pyod Documentation

Release 0.6.7

Yue Zhao

Jan 16, 2019

Getting Started

Deployment & Documentation & Stats

Build Status & Code Coverage & Maintainability

PyOD is a comprehensive and scalable **Python toolkit** for **detecting outlying objects** in multivariate data. This exciting yet challenging field is commonly referred as Outlier Detection or Anomaly Detection. Since 2017, PyOD has been successfully used in various academic researches and commercial products *[AZH18a][AZH18b][AZHNL19]*. PyOD is featured for:

- **Unified APIs, detailed documentation, and interactive examples** across various algorithms.
- **Advanced models**, including **Neural Networks/Deep Learning** and **Outlier Ensembles**.
- **Optimized performance with JIT and parallelization** when possible, using numba and joblib.
- **Compatible with both Python 2 & 3** (scikit-learn compatible as well).

Important Notes: PyOD contains neural network based models, e.g., AutoEncoders, which are implemented in Keras. However, PyOD would **NOT** install **Keras** and/or **TensorFlow** automatically. This reduces the risk of damaging your local copies. If you want to use neural net based models, you should install Keras and back-end libraries like TensorFlow manually. An instruction is provided: neural-net FAQ. Similarly, some models, e.g., XGBOD, depend on **xgboost**, which would **NOT** be installed by default.

Key Links and Resources:

- View the latest codes on Github
- Execute Interactive Jupyter Notebooks
- Anomaly Detection Resources

CHAPTER 1

Quick Introduction

PyOD toolkit consists of three major groups of functionalities:

(i) Individual Detection Algorithms :

1. Linear Models for Outlier Detection:

Type	Abbr	Algorithm	Year	Class	Ref
Linear Model	PCA	Principal Component Analysis (the sum of weighted projected distances to the eigenvector hyperplanes)	2003	*pyod.models.pca.PCA*	*[ASCSC03]*
Linear Model	MCD	Minimum Covariance Determinant (use the mahalanobis distances as the outlier scores)	1999	*pyod.models.mcd.MCD*	*[ARD99][AHR04]*
Linear Model	OCSVM	One-Class Support Vector Machines	2003	*pyod.models.ocsvm.OCSVM*	*[AMP03]*
Proximity-Based	LOF	Local Outlier Factor	2000	*pyod.models.lof.LOF*	*[ABKNS00]*
Proximity-Based	CBLOF	Clustering-Based Local Outlier Factor	2003	*pyod.models.cblof.CBLOF*	*[AHXD03]*:
Proximity-Based	LOCI	LOCI: Fast outlier detection using the local correlation integral	2003	*pyod.models.loci.LOCI*	*[APKGF03]*
Proximity-Based	HBOS	Histogram-based Outlier Score	2012	*pyod.models.hbos.HBOS*	*[AGD12]*
Proximity-Based	kNN	k Nearest Neighbors (use the distance to the kth nearest neighbor as the outlier score	2000	*pyod.models.knn.KNN*	*[ARRS00][AAP02]*
Proximity-Based	AvgKNN	Average kNN (use the average distance to k nearest neighbors as the outlier score)	2002	*pyod.models.knn.KNN*	*[ARRS00][AAP02]*
Proximity-Based	MedKNN	Median kNN (use the median distance to k nearest neighbors as the outlier score)	2002	*pyod.models.knn.KNN*	*[ARRS00][AAP02]*
Probabilistic	ABOD	Angle-Based Outlier Detection	2008	*pyod.models.abod.ABOD*	*[AKZ+08]*
Probabilistic	FastABOD	Fast Angle-Based Outlier Detection using approximation	2008	*pyod.models.abod.ABOD*	*[AKZ+08]*
Probabilistic	SOS	Stochastic Outlier Selection	2012	*pyod.models.sos.SOS*	*[AJHuszarPvdH12]*
Outlier Ensembles	IForest	Isolation Forest	2008	*pyod.models.iforest.IForest*	*[ALTZ08][ALTZ12]*
Outlier Ensembles		Feature Bagging	2005	*pyod.models.feature_bagging.FeatureBagging*	*[ALK05]*
Outlier Ensembles	LSCP	LSCP: Locally Selective Combination of Parallel Outlier Ensembles	2019	*pyod.models.lscp.LSCP*	*[AZHNL19]*
Outlier Ensembles	XGBOD	Extreme Boosting Based Outlier Detection **(Supervised)**	2018	*pyod.models.xgbod.XGBOD*	*[AZH18a]*
Neural Networks	AutoEncoder	Fully connected AutoEncoder (use reconstruction error as the outlier score)	2015	*pyod.models.auto_encoder.AutoEncoder*	*[AAgg15]*
Neural Networks	SO_GAAL	Single-Objective Generative Adversarial Active Learning	2019	*pyod.models.so_gaal.SO_GAAL*	*[ALLZ+18]*
Neural Networks	MO_GAAL	Multiple-Objective Generative Adversarial Active Learning	2019	*pyod.models.mo_gaal.MO_GAAL*	*[ALLZ+18]*

(ii) Outlier Ensembles & Outlier Detector Combination Frameworks:

