combo Documentation

Release 0.1.3

Yue Zhao

Jan 14, 2023

GETTING STARTED

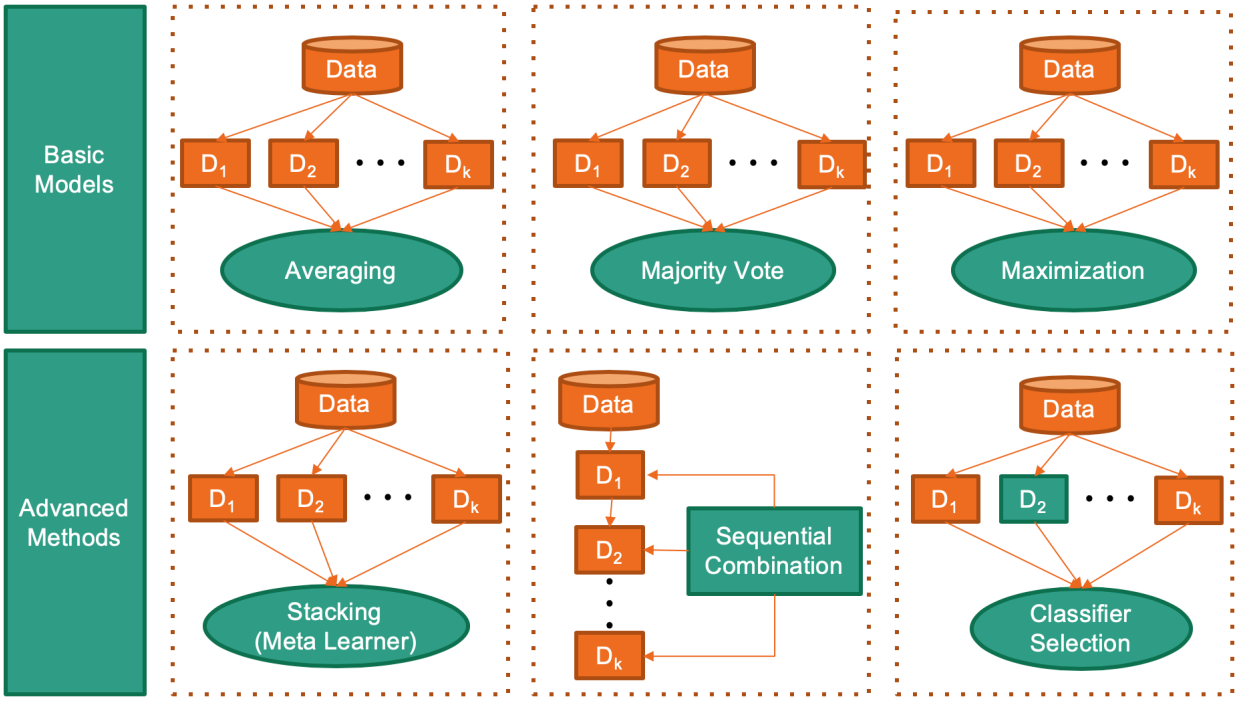
Deployment & Documentation & Stats

Build Status & Coverage & Maintainability & License

---

combo is a comprehensive Python toolbox for **combining machine learning (ML) models and scores**. **Model combination** can be considered as a subtask of ensemble learning, and has been widely used in real-world tasks and data science competitions like Kaggle [ABK07]. **combo** has been used/introduced in various research works since its inception [ARPN20, AZNL19].

combo library supports the combination of models and score from key ML libraries such as scikit-learn, xgboost, and LightGBM, for crucial tasks including classification, clustering, anomaly detection. See figure below for some representative combination approaches.



combo is featured for:

- **Unified APIs, detailed documentation, and interactive examples** across various algorithms.
- **Advanced and latest models**, such as Stacking/DCS/DES/EAC/LSCP.
- **Comprehensive coverage** for classification, clustering, anomaly detection, and raw score.
- **Optimized performance with JIT and parallelization** when possible, using numba and joblib.

API Demo:

```
from combo.models.classifier_stacking import Stacking
# initialize a group of base classifiers
classifiers = [DecisionTreeClassifier(), LogisticRegression(),
               KNeighborsClassifier(), RandomForestClassifier(),
               GradientBoostingClassifier()]

clf = Stacking(base_estimators=classifiers) # initialize a Stacking model
clf.fit(X_train, y_train) # fit the model

# predict on unseen data
y_test_labels = clf.predict(X_test)  # label prediction
y_test_proba = clf.predict_proba(X_test)  # probability prediction
```

Citing combo:

combo paper is published in AAAI 2020 (demo track). If you use combo in a scientific publication, we would appreciate citations to the following paper:

```
@inproceedings{zhao2020combo,
  title={Combining Machine Learning Models and Scores using combo library},
  author={Zhao, Yue and Wang, Xuejian and Cheng, Cheng and Ding, Xueying},
  booktitle={Thirty-Fourth AAAI Conference on Artificial Intelligence},
```

(continues on next page)

(continued from previous page)

```
  month = {Feb},
  year={2020},
  address = {New York, USA}
}
```

or:

```
Zhao, Y., Wang, X., Cheng, C. and Ding, X., 2020. Combining Machine Learning Models and␣
↪Scores using combo library. Thirty-Fourth AAAI Conference on Artificial Intelligence.
```

Key Links and Resources:

- awesome-ensemble-learning (ensemble learning related books, papers, and more)
- View the latest codes on Github
- View the documentation & API
- View all examples
- View the demo video for AAAI 2020
- Execute Interactive Jupyter Notebooks

CHAPTER

ONE

API CHEATSHEET & REFERENCE

Full API Reference: (https://pycombo.readthedocs.io/en/latest/api.html). The following APIs are applicable for most models for easy use.

- *combo.models.base.BaseAggregator.fit()*: Fit estimator. y is optional for unsupervised methods.
- *combo.models.base.BaseAggregator.predict()*: Predict on a particular sample once the estimator is fitted.
- *combo.models.base.BaseAggregator.predict_proba()*: Predict the probability of a sample belonging to each class once the estimator is fitted.
- *combo.models.base.BaseAggregator.fit_predict()*: Fit estimator and predict on X. y is optional for unsupervised methods.

For raw score combination (after the score matrix is generated), use individual methods from "score_comb.py" directly. Raw score combination API: (https://pycombo.readthedocs.io/en/latest/api.html#score-combination).

CHAPTER

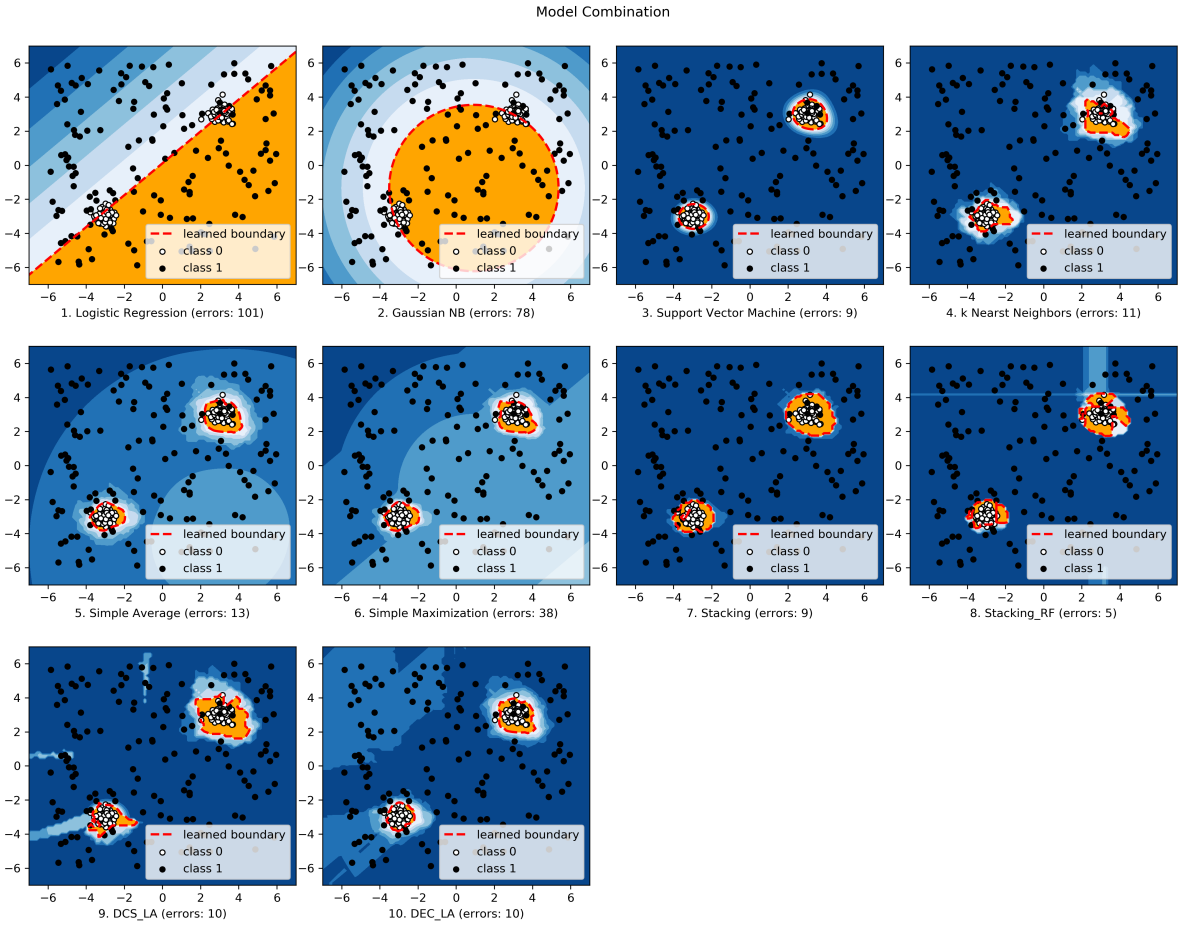
TWO

IMPLEMENTED ALGORITHMS

combo groups combination frameworks by tasks. General purpose methods are fundamental ones which can be applied to various tasks.

