multi-imbalance

Release 0.0.4

Damian Horna, Jacek Grycza, Hanna Klimczak, Kamil Pluciński

Jun 22, 2022

DOCSTRING

Multi-class imbalance is a common problem occurring in real-world supervised classifications tasks. While there has already been some research on the specialized methods aiming to tackle that challenging problem, most of them still lack coherent Python implementation that is simple, intuitive and easy to use. multi-imbalance is a python package tackling the problem of multi-class imbalanced datasets in machine learning.

CHAPTER

ONE

INSTALLATION:

```
pip install multi-imbalance
```

CHAPTER

TWO

EXAMPLE OF CODE:

```
from multi_imbalance.resampling.mdo import MDO

# Mahalanbois Distance Oversampling
mdo = MDO(k=9, k1_frac=0, seed=0)

# read the data
X_train, y_train, X_test, y_test = ...

# preprocess
X_train_resampled, y_train_resampled = mdo.fit_transform(np.copy(X_train), np.copy(y_
↪train))

# train the classifier on preprocessed data
clf_tree = DecisionTreeClassifier(random_state=0)
clf_tree.fit(X_train_resampled, y_train_resampled)

# make predictions
y_pred = clf_tree.predict(X_test)
```

2.1 multi_imbalance

2.1.1 multi_imbalance package

2.1.1.1 Subpackages

multi_imbalance.datasets package

Subpackages

multi_imbalance.datasets.tests package

Submodules

multi_imbalance.datasets.tests.test_data_loader module

Test the datasets loader.

multi_imbalance.datasets.tests.test_data_loader.**test_load_datasets**()

Module contents

Module contents

multi_imbalance.datasets.**load_datasets**(*data_home='./../../data/'*)
Load the benchmark datasets.

Parameters **data_home** – Default catalogue in which the data is stored in .tar.gz format.

Returns

OrderedDict of Bunch object. Each Bunch object refered as dataset have the following attributes:

- **dataset.data :** ndarray, shape (n_samples, n_features)
- **dataset.target :** ndarray, shape (n_samples,)
- **dataset.DESCR :** string Description of the each dataset.

multi_imbalance.ensemble package

Subpackages

multi_imbalance.ensemble.tests package

Submodules

multi_imbalance.ensemble.tests.test_ecoc module

multi_imbalance.ensemble.tests.test_ecoc.**test_dense_and_sparse_with_not_cached_matrices**(*enc*

multi_imbalance.ensemble.tests.test_ecoc.**test_ecoc_with_sklearn_pipeline**(*encoding_strategy*, *over-sam-pling*)

multi_imbalance.ensemble.tests.test_ecoc.**test_encoding**(*encoding_strategy*, *over-sampling*)

multi_imbalance.ensemble.tests.test_ecoc.**test_hamming_distance**()

multi_imbalance.ensemble.tests.test_ecoc.**test_no_oversampling**()

multi_imbalance.ensemble.tests.test_ecoc.**test_own_classifier_without_predict_and_fit**()

multi_imbalance.ensemble.tests.test_ecoc.**test_own_preprocessing_without_fit_transform**()

multi_imbalance.ensemble.tests.test_ecoc.**test_predefined_classifiers_and_weighting_without_**

multi_imbalance.ensemble.tests.test_ecoc.**test_random_oversampling**()

multi_imbalance.ensemble.tests.test_ecoc.**test_unknown_classifier**()

multi_imbalance.ensemble.tests.test_ecoc.**test_unknown_preprocessing**()

multi_imbalance.ensemble.tests.test_ecoc.**test_with_own_classifier**()

multi_imbalance.ensemble.tests.test_ecoc.**test_with_own_preprocessing**()

multi_imbalance.ensemble.tests.test_mrbbagging module

class multi_imbalance.ensemble.tests.test_mrbbagging.**TestMRBBagging**(*methodName='runTest'*)
Bases: unittest.case.TestCase

Create an instance of the class that will use the named test method when executed. Raises a ValueError if the instance does not have a method with the specified name.

