parameters, Data, class Output :

Args:

param model: the defined model.
type model: object
param parameters: all possible parameters of the model for the gridsearch, format is a dictionary of lists.

Example : *'parameters':{'loss':['log', 'squared_hinge', 'perceptron']}}*

type parameters: dict
param X: features.
type X: numpy.array
param y: labels.
type y: numpy.array

Returns:

param chosen_params: Chosen combinaison of parameters, score of the cross validation with this combinaison.
type chosen_params: dict
param score_max: score of the chosen parameters, which is the highest score.
type score_max: float

`src.utils.`**`split_data`**(*X*, *y*) [source]

Split the data into training set and validation set

Args:

param X: The features
type X: numpy.array
param y: labels
type y: numpy.array

Returns: Training set and validation set of the data, training set and validation set of the class

`src.utils.`**`test_evaluate`**(*model*, *X*, *y*) [source]

Evaluate the model on test data and returns the f1 score.

`src.utils.`**`training`**(*model*, *parameters*, *X*, *y*) [source]

Train the model with defined parameters and returns the cross validation score

Input : Model, parameters of the model, data, class Output : the score of the cross validation

Args:

param model: the defined model.
type model: object
param parameters: parameters of the model.
type parameters: dict
param X: features.
type X: numpy.array
param y: labels.
type y: numpy.array

Returns:

param score: the score of the cross validation.
type score: float
param clf: the trained model.
type clf: object