Class/Function	Task	Algorithm	Year	Ref
combo.models.score_comb.average	General Purpose	Average & Weighted Average: average across all scores/prediction results, maybe with weights	N/A	[AZho12]
combo.models.score_comb.maximization	General Purpose	Maximization: simple combination by taking the maximum scores	N/A	[AZho12]
combo.models.score_comb.median	General Purpose	Median: take the median value across all scores/prediction results	N/A	[AZho12]
combo.models.score_comb.majority_vote	General Purpose	Majority Vote & Weighted Majority Vote	N/A	[AZho12]
combo.models.classifier_comb.SimpleClassifierAggregator	Classification	SimpleClassifierAggregator: combining classifiers by general purpose methods above	N/A	N/A
combo.models.classifier_dcs.DCS_LA	Classification	DCS: Dynamic Classifier Selection (Combination of multiple classifiers using local accuracy estimates)	1997	[AWKB97]
combo.models.classifier_des.DES_LA	Classification	DES: Dynamic Ensemble Selection (From dynamic classifier selection to dynamic ensemble selection)	2008	[AKSBJ08]
combo.models.classifier_stacking.Stacking	Classification	Stacking (meta ensembling): use a meta learner to learn the base classifier results	N/A	[AGor16]
combo.models.cluster_comb.ClustererEnsemble	Clustering	Clusterer Ensemble: combine the results of multiple clustering results by relabeling	2006	[AZT06]
combo.models.cluster_eac.EAC	Clustering	Combining multiple clusterings using evidence accumulation (EAC)	2002	[AFJ05]
combo.models.detector_comb.SimpleDetectorAggregator	Anomaly Detection	SimpleDetectorCombination: combining outlier detectors by general purpose methods above	N/A	[AAS17]
combo.models.score_comb.aom	Anomaly Detection	Average of Maximum (AOM): divide base detectors into subgroups to take the maximum, and then average	2015	[AAS15]
combo.models.score_comb.moa	Anomaly Detection	Maximum of Average (MOA): divide base detectors into subgroups to take the average, and then maximize	2015	[AAS15]
combo.models.detector_xgbod.XGBOD	Anomaly Detection	XGBOD: a semi-supervised combination framework for outlier detection	2018	[AZH18]
combo.models.detector_lscp.LSCP	Anomaly Detection	Locally Selective Combination (LSCP)	2019	[AZNHL19]

The comparison among selected implemented models is made available below (Figure, compare_selected_classifiers.py, Interactive Jupyter Notebooks). For Jupyter Notebooks, please navigate to **"/notebooks/compare_selected_classifiers.ipynb"**.


Model Combination
learned boundary
class 0
class 1
1. Logistic Regression (errors: 101)
2. Gaussian NB (errors: 78)
3. Support Vector Machine (errors: 9)
4. k Nearst Neighbors (errors: 11)
5. Simple Average (errors: 13)
6. Simple Maximization (errors: 38)
7. Stacking (errors: 9)
8. Stacking_RF (errors: 5)
9. DCS_LA (errors: 10)
10. DEC_LA (errors: 10)

CHAPTER

THREE

DEVELOPMENT STATUS

combo is currently **under development** as of Feb, 2020. A concrete plan has been laid out and will be implemented in the next few months.

Similar to other libraries built by us, e.g., Python Outlier Detection Toolbox (pyod), **combo** is also targeted to be published in *Journal of Machine Learning Research (JMLR)*, open-source software track. A demo paper has been presented in *AAAI 2020* for progress update.

Watch & Star to get the latest update! Also feel free to send me an email (zhaoy@cmu.edu) for suggestions and ideas.

3.1 Installation

It is recommended to use **pip** for installation. Please make sure **the latest version** is installed, as combo is updated frequently:

```
pip install combo            # normal install
pip install --upgrade combo  # or update if needed
pip install --pre combo      # or include pre-release version for new features
```

Alternatively, you could clone and run setup.py file:

```
git clone https://github.com/yzhao062/combo.git
cd combo
pip install .
```

Required Dependencies:

- Python 3.5, 3.6, or 3.7
- joblib
- matplotlib (**optional for running examples**)
- numpy>=1.13
- numba>=0.35
- pyod
- scipy>=0.19.1
- scikit_learn>=0.20

Note on Python 2: The maintenance of Python 2.7 will be stopped by January 1, 2020 (see official announcement). To be consistent with the Python change and combo's dependent libraries, e.g., scikit-learn, **combo only supports Python 3.5+** and we encourage you to use Python 3.5 or newer for the latest functions and bug fixes. More information can be found at Moving to require Python 3.

3.2 Examples by Tasks

All implemented modes are associated with examples, check "combo examples" for more information.

3.2.1 Example of Stacking/DCS/DES

"examples/classifier_stacking_example.py" demonstrates the basic API of stacking (meta ensembling). "examples/classifier_dcs_la_example.py" demonstrates the basic API of Dynamic Classifier Selection by Local Accuracy. "examples/classifier_des_la_example.py" demonstrates the basic API of Dynamic Ensemble Selection by Local Accuracy.

It is noted **the basic API is consistent across all these models**.

1. Initialize a group of classifiers as base estimators

```
# initialize a group of classifiers
classifiers = [DecisionTreeClassifier(random_state=random_state),
               LogisticRegression(random_state=random_state),
               KNeighborsClassifier(),
               RandomForestClassifier(random_state=random_state),
               GradientBoostingClassifier(random_state=random_state)]
```

2. Initialize, fit, predict, and evaluate with Stacking

```
from combo.models.classifier_stacking import Stacking

clf = Stacking(base_estimators=classifiers, n_folds=4, shuffle_data=False,
               keep_original=True, use_proba=False, random_state=random_state)

clf.fit(X_train, y_train)
y_test_predict = clf.predict(X_test)
evaluate_print('Stacking | ', y_test, y_test_predict)
```

3. See a sample output of classifier_stacking_example.py

```
Decision Tree         | Accuracy:0.9386, ROC:0.9383, F1:0.9521
Logistic Regression   | Accuracy:0.9649, ROC:0.9615, F1:0.973
K Neighbors           | Accuracy:0.9561, ROC:0.9519, F1:0.9662
Gradient Boosting     | Accuracy:0.9605, ROC:0.9524, F1:0.9699
Random Forest         | Accuracy:0.9605, ROC:0.961, F1:0.9693

Stacking              | Accuracy:0.9868, ROC:0.9841, F1:0.9899
```

3.2.2 Example of Classifier Combination

"examples/classifier_comb_example.py" demonstrates the basic API of predicting with multiple classifiers. **It is noted that the API across all other algorithms are consistent/similar**.

1. Initialize a group of classifiers as base estimators

```
# initialize a group of classifiers
classifiers = [DecisionTreeClassifier(random_state=random_state),
               LogisticRegression(random_state=random_state),
               KNeighborsClassifier(),
               RandomForestClassifier(random_state=random_state),
               GradientBoostingClassifier(random_state=random_state)]
```

2. Initialize, fit, predict, and evaluate with a simple aggregator (average)

```
from combo.models.classifier_comb import SimpleClassifierAggregator

clf = SimpleClassifierAggregator(classifiers, method='average')
clf.fit(X_train, y_train)
y_test_predicted = clf.predict(X_test)
evaluate_print('Combination by avg   |', y_test, y_test_predicted)
```

3. See a sample output of classifier_comb_example.py

```
Decision Tree        | Accuracy:0.9386, ROC:0.9383, F1:0.9521
Logistic Regression  | Accuracy:0.9649, ROC:0.9615, F1:0.973
K Neighbors          | Accuracy:0.9561, ROC:0.9519, F1:0.9662
Gradient Boosting    | Accuracy:0.9605, ROC:0.9524, F1:0.9699
Random Forest        | Accuracy:0.9605, ROC:0.961, F1:0.9693

Combination by avg   | Accuracy:0.9693, ROC:0.9677, F1:0.9763
Combination by w_avg | Accuracy:0.9781, ROC:0.9716, F1:0.9833
Combination by max   | Accuracy:0.9518, ROC:0.9312, F1:0.9642
Combination by w_vote| Accuracy:0.9649, ROC:0.9644, F1:0.9728
Combination by median| Accuracy:0.9693, ROC:0.9677, F1:0.9763
```

3.2.3 Example of Clustering Combination

"examples/cluster_comb_example.py" demonstrates the basic API of combining multiple base clustering estimators. "examples/cluster_eac_example.py" demonstrates the basic API of Combining multiple clusterings using evidence accumulation (EAC).