test__group_data()

test__group_data_with_none()

test_api()

test_api_multiple_trees()

test_api_with_feature_selection()

test_api_with_feature_selection_sqrt_features()

test_api_with_random_feature_selection()

test_fit_with_invalid_classifier()

test_fit_with_invalid_labels()

test_with_invalid_k()

multi_imbalance.ensemble.tests.test_ovo module

multi_imbalance.ensemble.tests.test_ovo.**test_binary_classifiers**(*classifier*)

multi_imbalance.ensemble.tests.test_ovo.**test_ecoc_with_sklearn_pipeline**(*preprocessing_btwn*, *classifier*, *preprocessing*)

multi_imbalance.ensemble.tests.test_ovo.**test_fit_predict**()

multi_imbalance.ensemble.tests.test_ovo.**test_max_voting**()

multi_imbalance.ensemble.tests.test_ovo.**test_own_preprocessing_without_fit_resample**()

multi_imbalance.ensemble.tests.test_ovo.**test_predefined_classifiers_and_preprocessings_wit**

multi_imbalance.ensemble.tests.test_ovo.**test_unknown_preprocessing**()

multi_imbalance.ensemble.tests.test_ovo.**test_unknown_preprocessing_between_strategy_raises**

multi_imbalance.ensemble.tests.test_ovo.**test_with_own_classifier**()

multi_imbalance.ensemble.tests.test_ovo.**test_with_own_preprocessing**()

multi_imbalance.ensemble.tests.test_soupbagging module

multi_imbalance.ensemble.tests.test_soupbagging.**test_default_classifier**()

multi_imbalance.ensemble.tests.test_soupbagging.**test_exception**()

multi_imbalance.ensemble.tests.test_soupbagging.**test_fit_classifier_classifier**()

multi_imbalance.ensemble.tests.test_soupbagging.**test_soubagging**()

Module contents

Submodules

multi_imbalance.ensemble.ecoc module

class multi_imbalance.ensemble.ecoc.**ECOC**(*binary_classifier='KNN'*, *preprocessing='SOUP'*, *encoding='OVO'*, *n_neighbors=3*, *weights=None*)

Bases: sklearn.ensemble._bagging.BaggingClassifier

ECOC (Error Correcting Output Codes) is ensemble method for multi-class classification problems. Each class is encoded with unique binary or ternary code (where 0 means that class is excluded from training set of binary classifier). Then in the learning phase each binary classifier is learned. In the decoding phase the class which is closest to test instance in the sense of Hamming distance is chosen.

Parameters

- **binary_classifier** – binary classifier used by the algorithm. Possible classifiers:
  - **'tree':** Decision Tree Classifier,
  - **'NB':** Naive Bayes Classifier,
  - **'KNN' :** K-Nearest Neighbors
  - **'ClassifierMixin' :** An instance of a class that implements ClassifierMixin
- **preprocessing** – method for oversampling between aggregated classes in each dichotomy. Possible methods:
  - **None :** no oversampling applied,
  - **'globalCS' :** random oversampling - randomly chosen instances of minority classes are duplicated
  - **'SMOTE' :** Synthetic Minority Oversampling Technique
  - **'SOUP' :** Similarity Oversampling Undersampling Preprocessing
  - **'TransformerMixin' :** An instance of a class that implements TransformerMixin
- **encoding** – algorithm for encoding classes. Possible encodings:
  - **'dense':** ceil(10log2(num_of_classes)) dichotomies, -1 and 1 with probability 0.5 each
  - **'sparse' :** ceil(10log2(num_of_classes)) dichotomies, 0 with probability 0.5, -1 and 1 with probability 0.25 each

- **'OVO' :** 'one vs one' - n(n-1)/2 dichotomies, where n is number of classes, one for each pair of classes. Each column has one 1 and one -1 for classes included in particular pair, 0s for remaining classes.
- **'OVA' :** 'one vs all' - number of dichotomies is equal to number of classes. Each column has one 1 and -1 for all remaining rows
- **'complete'** [2^(n-1)-1 dichotomies, reference] T. G. Dietterich and G. Bakiri. Solving multiclass learning problems via error-correcting output codes. Journal of Artificial Intelligence Research, 2:263–286, 1995.

- **n_neighbors** –
- **weights** – strategy for dichotomies weighting. Possible values:
  - **None :** no weighting applied
  - **'acc' :** accuracy-based weights
  - **'avg_tpr_min' :** weights based on average true positive rates of dichotomies

fit(*X*, *y*, *minority_classes=None*)

Parameters

- **X** – two dimensional numpy array (number of samples x number of features) with float numbers
- **y** – one dimensional numpy array with labels for rows in X
- **minority_classes** – list of classes considered to be minority classes

Returns self: object

predict(*X*)

Parameters **X** – two dimensional numpy array (number of samples x number of features) with float numbers

Returns numpy array, shape = [number of samples]. Predicted target values for X.

multi_imbalance.ensemble.mrbbagging module

class multi_imbalance.ensemble.mrbbagging.**MRBBagging**(*k*, *learning_algorithm*, *undersampling=True*, *feature_selection=False*, *random_fs=False*, *half_features=True*, *random_state=None*)

Bases: sklearn.ensemble._bagging.BaggingClassifier

Multi-class Roughly Balanced Bagging (MRBBagging) is a generalization of MRBBagging for adapting to multiple minority classes.

Reference: M. Lango, J. Stefanowski: Multi-class and feature selection extensions of RoughlyBalanced Bagging for imbalanced data. J. Intell Inf Syst (2018) 50: 97

Parameters

- **k** – number of classifiers (multiplied by 3 when choosing feature selection)
- **learning_algorithm** – classifier to be used

- **undersampling** – (optional) boolean value to determine if undersampling or oversampling should be performed
- **feature_selection** – (optional) boolean value to determine if feature selection should be performed
- **random_fs** – (optional) boolean value to determine if feature selection should be all random (if False, chi^2, F test and random feature selection are performed)
- **half_features** – (optional) boolean value to determine if the number of features to be selected should be 50% (if False, it is set to the square root of the base number of features)
- **random_state** – (optional) the seed of the pseudo random number generator

fit (*x*, *y*, ***kwargs*)
Build a MRBBagging ensemble of estimators from the training data.

Parameters

- **x** – Two dimensional numpy array (number of samples x number of features) with float numbers.
- **y** – One dimensional numpy array with labels for rows in X.

Returns self (object)

predict (*data*)
Predict classes for examples in data.

Parameters **data** – Two dimensional numpy array (number of samples x number of features) with float numbers.

multi_imbalance.ensemble.ovo module

class multi_imbalance.ensemble.ovo.**OVO**(*binary_classifier='tree'*, *n_neighbors=3*, *preprocessing='SOUP'*, *preprocessing_between='all'*)
Bases: sklearn.ensemble._bagging.BaggingClassifier

OVO (One vs One) is an ensemble method that makes predictions for multi-class problems. OVO decomposes problem into m(m-1)/2 binary problems, where m is number of classes. Each of binary classifiers distinguishes between two classes. In the learning phase each classifier is learned only with instances from particular two classes. In prediction phase each classifier decides between these two classes. Results are aggregated and final output is derived depending on chosen aggregation model.

Parameters

- **binary_classifier** – binary classifier. Possible classifiers:
  - **'tree':** Decision Tree Classifier,
  - **'KNN':** K-Nearest Neighbors
  - **'NB' :** Naive Bayes
  - **'ClassifierMixin' :** An instance of a class that implements ClassifierMixin
- **n_neighbors** – number of nearest neighbors in KNN, works only if binary_classifier=='KNN'
- **preprocessing** – method for preprocessing of pairs of classes in the learning phase of ensemble. Possible values:
  - **None:** no preprocessing applied

- **'globalCS':** oversampling with globalCS algorithm
- **'SMOTE':** oversampling with SMOTE algorithm
- **'SOUP':** oversampling and undersampling with SOUP algorithm
- **'TransformerMixin' :** An instance of a class that implements TransformerMixin

• **preprocessing_between** – types of classes between which resampling should be applied. Possible values:

- **'all' :** oversampling between each pair of classes
- **'maj-min' :** oversampling only between majority ad minority classes

fit (*X*, *y*, *minority_classes=None*)

Parameters

• **X** – two dimensional numpy array (number of samples x number of features) with float numbers

• **y** – one dimensional numpy array with labels for rows in X

• **minority_classes** – list of classes considered to be minority

Returns self: object

predict (*X*)

Parameters **X** – two dimensional numpy array (number of samples x number of features) with float numbers

Returns numpy array, shape = [number of samples]. Predicted target values for X.

should_perform_oversampling (*first_class*, *second_class*)

multi_imbalance.ensemble.soup_bagging module

class multi_imbalance.ensemble.soup_bagging.**SOUPBagging** (*classifier=None*, *maj_int_min=None*, *n_classifiers=5*)

Bases: sklearn.ensemble._bagging.BaggingClassifier

Version of Bagging that applies SOUP in each classifier

Reference: Lango, M., and Stefanowski, J. SOUP-Bagging: a new approach for multi-class imbalanced data classification. PP-RAI '19: Polskie Porozumienie na Rzecz Sztucznej Inteligencji (2019).