1. Initialize a group of clustering methods as base estimators

```
# Initialize a set of estimators
estimators = [KMeans(n_clusters=n_clusters),
              MiniBatchKMeans(n_clusters=n_clusters),
              AgglomerativeClustering(n_clusters=n_clusters)]
```

2. Initialize a Clusterer Ensemble class and fit the model

```
from combo.models.cluster_comb import ClustererEnsemble
# combine by Clusterer Ensemble
clf = ClustererEnsemble(estimators, n_clusters=n_clusters)
clf.fit(X)
```

3. Get the aligned results

```
# generate the labels on X
aligned_labels = clf.aligned_labels_
predicted_labels = clf.labels_
```

3.2.4 Example of Outlier Detector Combination

"examples/detector_comb_example.py" demonstrates the basic API of combining multiple base outlier detectors.

1. Initialize a group of outlier detection methods as base estimators

```
# Initialize a set of estimators
detectors = [KNN(), LOF(), OCSVM()]
```

2. Initialize a simple averaging aggregator, fit the model, and make the prediction.

```
from combo.models.detector combination import SimpleDetectorAggregator
clf = SimpleDetectorAggregator(base_estimators=detectors)
clf_name = 'Aggregation by Averaging'
clf.fit(X_train)

y_train_pred = clf.labels_  # binary labels (0: inliers, 1: outliers)
y_train_scores = clf.decision_scores_  # raw outlier scores

# get the prediction on the test data
y_test_pred = clf.predict(X_test)  # outlier labels (0 or 1)
y_test_scores = clf.decision_function(X_test)  # outlier scores
```

3. Evaluate the prediction using ROC and Precision @ Rank n.

```
# evaluate and print the results
print("\nOn Training Data:")
evaluate_print(clf_name, y_train, y_train_scores)
print("\nOn Test Data:")
evaluate_print(clf_name, y_test, y_test_scores)
```

4. See sample outputs on both training and test data.

```
On Training Data:
Aggregation by Averaging ROC:0.9994, precision @ rank n:0.95

On Test Data:
Aggregation by Averaging ROC:1.0, precision @ rank n:1.0
```

3.3 API CheatSheet

Full API Reference: (https://pycombo.readthedocs.io/en/latest/api.html). The following APIs are consistent for most of the models (API Cheatsheet: https://pycombo.readthedocs.io/en/latest/api_cc.html).

- *combo.models.base.BaseAggregator.fit()*: Fit estimator. y is optional for unsupervised methods.
- *combo.models.base.BaseAggregator.predict()*: Predict on a particular sample once the estimator is fitted.
- *combo.models.base.BaseAggregator.predict_proba()*: Predict the probability of a sample belonging to each class once the estimator is fitted.
- *combo.models.base.BaseAggregator.fit_predict()*: Fit estimator and predict on X. y is optional for unsupervised methods.

Helpful functions:

- *combo.models.base.BaseAggregator.get_params()*: Get the parameters of the model.
- *combo.models.base.BaseAggregator.set_params()*: Set the parameters of the model.
- Each base estimator can be accessed by calling clf[i] where i is the estimator index.

For raw score combination (after the score matrix is generated), use individual methods from "score_comb.py" directly. Raw score combination API: (https://pycombo.readthedocs.io/en/latest/api.html#score-combination).

See base class definition below:

3.3.1 combo.models.base module

Base class for core models

class combo.models.base.**BaseAggregator**(*base_estimators*, *pre_fitted=False*)

Bases: ABC

Abstract class for all combination classes.

Parameters

- **base_estimators** (*list, length must be greater than 1*) – A list of base estimators. Certain methods must be present, e.g., *fit* and *predict*.
- **pre_fitted** (*bool, optional (default=False)*) – Whether the base estimators are trained. If True, *fit* process may be skipped.

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

Fit estimator. y is optional for unsupervised methods.

Parameters

- **X** (*numpy array of shape (n_samples, n_features)*) – The input samples.
- **y** (*numpy array of shape (n_samples,), optional (default=None)*) – The ground truth of the input samples (labels).

Return type
self

abstract fit_predict(*X*, *y=None*)

Fit estimator and predict on X. y is optional for unsupervised methods.

Parameters

- **X** (*numpy array of shape (n_samples, n_features)*) – The input samples.
- **y** (*numpy array of shape (n_samples,), optional (default=None)*) – The ground truth of the input samples (labels).

Returns
labels – Class labels for each data sample.

Return type
numpy array of shape (n_samples,)

get_params(*deep=True*)

Get parameters for this estimator.

See http://scikit-learn.org/stable/modules/generated/sklearn.base.BaseEstimator.html and sklearn/base.py for more information.

Parameters
deep (*boolean, optional*) – If True, will return the parameters for this estimator and contained subobjects that are estimators.

Returns
params – Parameter names mapped to their values.

Return type
mapping of string to any

abstract predict(*X*)

Predict the class labels for the provided data.

Parameters
X (*numpy array of shape (n_samples, n_features)*) – The input samples.

Returns
labels – Class labels for each data sample.

Return type
numpy array of shape (n_samples,)

abstract predict_proba(*X*)

Return probability estimates for the test data X.

Parameters
X (*numpy array of shape (n_samples, n_features)*) – The input samples.

Returns
p – The class probabilities of the input samples. Classes are ordered by lexicographic order.

Return type
numpy array of shape (n_samples,)

set_params(***params*)

Set the parameters of this estimator. The method works on simple estimators as well as on nested objects (such as pipelines). The latter have parameters of the form <component>__<parameter> so that it's possible to update each component of a nested object.

See http://scikit-learn.org/stable/modules/generated/sklearn.base.BaseEstimator.html and sklearn/base.py for more information.

Returns
self

Return type
object

3.4 API Reference

3.4.1 Classifier Combination

- *combo.models.classifier_comb.SimpleClassifierAggregator*: a collection of classifier combination methods, e.g., average, median, and majority vote.
- *combo.models.classifier_stacking.Stacking*: Stacking (meta ensembling). Check this introductory article by Kaggle.
- *combo.models.classifier_dcs.DCS_LA*: Dynamic classifier selection (DCS) by local accuracy.
- *combo.models.classifier_des.DES_LA*: Dynamic ensemble selection (DES) by local accuracy.

3.4.2 Cluster Combination

- *combo.models.cluster_comb.ClustererEnsemble*: Clusterer Ensemble combines multiple base clustering estimators by alignment.
- *combo.models.cluster_comb.clusterer_ensemble_scores()*: Clusterer Ensemble on clustering results directly.
- *combo.models.cluster_eac.EAC*: Combining multiple clusterings using evidence accumulation (EAC).

3.4.3 Outlier Detector Combination

- *combo.models.detector_comb.SimpleDetectorAggregator*: a collection of outlier detector combination methods, e.g., average, median, and maximization. Refer PyOD for more information.
- *combo.models.detector_lscp.LSCP*: Locally Selective Combination of Parallel Outlier Ensembles (LSCP).

3.4.4 Score Combination

combo.models.score_comb: a collection of (raw) score combination methods.

- *combo.models.score_comb.average()*
- *combo.models.score_comb.maximization()*
- *combo.models.score_comb.median()*
- *combo.models.score_comb.majority_vote()*
- *combo.models.score_comb.aom()*
- *combo.models.score_comb.moa()*

3.4.5 All Models

Core Models

combo.models.classifier_comb module

A collection of methods for combining classifiers

class combo.models.classifier_comb.**SimpleClassifierAggregator**(*base_estimators*, *method='average'*, *threshold=0.5*, *weights=None*, *pre_fitted=False*)

Bases: *BaseAggregator*

A collection of simple classifier combination methods.

Parameters

- **base_estimators** (*list or numpy array (n_estimators,)*) – A list of base classifiers.
- **method** (*str, optional (default='average')*) – Combination method: {'average', 'maximization', 'majority vote', 'median'}. Pass in weights of classifier for weighted version.
- **threshold** (*float in (0, 1), optional (default=0.5)*) – Cut-off value to convert scores into binary labels.
- **weights** (*numpy array of shape (1, n_classifiers)*) – Classifier weights.
- **pre_fitted** (*bool, optional (default=False)*) – Whether the base classifiers are trained. If True, *fit* process may be skipped.

fit(*X*, *y*)

Fit classifier.

Parameters

- **X** (*numpy array of shape (n_samples, n_features)*) – The input samples.
- **y** (*numpy array of shape (n_samples,), optional (default=None)*) – The ground truth of the input samples (labels).

fit_predict(*X*, *y*)

Fit estimator and predict on X

Parameters

- **X** (*numpy array of shape (n_samples, n_features)*) – The input samples.
- **y** (*numpy array of shape (n_samples,), optional (default=None)*) – The ground truth of the input samples (labels).

Returns

labels – Class labels for each data sample.

Return type

numpy array of shape (n_samples,)

get_params(*deep=True*)

Get parameters for this estimator.

See http://scikit-learn.org/stable/modules/generated/sklearn.base.BaseEstimator.html and sklearn/base.py for more information.

Parameters
deep (*boolean, optional*) – If True, will return the parameters for this estimator and contained subobjects that are estimators.

Returns
params – Parameter names mapped to their values.

Return type
mapping of string to any

`predict`(*X*)

Predict the class labels for the provided data.

Parameters
X (*numpy array of shape (n_samples, n_features)*) – The input samples.

Returns
labels – Class labels for each data sample.

Return type
numpy array of shape (n_samples,)

`predict_proba`(*X*)

Return probability estimates for the test data X.

Parameters
X (*numpy array of shape (n_samples, n_features)*) – The input samples.

Returns
p – The class probabilities of the input samples. Classes are ordered by lexicographic order.

Return type
numpy array of shape (n_samples,)

`set_params`(***params*)

Set the parameters of this estimator. The method works on simple estimators as well as on nested objects (such as pipelines). The latter have parameters of the form `<component>__<parameter>` so that it's possible to update each component of a nested object.