Parameters

• **classifier** – Instance of classifier

• **maj_int_min** – dict {'maj': majority class labels, 'min': minority class labels}

• **n_classifiers** – number of classifiers

fit (*X*, *y*, ***kwargs*)

Parameters

• **X** – array-like, sparse matrix of shape = [n_samples, n_features] The training input samples.

• **y** – array-like, shape = [n_samples]. The target values (class labels).

• ****kwargs** – dict (optional)

Returns self object

static fit_classifier(*args*)

predict(*X*, *strategy: str = 'average'*)

Predict class for X. The predicted class of an input sample is computed as the class with the highest sum of predicted probability.

Parameters

- **X** – {array-like, sparse matrix} of shape = [n_samples, n_features]. The training input samples.
- **strategy** –
  - **'average' :** takes max from average values in prediction
  - **'optimistic' :** takes always best value of probability
  - **'pessimistic' :** takes always the worst value of probability
  - **'mixed' :** for minority classes takes optimistic strategy, and pessimistic for others. It requires maj_int_min

Returns array of shape = [n_samples]. The predicted classes.

predict_proba(*X*)

Predict class probabilities for X.

Parameters **X** – {array-like, sparse matrix} of shape = [n_samples, n_features]. The training input samples.

Returns array of shape = [n_classifiers, n_samples, n_classes]. The class probabilities of the input samples.

multi_imbalance.ensemble.soup_bagging.**fit_clf**(*args*)

Module contents

multi_imbalance.resampling package

Subpackages

multi_imbalance.resampling.tests package

Submodules

multi_imbalance.resampling.tests.test_globalcs module

multi_imbalance.resampling.tests.test_globalcs.**calc_duplicates_quantities**(*X*, *y*, *X_oversampled*)

multi_imbalance.resampling.tests.test_globalcs.**get_goal_quantity**(*y*)

multi_imbalance.resampling.tests.test_globalcs.**global_cs_mock**()

multi_imbalance.resampling.tests.test_globalcs.**test_output_equal_replication**(*X*, *y*, *global_cs_mock*)

multi_imbalance.resampling.tests.test_globalcs.**test_output_length_validate**(*X*, *y*, *global_cs_mock*)

multi_imbalance.resampling.tests.test_mdo module

multi_imbalance.resampling.tests.test_mdo.**mdo_mock**()

multi_imbalance.resampling.tests.test_mdo.**test_choose_samples**(*X*, *y*, *sc_minor_expected*, *weights_expected*, *mdo_mock*)

multi_imbalance.resampling.tests.test_mdo.**test_choose_samples_when_correct**(*mdo_mock*)

multi_imbalance.resampling.tests.test_mdo.**test_choose_samples_when_zero_samples_expected**(*m*

multi_imbalance.resampling.tests.test_mdo.**test_mdo_api**(*mdo_mock*)

multi_imbalance.resampling.tests.test_mdo.**test_zero_variance**(*mdo_mock*)

multi_imbalance.resampling.tests.test_soup module

multi_imbalance.resampling.tests.test_soup.**soup_mock**()

multi_imbalance.resampling.tests.test_soup.**test_calculating_safe_levels_for_class**(*X*, *y*, *zero_safe_le*
one_safe_lev
*first_sample_*
soup_mock)

multi_imbalance.resampling.tests.test_soup.**test_calculating_safe_levels_for_sample**(*X*, *y*, *zero_safe_*
one_safe_l
first_sampl
soup_mock

multi_imbalance.resampling.tests.test_soup.**test_oversample**(*X*, *y*, *class_name*, *expected_undersampling*, *expected_oversampling*, *soup_mock*)

multi_imbalance.resampling.tests.test_soup.**test_undersample**(*X*, *y*, *class_name*, *expected_undersampling*, *expected_oversampling*, *soup_mock*)

multi_imbalance.resampling.tests.test_spider module

multi_imbalance.resampling.tests.test_spider.**test_estimate_cost_matrix**()

multi_imbalance.resampling.tests.test_spider.**test_fit_resample**()

multi_imbalance.resampling.tests.test_spider.**test_intersect**()

multi_imbalance.resampling.tests.test_spider.**test_knn**()

multi_imbalance.resampling.tests.test_spider.**test_min_cost_classes**()

multi_imbalance.resampling.tests.test_spider.**test_setdiff**()

multi_imbalance.resampling.tests.test_spider.**test_union**()

multi_imbalance.resampling.tests.test_static_smote module

multi_imbalance.resampling.tests.test_static_smote.**test_static_smote**()