See http://scikit-learn.org/stable/modules/generated/sklearn.base.BaseEstimator.html and sklearn/base.py for more information.

Returns
self

Return type
object

combo.models.classifier_dcs module

Stacking (meta ensembling). See http://blog.kaggle.com/2016/12/27/a-kagglers-guide-to-model-stacking-in-practice/ for more information.

class combo.models.classifier_dcs.**DCS_LA**(*base_estimators*, *local_region_size=30*, *threshold=None*, *pre_fitted=None*)

Bases: *BaseAggregator*

Dynamic Classifier Selection (DCS) is an established combination framework for classification tasks. The technique was first proposed by Ho et al. in 1994 [BHHS94] and then extended, under the name DCS Local Accuracy, by Woods et al. in 1997 [BWKB97] to select the most accurate base classifier in a local region. The motivation

behind this approach is that base classifiers often make distinctive errors and over a degree of complementarity. Consequently, selectively combining base classifier can result in a performance improvement over generic ensembles which use the majority vote of all base classifiers.

See [BWKB97] for details.

Parameters

- **base_estimators** (*list or numpy array (n_estimators,)*) – A list of base classifiers.
- **local_region_size** (*int, optional (default=30)*) – Number of training points to consider in each iteration of the local region generation process (30 by default).
- **threshold** (*float in (0, 1), optional (default=None)*) – Cut-off value to convert scores into binary labels.
- **pre_fitted** (*bool, optional (default=False)*) – Whether the base classifiers are trained. If True, *fit* process may be skipped.

fit(*X*, *y*)

Fit classifier.

Parameters

- **X** (*numpy array of shape (n_samples, n_features)*) – The input samples.
- **y** (*numpy array of shape (n_samples,), optional (default=None)*) – The ground truth of the input samples (labels).

fit_predict(*X*, *y*)

Fit estimator and predict on X

Parameters

- **X** (*numpy array of shape (n_samples, n_features)*) – The input samples.
- **y** (*numpy array of shape (n_samples,), optional (default=None)*) – The ground truth of the input samples (labels).

Returns

labels – Class labels for each data sample.

Return type

numpy array of shape (n_samples,)

get_params(*deep=True*)

Get parameters for this estimator.

See http://scikit-learn.org/stable/modules/generated/sklearn.base.BaseEstimator.html and sklearn/base.py for more information.

Parameters

deep (*boolean, optional*) – If True, will return the parameters for this estimator and contained subobjects that are estimators.

Returns

params – Parameter names mapped to their values.

Return type

mapping of string to any

predict(*X*)

Predict the class labels for the provided data.

Parameters
X (*numpy array of shape (n_samples, n_features)*) – The input samples.

Returns
labels – Class labels for each data sample.

Return type
numpy array of shape (n_samples,)

predict_proba(*X*)

Return probability estimates for the test data X.

Parameters
X (*numpy array of shape (n_samples, n_features)*) – The input samples.

Returns
p – The class probabilities of the input samples. Classes are ordered by lexicographic order.

Return type
numpy array of shape (n_samples,)

set_params(***params*)

Set the parameters of this estimator. The method works on simple estimators as well as on nested objects (such as pipelines). The latter have parameters of the form <component>__<parameter> so that it's possible to update each component of a nested object.

See http://scikit-learn.org/stable/modules/generated/sklearn.base.BaseEstimator.html and sklearn/base.py for more information.

Returns
self

Return type
object

combo.models.classifier_des module

Dynamic Classifier Selection (DES) is an established combination framework for classification tasks.

class combo.models.classifier_des.**DES_LA**(*base_estimators*, *local_region_size=30*, *n_selected_clfs=None*, *use_weights=False*, *threshold=None*, *pre_fitted=None*)

Bases: *BaseAggregator*

Dynamic Ensemble Selection (DES) is an established combination framework for classification tasks. The technique was based on Dynamic Classifier Selection (DCS) proposed by Ho et al. in 1994 [BHHS94]. The motivation behind this approach is that base classifiers often make distinctive errors and over a degree of complementarity. Consequently, selectively combining base classifier can result in a performance improvement over generic ensembles which use the majority vote of all base classifiers.

Compared with DCS, DES uses a group of best classifiers to conduct a second phase combination, other than only the best classifier. The implemented version in this class is DES_LA which uses local accuracy as the metric for evaluating base classifier performance. *predict* uses (weighted) majority vote and *predict_proba* uses (weighted) average.

See [BKSBJ08] for details.

Parameters

- **base_estimators** (*list or numpy array (n_estimators,)*) – A list of base classifiers.
- **local_region_size** (*int, optional (default=30)*) – Number of training points to consider in each iteration of the local region generation process (30 by default).
- **n_selected_clfs** (*int, optional (default=None)*) – Number of selected base classifiers in the second phase combination. If None, set it to 1/2 * n_base_estimators
- **use_weights** (*bool, optional (default=False)*) – If True, use the classifiers' performance on the local region as their weight.
- **threshold** (*float in (0, 1), optional (default=None)*) – Cut-off value to convert scores into binary labels.
- **pre_fitted** (*bool, optional (default=False)*) – Whether the base classifiers are trained. If True, *fit* process may be skipped.

fit(*X*, *y*)

Fit classifier.

Parameters

- **X** (*numpy array of shape (n_samples, n_features)*) – The input samples.
- **y** (*numpy array of shape (n_samples,), optional (default=None)*) – The ground truth of the input samples (labels).

fit_predict(*X*, *y*)

Fit estimator and predict on X

Parameters

- **X** (*numpy array of shape (n_samples, n_features)*) – The input samples.
- **y** (*numpy array of shape (n_samples,), optional (default=None)*) – The ground truth of the input samples (labels).

Returns

labels – Class labels for each data sample.

Return type

numpy array of shape (n_samples,)

get_params(*deep=True*)

Get parameters for this estimator.

See http://scikit-learn.org/stable/modules/generated/sklearn.base.BaseEstimator.html and sklearn/base.py for more information.

Parameters

deep (*boolean, optional*) – If True, will return the parameters for this estimator and contained subobjects that are estimators.

Returns

params – Parameter names mapped to their values.

Return type

mapping of string to any

predict(*X*)

Predict the class labels for the provided data.

Parameters
X (*numpy array of shape (n_samples, n_features)*) – The input samples.

Returns
labels – Class labels for each data sample.

Return type
numpy array of shape (n_samples,)

`predict_proba`(*X*)

Return probability estimates for the test data X.

Parameters
X (*numpy array of shape (n_samples, n_features)*) – The input samples.

Returns
p – The class probabilities of the input samples. Classes are ordered by lexicographic order.

Return type
numpy array of shape (n_samples,)

`set_params`(***params*)

Set the parameters of this estimator. The method works on simple estimators as well as on nested objects (such as pipelines). The latter have parameters of the form `<component>__<parameter>` so that it's possible to update each component of a nested object.

See http://scikit-learn.org/stable/modules/generated/sklearn.base.BaseEstimator.html and sklearn/base.py for more information.

Returns
self

Return type
object

combo.models.classifier_stacking module

Stacking (meta ensembling). See http://blog.kaggle.com/2016/12/27/a-kagglers-guide-to-model-stacking-in-practice/ for more information.

`class combo.models.classifier_stacking.Stacking`(*base_estimators*, *meta_clf=None*, *n_folds=2*, *keep_original=True*, *use_proba=False*, *shuffle_data=False*, *random_state=None*, *threshold=None*, *pre_fitted=None*)

Bases: *BaseAggregator*

Meta ensembling, also known as stacking. See http://blog.kaggle.com/2016/12/27/a-kagglers-guide-to-model-stacking-in-practice/ for more information

Parameters

- **base_estimators** (*list or numpy array (n_estimators,)*) – A list of base classifiers.
- **meta_clf** (*object, optional (default=LogisticRegression)*) – The meta classifier to make the final prediction.
- **n_folds** (*int, optional (default=2)*) – The number of splits of the training sample.
- **keep_original** (*bool, optional (default=False)*) – If True, keep the original features for training and predicting.

- **use_proba** (*bool, optional (default=False)*) – If True, use the probability prediction as the new features.
- **shuffle_data** (*bool, optional (default=False)*) – If True, shuffle the input data.
- **random_state** (*int, RandomState or None, optional (default=None)*) – If int, random_state is the seed used by the random number generator; If RandomState instance, random_state is the random number generator; If None, the random number generator is the RandomState instance used by *np.random*.
- **threshold** (*float in (0, 1), optional (default=None)*) – Cut-off value to convert scores into binary labels.
- **pre_fitted** (*bool, optional (default=False)*) – Whether the base classifiers are trained. If True, *fit* process may be skipped.

fit(*X*, *y*)

Fit classifier.

Parameters

- **X** (*numpy array of shape (n_samples, n_features)*) – The input samples.
- **y** (*numpy array of shape (n_samples,), optional (default=None)*) – The ground truth of the input samples (labels).

fit_predict(*X*, *y*)

Fit estimator and predict on X

Parameters

- **X** (*numpy array of shape (n_samples, n_features)*) – The input samples.
- **y** (*numpy array of shape (n_samples,), optional (default=None)*) – The ground truth of the input samples (labels).

Returns
labels – Class labels for each data sample.

Return type
numpy array of shape (n_samples,)

get_params(*deep=True*)

Get parameters for this estimator.

See http://scikit-learn.org/stable/modules/generated/sklearn.base.BaseEstimator.html and sklearn/base.py for more information.

Parameters
deep (*boolean, optional*) – If True, will return the parameters for this estimator and contained subobjects that are estimators.

Returns
params – Parameter names mapped to their values.

Return type
mapping of string to any

predict(*X*)

Predict the class labels for the provided data.

Parameters
X (*numpy array of shape (n_samples, n_features)*) – The input samples.

Returns
labels – Class labels for each data sample.

Return type
numpy array of shape (n_samples,)

`predict_proba`(*X*)

Return probability estimates for the test data X.

Parameters
X (*numpy array of shape (n_samples, n_features)*) – The input samples.

Returns
p – The class probabilities of the input samples. Classes are ordered by lexicographic order.

Return type
numpy array of shape (n_samples,)

`set_params`(***params*)

Set the parameters of this estimator. The method works on simple estimators as well as on nested objects (such as pipelines). The latter have parameters of the form `<component>__<parameter>` so that it's possible to update each component of a nested object.

See http://scikit-learn.org/stable/modules/generated/sklearn.base.BaseEstimator.html and sklearn/base.py for more information.

Returns
self

Return type
object

`combo.models.classifier_stacking.`**`split_datasets`**(*X*, *y*, *n_folds=3*, *shuffle_data=False*, *random_state=None*)

Utility function to split the data for stacking. The data is split into n_folds with roughly equal rough size.

Parameters

- **X** (*numpy array of shape (n_samples, n_features)*) – The input samples.
- **y** (*numpy array of shape (n_samples,)*) – The ground truth of the input samples (labels).
- **n_folds** (*int, optional (default=3)*) – The number of splits of the training sample.
- **shuffle_data** (*bool, optional (default=False)*) – If True, shuffle the input data.
- **random_state** (*RandomState, optional (default=None)*) – A random number generator instance to define the state of the random permutations generator.

Returns

- **X** (*numpy array of shape (n_samples, n_features)*) – The input samples. If shuffle_data, return the shuffled data.
- **y** (*numpy array of shape (n_samples,)*) – The ground truth of the input samples (labels). If shuffle_data, return the shuffled data.
- **index_lists** (*list of list*) – The list of indexes of each fold regarding the returned X and y. For instance, index_lists[0] contains the indexes of fold 0.

combo.models.cluster_comb module

A collection of combination methods for clustering

class combo.models.cluster_comb.**ClustererEnsemble**(*base_estimators*, *n_clusters*, *weights=None*, *reference_idx=0*, *pre_fitted=False*)

Bases: *BaseAggregator*

Clusterer Ensemble combines multiple base clustering estimators by alignment. See [BZT06] for details.

Parameters

- **base_estimators** (*list or numpy array of shape (n_estimators,)*) – A list of base estimators. Estimators must have a *labels_* attribute once fitted. Sklearn clustering estimators are recommended.
- **n_clusters** (*int, optional (default=8)*) – The number of clusters.
- **weights** (*numpy array of shape (n_estimators,)*) – Estimator weights. May be used after the alignment.
- **reference_idx** (*int in range [0, n_estimators-1], optional (default=0)*) – The ith base estimator used as the reference for label alignment.
- **pre_fitted** (*bool, optional (default=False)*) – Whether the base estimators are trained. If True, *fit* process may be skipped.

labels_

The predicted label of the fitted data.

Type
int

fit(*X*)

Fit estimators.

Parameters
X (*numpy array of shape (n_samples, n_features)*) – The input samples.

fit_predict(*X*, *y=None*)

Fit estimator and predict on X. y is optional for unsupervised methods.

Parameters

- **X** (*numpy array of shape (n_samples, n_features)*) – The input samples.
- **y** (*numpy array of shape (n_samples,), optional (default=None)*) – The ground truth of the input samples (labels).

Returns
labels – Cluster labels for each data sample.

Return type
numpy array of shape (n_samples,)

get_params(*deep=True*)

Get parameters for this estimator.

See http://scikit-learn.org/stable/modules/generated/sklearn.base.BaseEstimator.html and sklearn/base.py for more information.

Parameters
deep (*boolean, optional*) – If True, will return the parameters for this estimator and contained subobjects that are estimators.

Returns
params – Parameter names mapped to their values.

Return type
mapping of string to any

predict(*X*)

Predict the class labels for the provided data.

Parameters
X (*numpy array of shape (n_samples, n_features)*) – The input samples.

Returns
labels – Class labels for each data sample.

Return type
numpy array of shape (n_samples,)

predict_proba(*X*)

Predict the class labels for the provided data.

Parameters
X (*numpy array of shape (n_samples, n_features)*) – The input samples.

Returns
labels – Class labels for each data sample.

Return type
numpy array of shape (n_samples,)

set_params(***params*)

Set the parameters of this estimator. The method works on simple estimators as well as on nested objects (such as pipelines). The latter have parameters of the form <component>__<parameter> so that it's possible to update each component of a nested object.

See http://scikit-learn.org/stable/modules/generated/sklearn.base.BaseEstimator.html and sklearn/base.py for more information.

Returns
self

Return type
object

combo.models.cluster_comb.**clusterer_ensemble_scores**(*original_labels*, *n_estimators*, *n_clusters*, *weights=None*, *return_results=False*, *reference_idx=0*)

Function to align the raw clustering results from base estimators. Different from ClustererEnsemble class, this function takes in the output from base estimators directly without training and prediction.

Parameters

- **original_labels** (*numpy array of shape (n_samples, n_estimators)*) – The raw output from base estimators
- **n_estimators** (*int*) – The number of base estimators.
- **n_clusters** (*int, optional (default=8)*) – The number of clusters.

- **weights** (*numpy array of shape (1, n_estimators)*) – Estimators weights.
- **return_results** (*bool, optional (default=False)*) – If True, also return the aligned label matrix.
- **reference_idx** (*int in range [0, n_estimators-1], optional (default=0)*) – The ith base estimator used as the reference for label alignment.

Returns
aligned_labels – The aligned label results by using reference_idx estimator as the reference.

Return type
numpy array of shape (n_samples, n_estimators)

combo.models.cluster_eac module

Combining multiple clusterings using evidence accumulation (EAC).

class combo.models.cluster_eac.**EAC**(*base_estimators*, *n_clusters*, *linkage_method='single'*, *weights=None*, *pre_fitted=False*)

Bases: *BaseAggregator*

Combining multiple clusterings using evidence accumulation (EAC) first builds similarity matrix for each base clustering to model the similarity among the cluster assignment among each sample. After the similarity matrices are aggregated, a hierarchical clustering is built on it. See [BFJ05] for details.

Parameters

- **base_estimators** (*list or numpy array of shape (n_estimators,)*) – A list of base estimators. Estimators must have a *labels_* attribute once fitted. Sklearn clustering estimators are recommended.
- **n_clusters** (*int, optional (default=8)*) – The number of clusters.
- **linkage_method** (*str, optional (default='single')*) – The linkage method to use (single, complete, average, weighted, median centroid, ward). See https://docs.scipy.org/doc/scipy/reference/cluster.hierarchy.html for more information.
- **weights** (*numpy array of shape (n_estimators,)*) – Estimator weights. May be used after the alignment.
- **pre_fitted** (*bool, optional (default=False)*) – Whether the base estimators are trained. If True, *fit* process may be skipped.

labels_

The predicted label of the fitted data.

Type
int

Z_

The linkage matrix encoding the hierarchical clustering. This can be used to plot dendrogram using scipy.

Type
numpy array

fit(*X*)

Fit estimators.

Parameters
X (*numpy array of shape (n_samples, n_features)*) – The input samples.

`fit_predict`(*X*, *y=None*)

Fit estimator and predict on X. y is optional for unsupervised methods.

Parameters

- **X** (*numpy array of shape (n_samples, n_features)*) – The input samples.
- **y** (*numpy array of shape (n_samples,), optional (default=None)*) – The ground truth of the input samples (labels).

Returns
labels – Cluster labels for each data sample.

Return type
numpy array of shape (n_samples,)

`get_params`(*deep=True*)

Get parameters for this estimator.

See http://scikit-learn.org/stable/modules/generated/sklearn.base.BaseEstimator.html and sklearn/base.py for more information.

Parameters
deep (*boolean, optional*) – If True, will return the parameters for this estimator and contained subobjects that are estimators.

Returns
params – Parameter names mapped to their values.

Return type
mapping of string to any

`predict`(*X*)

Predict the class labels for the provided data.

Parameters
X (*numpy array of shape (n_samples, n_features)*) – The input samples.

Returns
labels – Class labels for each data sample.

Return type
numpy array of shape (n_samples,)

`predict_proba`(*X*)

Predict the class labels for the provided data.

Parameters
X (*numpy array of shape (n_samples, n_features)*) – The input samples.

Returns
labels – Class labels for each data sample.

Return type
numpy array of shape (n_samples,)

`set_params`(***params*)

Set the parameters of this estimator. The method works on simple estimators as well as on nested objects (such as pipelines). The latter have parameters of the form `<component>__<parameter>` so that it's possible to update each component of a nested object.

See http://scikit-learn.org/stable/modules/generated/sklearn.base.BaseEstimator.html and sklearn/base.py for more information.

Returns
self

Return type
object

combo.models.detector_comb module

A collection of methods for combining detectors

class combo.models.detector_comb.**SimpleDetectorAggregator**(*base_estimators*, *method='average'*, *contamination=0.1*, *standardization=True*, *weights=None*, *pre_fitted=False*)

Bases: *BaseAggregator*

A collection of simple detector combination methods.