Module contents

Submodules

multi_imbalance.resampling.global_cs module

class multi_imbalance.resampling.global_cs.**GlobalCS**(*shuffle: bool = True*)

Bases: imblearn.base.BaseSampler

Global CS is an algorithm that equalizes number of samples in each class. It duplicates all samples equally for each class to achieve majority class size

multi_imbalance.resampling.mdo module

class multi_imbalance.resampling.mdo.**MDO**(*k=5*, *k1_frac=0.4*, *seed=0*, *prop=1*, *maj_int_min=None*)

Bases: imblearn.base.BaseSampler

Mahalanbois Distance Oversampling is an algorithm that oversamples all classes to a quantity of the major class. Samples for oversampling are chosen based on their k neighbours and new samples are created in random place but with the same Mahalanbois distance from the centre of class to chosen sample.

Parameters

- **k** – Number of neighbours considered during the neighbourhood analysis
- **k1_frac** – Ratio of the number of neighbours in the sample class to all neighbours in the neighbourhood. If the ratio is greater, the example will not be considered noise
- **seed** –
- **prop** – Oversampling ratio, if equal to one the class size after resampling will be equal to the size of the largest class
- **maj_int_min** – dict {'maj': majority class labels, 'min': minority class labels}

calculate_same_class_neighbour_quantities(*S_minor*, *S_minor_label*)

multi_imbalance.resampling.soup module

class multi_imbalance.resampling.soup.**SOUP**(*k: int = 7*, *shuffle=False*, *maj_int_min=None*)
Bases: imblearn.base.BaseSampler

Similarity Oversampling and Undersampling Preprocessing (SOUP) is an algorithm that equalizes number of samples in each class. It also takes care of the similarity between classes, which means that it removes samples from majority class, that are close to samples from the other class and duplicate samples from the minority classes, which are in the safest area in space

Parameters

- **k** – number of neighbors
- **shuffle** – bool - output will be shuffled
- **maj_int_min** – dict {'maj': majority class labels, 'min': minority class labels}

multi_imbalance.resampling.spider module

class multi_imbalance.resampling.spider.**SPIDER3**(*k*, *maj_int_min=None*, *cost=None*)
Bases: imblearn.base.BaseSampler

SPIDER3 algorithm implementation for selective preprocessing of multi-class imbalanced data sets.

Reference: Wojciechowski, S., Wilk, S., Stefanowski, J.: An Algorithm for Selective Preprocessing of Multi-class Imbalanced Data. Proceedings of the 10th International Conference on Computer Recognition Systems CORES 2017

Parameters

- **k** – Number of nearest neighbors considered while resampling.
- **maj_int_min** – Dict that contains lists of majority, intermediate and minority classes labels.
- **cost** – The cost matrix. An element c[i, j] of this matrix represents the cost associated with misclassifying an example from class i as class one from class j.

amplify(*int_min_class*)

clean(*int_min_class*)

relabel(*int_min_class*)

multi_imbalance.resampling.static_smote module

class multi_imbalance.resampling.static_smote.**StaticSMOTE**
Bases: imblearn.base.BaseSampler

Static SMOTE implementation:

Reference: Fernández-Navarro, F., Hervás-Martínez, C., Gutiérrez, P.A.: A dynamic over-sampling procedure based on sensitivity for multi-class problems. Pattern Recognit. 44, 1821–1833 (2011)

Module contents

multi_imbalance.utils package

Submodules

multi_imbalance.utils.array_util module

multi_imbalance.utils.array_util.**contains**(*dataset*, *example*)
Returns if dataset contains the example. :param dataset: :param example: :return: True or False depending on whether dataset contains the example.

multi_imbalance.utils.array_util.**index_of**(*arr*, *example*)

Returns Index of learning exmaple in arr.

multi_imbalance.utils.array_util.**intersect**(*arr1*, *arr2*)
Performs the intersection operation over two numpy arrays (not removing duplicates).

Parameters

- **arr1** – Numpy array number 1.
- **arr2** – Numpy array number 2.

Returns The intersection of arr1 and arr2.

multi_imbalance.utils.array_util.**setdiff**(*arr1*, *arr2*)
Performs the difference over two numpy arrays.

Parameters

- **arr1** – Numpy array number 1.
- **arr2** – Numpy array number 2.

Returns Result of the difference of arr1 and arr2.

multi_imbalance.utils.array_util.**union**(*arr1*, *arr2*)
Performs the union over two numpy arrays (not removing duplicates, as it's how the algorithm SPIDER3 actually works).

Parameters

- **arr1** – Numpy array number 1.
- **arr2** – Numpy array number 2.

Returns The union of arr1 and arr2.

multi_imbalance.utils.data module

multi_imbalance.utils.data.**construct_flat_2pc_df**(*X*, *y*) → pandas.core.frame.DataFrame
This function takes two dimensional X and one dimensional y arrays, concatenates and returns them as data frame

Parameters

- **X** – two dimensional numpy array
- **y** – one dimensional numpy array with labels

Returns Data frame with 3 columns x1 x2 and y and with number of rows equal to number of rows in X

multi_imbalance.utils.data.**construct_maj_int_min**(*y: numpy.ndarray*, *strategy='median'*) → collections.OrderedDict

This function creates dictionary with information which classes are minority or majority

Parameters

- **y** – One dimensional numpy array that contains class labels
- **strategy** – The principle according to which the division into minority and majority classes will be determined:
  - **'median':** A class whose size is equal to the median of the class sizes will be considered "intermediate"
  - **'average':** The average class size will be calculated, all classes that are smaller will be considered as minority and the rest will be considered majority

Returns dictionary with keys 'maj', 'int', 'min. The value for each key is a list containing the class labels belonging to the given group

multi_imbalance.utils.data.**get_project_root**() → pathlib.Path

Returns project root folder.

multi_imbalance.utils.data.**load_arff_dataset**(*path: str*, *one_hot_encode: bool = True*, *return_non_cat_length: bool = False*)

Load and return the dataset saved in arff type file

Parameters

- **path** (*str*) – location of dataset file
- **one_hot_encode** (*bool*) – flag, if true encodes categorical variables using OneHotEncoder
- **return_non_cat_length** (*bool*) – flag, if true returns the number of non categorical variables

Returns

- ndarray X - dimensional numpy array where non categorical variables are stored in first columns followed by categorical variables
- ndarray y - one dimensional numpy array with the classification target
- bool non_cat_length - number of non categorical variables (only if return_non_cat_length=True)

multi_imbalance.utils.data.**load_datasets_arff**(*return_non_cat_length=False*, *dataset_paths=None*)

multi_imbalance.utils.metrics module

multi_imbalance.utils.metrics.**gmean_score**(*y_test*, *y_pred*, *correction: float = 0.001*) → float

Calculate geometric mean score

Parameters

- **y_test** – numpy array with labels
- **y_pred** – numpy array with predicted labels
- **correction** – value that replaces 0 during multiplication to avoid zeroing the result

Returns geometric_mean_score: float

multi_imbalance.utils.min_int_maj module

multi_imbalance.utils.plot module

multi_imbalance.utils.plot.**plot_cardinality_and_2d_data**(*X*, *y*, *dataset_name=''*) → None

Plots cardinality of classes from y as well as scatter plot of X transformed to two dimensions using PCA

Parameters

- **X** (*ndarray*) – two dimensional numpy array
- **y** (*ndarray*) – one dimensional numpy array
- **dataset_name** (*str*) – title of chart

multi_imbalance.utils.plot.**plot_visual_comparision_datasets**(*X1*, *y1*, *X2*, *y2*, *dataset_name1=''*, *dataset_name2=''*) → None

Plots comparision of X1 y1 and X2 y2 using plot_cardinality_and_2d_data, which plots cardinality of classes from y as well as scatter plot of X transformed to two dimensions using PCA

Parameters

- **X1** (*ndarray*) – two dimensional numpy array with data from dataset1
- **y1** (*ndarray*) – one dimensional numpy array with target classes from dataset1
- **X2** (*ndarray*) – two dimensional numpy array with data from dataset2
- **y2** (*ndarray*) – one dimensional numpy array with target classes from dataset1
- **dataset_name1** (*str*) – first dataset chart title
- **dataset_name2** (*str*) – second dataset chart title

Module contents

2.1.1.2 Module contents

2.2 License

2.3 Contact

Question? Please contact test@gmail.com

PYTHON MODULE INDEX

m

INDEX

O

P

R

S

T

U