Parameters

- **base_estimators** (*list, length must be greater than 1*) – Base unsupervised outlier detectors from PyOD. (Note: requires fit and decision_function methods)
- **method** (*str, optional (default='average')*) – Combination method: {'average', 'maximization', 'median'}. Pass in weights of detector for weighted version.
- **contamination** (*float in (0., 0.5), optional (default=0.1)*) – The amount of contamination of the data set, i.e. the proportion of outliers in the data set. Used when fitting to define the threshold on the decision function.
- **standardization** (*bool, optional (default=True)*) – If True, perform standardization first to convert prediction score to zero mean and unit variance. See http://scikit-learn.org/stable/auto_examples/preprocessing/plot_scaling_importance.html
- **weights** (*numpy array of shape (1, n_detectors)*) – detector weights.
- **pre_fitted** (*bool, optional (default=False)*) – Whether the base detectors are trained. If True, *fit* process may be skipped.

decision_scores_

The outlier scores of the training data. The higher, the more abnormal. Outliers tend to have higher scores. This value is available once the detector is fitted.

Type
numpy array of shape (n_samples,)

threshold_

The threshold is based on `contamination`. It is the `n_samples * contamination` most abnormal samples in `decision_scores_`. The threshold is calculated for generating binary outlier labels.

Type
float

labels_

The binary labels of the training data. 0 stands for inliers and 1 for outliers/anomalies. It is generated by applying `threshold_` on `decision_scores_`.

Type
int, either 0 or 1

decision_function(*X*)

Predict raw anomaly scores of X using the fitted detector.

The anomaly score of an input sample is computed based on the fitted detector. For consistency, outliers are assigned with higher anomaly scores.

Parameters
X (*numpy array of shape (n_samples, n_features)*) – The input samples. Sparse matrices are accepted only if they are supported by the base estimator.

Returns
anomaly_scores – The anomaly score of the input samples.

Return type
numpy array of shape (n_samples,)

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

Fit detector. y is optional for unsupervised methods.

Parameters

- **X** (*numpy array of shape (n_samples, n_features)*) – The input samples.
- **y** (*numpy array of shape (n_samples,), optional (default=None)*) – The ground truth of the input samples (labels).

Returns
labels_ – Return the generated labels.

Return type
numpy array of shape (n_samples,)

fit_predict(*X*, *y=None*)

Fit estimator and predict on X. y is optional for unsupervised methods.

Parameters

- **X** (*numpy array of shape (n_samples, n_features)*) – The input samples.
- **y** (*numpy array of shape (n_samples,), optional (default=None)*) – The ground truth of the input samples (labels).

Returns
labels – Class labels for each data sample.

Return type
numpy array of shape (n_samples,)

get_params(*deep=True*)

Get parameters for this estimator.

See http://scikit-learn.org/stable/modules/generated/sklearn.base.BaseEstimator.html and sklearn/base.py for more information.

Parameters
deep (*boolean, optional*) – If True, will return the parameters for this estimator and contained subobjects that are estimators.

Returns
params – Parameter names mapped to their values.

Return type
mapping of string to any

predict(*X*)

Predict if a particular sample is an outlier or not.

Parameters
X (*numpy array of shape (n_samples, n_features)*) – The input samples.

Returns
outlier_labels – For each observation, tells whether or not it should be considered as an outlier according to the fitted model. 0 stands for inliers and 1 for outliers.

Return type
numpy array of shape (n_samples,)

predict_proba(*X*, *proba_method='linear'*)

Predict the probability of a sample being outlier. Two approaches are possible:

1. simply use Min-max conversion to linearly transform the outlier scores into the range of [0,1]. The model must be fitted first.
2. use unifying scores, see [BKKSZ11].

Parameters

- **X** (*numpy array of shape (n_samples, n_features)*) – The input samples.
- **proba_method** (*str, optional (default='linear')*) – Probability conversion method. It must be one of 'linear' or 'unify'.

Returns
outlier_labels – For each observation, tells whether or not it should be considered as an outlier according to the fitted model. Return the outlier probability, ranging in [0,1].

Return type
numpy array of shape (n_samples,)

set_params(***params*)

Set the parameters of this estimator. The method works on simple estimators as well as on nested objects (such as pipelines). The latter have parameters of the form <component>__<parameter> so that it's possible to update each component of a nested object.

See http://scikit-learn.org/stable/modules/generated/sklearn.base.BaseEstimator.html and sklearn/base.py for more information.

Returns
self

Return type
object

combo.models.detector_lscp module

Locally Selective Combination of Parallel Outlier Ensembles (LSCP). Implemented on PyOD library (https://github.com/yzhao062/pyod).

class combo.models.detector_lscp.**LSCP**(*base_estimators*, *local_region_size=30*, *local_max_features=1.0*, *n_bins=10*, *random_state=None*, *contamination=0.1*, *pre_fitted=False*)

Bases: *BaseAggregator*

Locally Selection Combination in Parallel Outlier Ensembles

LSCP is an unsupervised parallel outlier detection ensemble which selects competent detectors in the local region of a test instance. This implementation uses an Average of Maximum strategy. First, a heterogeneous list of base detectors is fit to the training data and then generates a pseudo ground truth for each train instance is generated by taking the maximum outlier score.

For each test instance: 1) The local region is defined to be the set of nearest training points in randomly sampled feature subspaces which occur more frequently than a defined threshold over multiple iterations.

2) Using the local region, a local pseudo ground truth is defined and the pearson correlation is calculated between each base detector's training outlier scores and the pseudo ground truth.

3) A histogram is built out of pearson correlation scores; detectors in the largest bin are selected as competent base detectors for the given test instance.

4) The average outlier score of the selected competent detectors is taken to be the final score.

See [BZNHL19] for details.

Parameters

- **base_estimators** (*list, length must be greater than 1*) – Base unsupervised outlier detectors from PyOD. (Note: requires fit and decision_function methods)
- **local_region_size** (*int, optional (default=30)*) – Number of training points to consider in each iteration of the local region generation process (30 by default).
- **local_max_features** (*float in (0.5, 1.), optional (default=1.0)*) – Maximum proportion of number of features to consider when defining the local region (1.0 by default).
- **n_bins** (*int, optional (default=10)*) – Number of bins to use when selecting the local region
- **random_state** (*RandomState, optional (default=None)*) – A random number generator instance to define the state of the random permutations generator.
- **contamination** (*float in (0., 0.5), optional (default=0.1)*) – The amount of contamination of the data set, i.e. the proportion of outliers in the data set. Used when fitting to define the threshold on the decision function (0.1 by default).
- **pre_fitted** (*bool, optional (default=False)*) – Whether the base estimators are trained. If True, *fit* process may be skipped.

decision_scores_

The outlier scores of the training data. The higher, the more abnormal. Outliers tend to have higher scores. This value is available once the detector is fitted.

Type
numpy array of shape (n_samples,)

threshold_

The threshold is based on `contamination`. It is the `n_samples * contamination` most abnormal samples in `decision_scores_`. The threshold is calculated for generating binary outlier labels.

Type
float

labels_

The binary labels of the training data. 0 stands for inliers and 1 for outliers/anomalies. It is generated by applying `threshold_` on `decision_scores_`.

Type
int, either 0 or 1

`decision_function`(*X*)

Predict raw anomaly scores of X using the fitted detector.

The anomaly score of an input sample is computed based on the fitted detector. For consistency, outliers are assigned with higher anomaly scores.

Parameters
X (*numpy array of shape (n_samples, n_features)*) – The input samples. Sparse matrices are accepted only if they are supported by the base estimator.

Returns
anomaly_scores – The anomaly score of the input samples.

Return type
numpy array of shape (n_samples,)

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

Fit detector. y is optional for unsupervised methods.

Parameters

- **X** (*numpy array of shape (n_samples, n_features)*) – The input samples.
- **y** (*numpy array of shape (n_samples,), optional (default=None)*) – The ground truth of the input samples (labels).

`fit_predict`(*X*, *y=None*)

Fit estimator and predict on X. y is optional for unsupervised methods.

Parameters

- **X** (*numpy array of shape (n_samples, n_features)*) – The input samples.
- **y** (*numpy array of shape (n_samples,), optional (default=None)*) – The ground truth of the input samples (labels).

Returns
labels – Class labels for each data sample.

Return type
numpy array of shape (n_samples,)

`get_params`(*deep=True*)

Get parameters for this estimator.

See http://scikit-learn.org/stable/modules/generated/sklearn.base.BaseEstimator.html and sklearn/base.py for more information.

Parameters
deep (*boolean, optional*) – If True, will return the parameters for this estimator and contained subobjects that are estimators.

Returns
params – Parameter names mapped to their values.

Return type
mapping of string to any

predict(*X*)

Predict if a particular sample is an outlier or not.

Parameters
X (*numpy array of shape (n_samples, n_features)*) – The input samples.

Returns
outlier_labels – For each observation, tells whether or not it should be considered as an outlier according to the fitted model. 0 stands for inliers and 1 for outliers.

Return type
numpy array of shape (n_samples,)

predict_proba(*X*, *proba_method='linear'*)

Predict the probability of a sample being outlier. Two approaches are possible:

1. simply use Min-max conversion to linearly transform the outlier scores into the range of [0,1]. The model must be fitted first.
2. use unifying scores, see [BKKSZ11].

Parameters

- **X** (*numpy array of shape (n_samples, n_features)*) – The input samples.
- **proba_method** (*str, optional (default='linear')*) – Probability conversion method. It must be one of 'linear' or 'unify'.

Returns
outlier_labels – For each observation, tells whether or not it should be considered as an outlier according to the fitted model. Return the outlier probability, ranging in [0,1].

Return type
numpy array of shape (n_samples,)

set_params(***params*)

Set the parameters of this estimator. The method works on simple estimators as well as on nested objects (such as pipelines). The latter have parameters of the form `<component>__<parameter>` so that it's possible to update each component of a nested object.

See http://scikit-learn.org/stable/modules/generated/sklearn.base.BaseEstimator.html and sklearn/base.py for more information.

Returns
self

Return type
object

combo.models.score_comb module

A collection of combination methods for combining raw scores.

combo.models.score_comb.**aom**(*scores*, *n_buckets=5*, *method='static'*, *bootstrap_estimators=False*, *random_state=None*)

Average of Maximum - An ensemble method for combining multiple estimators. See [BAS15] for details.

First dividing estimators into subgroups, take the maximum score as the subgroup score. Finally, take the average of all subgroup scores.

Parameters

- **scores** (*numpy array of shape (n_samples, n_estimators)*) – The score matrix outputted from various estimators
- **n_buckets** (*int, optional (default=5)*) – The number of subgroups to build
- **method** (*str, optional (default='static')*) – {'static', 'dynamic'}, if 'dynamic', build subgroups randomly with dynamic bucket size.
- **bootstrap_estimators** (*bool, optional (default=False)*) – Whether estimators are drawn with replacement.
- **random_state** (*int, RandomState instance or None, optional (default=None)*) – If int, random_state is the seed used by the random number generator; If RandomState instance, random_state is the random number generator; If None, the random number generator is the RandomState instance used by *np.random*.

Returns
combined_scores – The combined scores.

Return type
Numpy array of shape (n_samples,)

combo.models.score_comb.**average**(*scores*, *estimator_weights=None*)

Combination method to merge the scores from multiple estimators by taking the average.

Parameters

- **scores** (*numpy array of shape (n_samples, n_estimators)*) – Score matrix from multiple estimators on the same samples.
- **estimator_weights** (*numpy array of shape (1, n_estimators)*) – If specified, using weighted average.

Returns
combined_scores – The combined scores.

Return type
numpy array of shape (n_samples,)

combo.models.score_comb.**majority_vote**(*scores*, *n_classes=2*, *weights=None*)

Combination method to merge the scores from multiple estimators by majority vote.

Parameters

- **scores** (*numpy array of shape (n_samples, n_estimators)*) – Score matrix from multiple estimators on the same samples.
- **n_classes** (*int, optional (default=2)*) – The number of classes in scores matrix
- **weights** (*numpy array of shape (1, n_estimators)*) – If specified, using weighted majority weight.

Returns
combined_scores – The combined scores.

Return type
numpy array of shape (n_samples,)

combo.models.score_comb.**maximization**(*scores*)

Combination method to merge the scores from multiple estimators by taking the maximum.

Parameters
scores (*numpy array of shape (n_samples, n_estimators)*) – Score matrix from multiple estimators on the same samples.

Returns
combined_scores – The combined scores.

Return type
numpy array of shape (n_samples,)

combo.models.score_comb.**median**(*scores*)

Combination method to merge the scores from multiple estimators by taking the median.

Parameters
scores (*numpy array of shape (n_samples, n_estimators)*) – Score matrix from multiple estimators on the same samples.

Returns
combined_scores – The combined scores.

Return type
numpy array of shape (n_samples,)

combo.models.score_comb.**moa**(*scores*, *n_buckets=5*, *method='static'*, *bootstrap_estimators=False*, *random_state=None*)

Maximization of Average - An ensemble method for combining multiple estimators. See [BAS15] for details.

First dividing estimators into subgroups, take the average score as the subgroup score. Finally, take the maximization of all subgroup outlier scores.

Parameters

- **scores** (*numpy array of shape (n_samples, n_estimators)*) – The score matrix outputted from various estimators
- **n_buckets** (*int, optional (default=5)*) – The number of subgroups to build
- **method** (*str, optional (default='static')*) – {'static', 'dynamic'}, if 'dynamic', build subgroups randomly with dynamic bucket size.
- **bootstrap_estimators** (*bool, optional (default=False)*) – Whether estimators are drawn with replacement.
- **random_state** (*int, RandomState instance or None, optional (default=None)*) – If int, random_state is the seed used by the random number generator; If RandomState instance, random_state is the random number generator; If None, the random number generator is the RandomState instance used by *np.random*.

Returns
combined_scores – The combined scores.

Return type
Numpy array of shape (n_samples,)

Module contents

References

Utils Functions

combo.utils.data module

Utility functions for manipulating data

combo.utils.data.**evaluate_print**(*clf_name*, *y*, *y_pred*)

Utility function for evaluating and printing the results for examples. Default metrics include accuracy, roc, and F1 score

Parameters

- **clf_name** (*str*) – The name of the estimator.
- **y** (*list or numpy array of shape (n_samples,)*) – The ground truth.
- **y_pred** (*list or numpy array of shape (n_samples,)*) – The raw scores as returned by a fitted model.

combo.utils.utility module

A set of utility functions to support model combination.

combo.utils.utility.**argmaxn**(*value_list*, *n*, *order='desc'*)

Return the index of top n elements in the list if order is set to 'desc', otherwise return the index of n smallest ones.

Parameters

- **value_list** (*list, array, numpy array of shape (n_samples,)*) – A list containing all values.
- **n** (*int*) – The number of elements to select.
- **order** (*str, optional (default='desc')*) – The order to sort {'desc', 'asc'}:
  - 'desc': descending
  - 'asc': ascending

Returns

index_list – The index of the top n elements.

Return type

numpy array of shape (n,)

combo.utils.utility.**check_detector**(*detector*)

Checks if fit and decision_function methods exist for given detector

Parameters

detector (*combo.models*) – Detector instance for which the check is performed.

combo.utils.utility.**generate_bagging_indices**(*random_state*, *bootstrap_features*, *n_features*, *min_features*, *max_features*)

Randomly draw feature indices. Internal use only.

Modified from sklearn/ensemble/bagging.py

Parameters

- **random_state** (*RandomState*) – A random number generator instance to define the state of the random permutations generator.
- **bootstrap_features** (*bool*) – Specifies whether to bootstrap indice generation
- **n_features** (*int*) – Specifies the population size when generating indices
- **min_features** (*int*) – Lower limit for number of features to randomly sample
- **max_features** (*int*) – Upper limit for number of features to randomly sample

Returns
feature_indices – Indices for features to bag

Return type
numpy array, shape (n_samples,)

combo.utils.utility.**generate_indices**(*random_state*, *bootstrap*, *n_population*, *n_samples*)

Draw randomly sampled indices. Internal use only.

See sklearn/ensemble/bagging.py

Parameters

- **random_state** (*RandomState*) – A random number generator instance to define the state of the random permutations generator.
- **bootstrap** (*bool*) – Specifies whether to bootstrap indice generation
- **n_population** (*int*) – Specifies the population size when generating indices
- **n_samples** (*int*) – Specifies number of samples to draw

Returns
indices – randomly drawn indices

Return type
numpy array, shape (n_samples,)

combo.utils.utility.**get_label_n**(*y*, *y_pred*, *n=None*)

Function to turn raw outlier scores into binary labels by assign 1 to top n outlier scores.

Parameters

- **y** (*list or numpy array of shape (n_samples,)*) – The ground truth. Binary (0: inliers, 1: outliers).
- **y_pred** (*list or numpy array of shape (n_samples,)*) – The raw outlier scores as returned by a fitted model.
- **n** (*int, optional (default=None)*) – The number of outliers. if not defined, infer using ground truth.

Returns
labels – binary labels 0: normal points and 1: outliers

Return type
numpy array of shape (n_samples,)

Examples

```
>>> from combo.utils.utility import get_label_n
>>> y = [0, 1, 1, 0, 0]
>>> y_pred = [0.1, 0.5, 0.3, 0.2, 0.7]
>>> get_label_n(y, y_pred)
array([0, 1, 0, 0, 1])
```

combo.utils.utility.**invert_order**(*scores*, *method='multiplication'*)

Invert the order of a list of values. The smallest value becomes the largest in the inverted list. This is useful while combining multiple detectors since their score order could be different.

Parameters

- **scores** (*list, array or numpy array with shape (n_samples,)*) – The list of values to be inverted
- **method** (*str, optional (default='multiplication')*) – Methods used for order inversion. Valid methods are:
  - 'multiplication': multiply by -1
  - 'subtraction': max(scores) - scores

Returns
inverted_scores – The inverted list

Return type
numpy array of shape (n_samples,)

Examples

```
>>> scores1 = [0.1, 0.3, 0.5, 0.7, 0.2, 0.1]
>>> invert_order(scores1)
array([-0.1, -0.3, -0.5, -0.7, -0.2, -0.1])
>>> invert_order(scores1, method='subtraction')
array([0.6, 0.4, 0.2, 0. , 0.5, 0.6])
```

combo.utils.utility.**list_diff**(*first_list*, *second_list*)

Utility function to calculate list difference (first_list-second_list)

Parameters

- **first_list** (*list*) – First list.
- **second_list** (*list*) – Second list.

Returns
diff

Return type
different elements.

combo.utils.utility.**precision_n_scores**(*y*, *y_pred*, *n=None*)

Utility function to calculate precision @ rank n.

Parameters

- **y** (*list or numpy array of shape (n_samples,)*) – The ground truth. Binary (0: inliers, 1: outliers).

- **y_pred** (*list or numpy array of shape (n_samples,)*) – The raw outlier scores as returned by a fitted model.
- **n** (*int, optional (default=None)*) – The number of outliers. if not defined, infer using ground truth.

Returns
precision_at_rank_n – Precision at rank n score.

Return type
float

combo.utils.utility.**score_to_label**(*pred_scores*, *outliers_fraction=0.1*)

Turn raw outlier outlier scores to binary labels (0 or 1).

Parameters

- **pred_scores** (*list or numpy array of shape (n_samples,)*) – Raw outlier scores. Outliers are assumed have larger values.
- **outliers_fraction** (*float in (0,1)*) – Percentage of outliers.

Returns
outlier_labels – For each observation, tells whether or not it should be considered as an outlier according to the fitted model. Return the outlier probability, ranging in [0,1].

Return type
numpy array of shape (n_samples,)

combo.utils.utility.**score_to_proba**(*scores*)

Internal function to random score matrix into probability.

Parameters
scores (*numpy array of shape (n_samples, n_classes)*) – Raw score matrix.

Returns
proba – Scaled probability matrix.

Return type
numpy array of shape (n_samples, n_classes)

combo.utils.utility.**standardizer**(*X*, *X_t=None*, *keep_scalar=False*)

Conduct Z-normalization on data to turn input samples become zero-mean and unit variance.

Parameters

- **X** (*numpy array of shape (n_samples, n_features)*) – The training samples
- **X_t** (*numpy array of shape (n_samples_new, n_features), optional (default=None)*) – The data to be converted
- **keep_scalar** (*bool, optional (default=False)*) – The flag to indicate whether to return the scalar

Returns

- **X_norm** (*numpy array of shape (n_samples, n_features)*) – X after the Z-score normalization
- **X_t_norm** (*numpy array of shape (n_samples, n_features)*) – X_t after the Z-score normalization
- **scalar** (*sklearn scalar object*) – The scalar used in conversion

Module contents

3.5 About us

3.5.1 Core Development Team

Yue Zhao (initialized the project in July 2019): Homepage

3.6 Frequently Asked Questions

3.6.1 What is the Next?

This is the central place to track important things to be fixed/added:

- GPU support
- Installation efficiency improvement, such as using docker
- Add contact channel with Gitter
- Support additional languages, see Manage Translations

Feel free to open on issue report if needed. See Issues.

3.6.2 Inclusion Criteria

Similarly to Similarly to scikit-learn, We mainly consider well-established algorithms for inclusion. A rule of thumb is at least two years since publication, 50+ citations, and usefulness.

However, we encourage the author(s) of newly proposed models to share and add your implementation into combo for boosting ML accessibility and reproducibility. This exception only applies if you could commit to the maintenance of your model for at least two year period.

3.7 Release History

3.7.1 Version 0.1 and earlier

Changelog

Initialize the project by Yue Zhao in Jul 2019.

- v<0.0.0>, <07/14/2019> – Initial release.
- v<0.0.1>, <07/15/2019> – Add basic functionalities and examples.

API

Key changes

- v<0.0.8>, <08/05/2019> – Add fit_predict as core API

New methods

- clustering combination
  - v<0.0.1>, <07/15/2019> – Add clusterer ensemble.
  - v<0.0.8>, <08/06/2019> – Add EAC model.
- classifier combination
  - v<0.0.5>, <07/28/2019> – Add Stacking (meta ensembling).
  - v<0.0.7>, <08/02/2019> – Add DCS_LA.
  - v<0.0.7>, <08/03/2019> – Refactor code for setting weights in Base class.
  - v<0.0.7>, <08/04/2019> – Add DES_LA.
  - v<0.0.8>, <08/06/2019> – Add EAC model.
- score combination
  - v<0.0.5>, <07/27/2019> – Add median combination and score_to_proba function.
- detector combination
  - v<0.0.6>, <07/29/2019> – Add simple outlier detector combination methods.
  - v<0.0.6>, <07/30/2019> – Add LSCP.

Logistics

- v<0.0.3>, <07/17/2019> – Add Travis-ci integration.
- v<0.0.4>, <07/17/2019> – Update unit test and clustering algorithms.
- v<0.0.4>, <07/21/2019> – Add code maintainability.
- v<0.0.6>, <07/29/2019> – Enable Appveyor integration.

Requirement Change:

- v<0.0.6>, <07/29/2019> – Update requirements file.

Examples

- v<0.0.8>, <08/08/2019> – Update clustering examples to show visualization.
- v<0.0.9>, <09/01/2019> – Add classifier_multiple_libs.py for multiple lib support.

References

CHAPTER

FOUR

INDICES AND TABLES

- genindex
- modindex
- search

BIBLIOGRAPHY

[BAS15] Charu C Aggarwal and Saket Sathe. Theoretical foundations and algorithms for outlier ensembles. *ACM SIGKDD Explorations Newsletter*, 17(1):24–47, 2015.

[BFJ05] Ana LN Fred and Anil K Jain. Combining multiple clusterings using evidence accumulation. *IEEE transactions on pattern analysis and machine intelligence*, 27(6):835–850, 2005.

[BHHS94] Tin Kam Ho, Jonathan J. Hull, and Sargur N. Srihari. Decision combination in multiple classifier systems. *IEEE Transactions on Pattern Analysis & Machine Intelligence*, pages 66–75, 1994.

[BKSBJ08] Albert HR Ko, Robert Sabourin, and Alceu Souza Britto Jr. From dynamic classifier selection to dynamic ensemble selection. *Pattern recognition*, 41(5):1718–1731, 2008.

[BKKSZ11] Hans-Peter Kriegel, Peer Kroger, Erich Schubert, and Arthur Zimek. Interpreting and unifying outlier scores. In *Proceedings of the 2011 SIAM International Conference on Data Mining*, 13–24. SIAM, 2011.

[BWKB97] Kevin Woods, W. Philip Kegelmeyer, and Kevin Bowyer. Combination of multiple classifiers using local accuracy estimates. *IEEE transactions on pattern analysis and machine intelligence*, 19(4):405–410, 1997.

[BZNHL19] Yue Zhao, Zain Nasrullah, Maciej K Hryniewicki, and Zheng Li. LSCP: locally selective combination in parallel outlier ensembles. In *Proceedings of the 2019 SIAM International Conference on Data Mining, SDM 2019*, 585–593. Calgary, Canada, May 2019. SIAM. URL: https://doi.org/10.1137/1.9781611975673.66, doi:10.1137/1.9781611975673.66.

[BZT06] Zhi-Hua Zhou and Wei Tang. Clusterer ensemble. *Knowledge-Based Systems*, 19(1):77–83, 2006.

[AAS15] Charu C Aggarwal and Saket Sathe. Theoretical foundations and algorithms for outlier ensembles. *ACM SIGKDD Explorations Newsletter*, 17(1):24–47, 2015.

[AAS17] Charu C Aggarwal and Saket Sathe. *Outlier ensembles: An introduction*. Springer, 2017.

[ABK07] Robert M Bell and Yehuda Koren. Lessons from the netflix prize challenge. *SIGKDD Explorations*, 9(2):75–79, 2007.

[AFJ05] Ana LN Fred and Anil K Jain. Combining multiple clusterings using evidence accumulation. *IEEE transactions on pattern analysis and machine intelligence*, 27(6):835–850, 2005.

[AGor16] Ben Gorman. A kaggler's guide to model stacking in practice. Available at http://blog.kaggle.com/2016/12/27/a-kagglers-guide-to-model-stacking-in-practice, 2016.

[AKSBJ08] Albert HR Ko, Robert Sabourin, and Alceu Souza Britto Jr. From dynamic classifier selection to dynamic ensemble selection. *Pattern recognition*, 41(5):1718–1731, 2008.

[ARPN20] Sebastian Raschka, Joshua Patterson, and Corey Nolet. Machine learning in python: main developments and technology trends in data science, machine learning, and artificial intelligence. *arXiv preprint arXiv:2002.04803*, 2020.

[AWKB97] Kevin Woods, W. Philip Kegelmeyer, and Kevin Bowyer. Combination of multiple classifiers using local accuracy estimates. *IEEE transactions on pattern analysis and machine intelligence*, 19(4):405–410, 1997.

[AZH18] Yue Zhao and Maciej K Hryniewicki. XGBOD: improving supervised outlier detection with unsupervised representation learning. In *2018 International Joint Conference on Neural Networks, IJCNN 2018*, 1–8. IEEE, 2018. URL: https://doi.org/10.1109/IJCNN.2018.8489605, doi:10.1109/IJCNN.2018.8489605.

[AZNHL19] Yue Zhao, Zain Nasrullah, Maciej K Hryniewicki, and Zheng Li. LSCP: locally selective combination in parallel outlier ensembles. In *Proceedings of the 2019 SIAM International Conference on Data Mining, SDM 2019*, 585–593. Calgary, Canada, May 2019. SIAM. URL: https://doi.org/10.1137/1.9781611975673.66, doi:10.1137/1.9781611975673.66.

[AZNL19] Yue Zhao, Zain Nasrullah, and Zheng Li. Pyod: a python toolbox for scalable outlier detection. *Journal of Machine Learning Research*, 20:1–7, 2019.

[AZho12] Zhi-Hua Zhou. *Ensemble methods: foundations and algorithms*. Chapman and Hall/CRC, 2012.

[AZT06] Zhi-Hua Zhou and Wei Tang. Clusterer ensemble. *Knowledge-Based Systems*, 19(1):77–83, 2006.

PYTHON MODULE INDEX

C

INDEX

G

I

L

M

P

S

T

Z