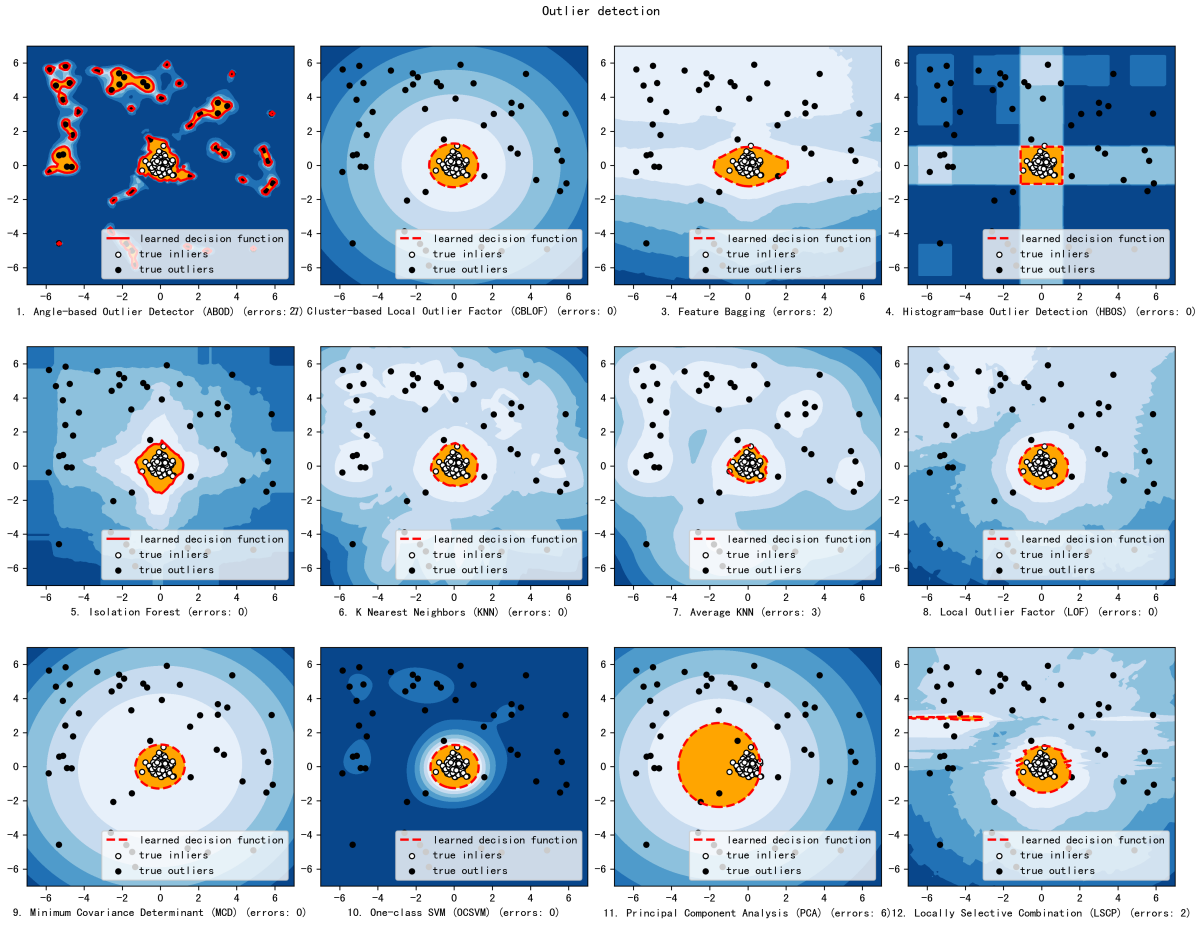
Type	Abbr	Algorithm	Year	Ref	
Outlier Ensembles		Feature Bagging	2005	*pyod.models.feature_bagging.FeatureBagging*	*[ALK05]*
Outlier Ensembles	LSCP	LSCP: Locally Selective Combination of Parallel Outlier Ensembles	2019	*pyod.models.lscp.LSCP*	*[AZHNL19]*
Combination	Average	Simple combination by averaging the scores	2015	*pyod.models.combination.average()*	*[AAS15]*
Combination	Weighted Average	Simple combination by averaging the scores with detector weights	2015	*pyod.models.combination.average()*	*[AAS15]*
Combination	Maximization	Simple combination by taking the maximum scores	2015	*pyod.models.combination.maximization()*	*[AAS15]*
Combination	AOM	Average of Maximum	2015	*pyod.models.combination.aom()*	*[AAS15]*
Combination	MOA	Maximization of Average	2015	*pyod.models.combination.moa()*	*[AAS15]*

(iii) Utility Functions:

Type	Name	Function
Data	*pyod.utils.data.generate_data()*	Synthesized data generation; normal data is generated by a multivariate Gaussian and outliers are generated by a uniform distribution
Stat	*pyod.utils.stat_models.wpearsonr()*	Calculate the weighted Pearson correlation of two samples
Utility	*pyod.utils.utility.get_label_n()*	Turn raw outlier scores into binary labels by assign 1 to top n outlier scores
Utility	*pyod.utils.utility.precision_n_scores()*	calculate precision @ rank n

Comparison of all implemented models are made available below (Code, Jupyter Notebooks):

For Jupyter Notebooks, please navigate to **"/notebooks/Compare All Models.ipynb"**


Outlier detection
learned decision function
true inliers
true outliers
1. Angle-based Outlier Detector (ABOD) (errors: 27)
2. Cluster-based Local Outlier Factor (CBLOF) (errors: 0)
3. Feature Bagging (errors: 2)
4. Histogram-base Outlier Detection (HBOS) (errors: 0)
5. Isolation Forest (errors: 0)
6. K Nearest Neighbors (KNN) (errors: 0)
7. Average KNN (errors: 3)
8. Local Outlier Factor (LOF) (errors: 0)
9. Minimum Covariance Determinant (MCD) (errors: 0)
10. One-class SVM (OCSVM) (errors: 0)
11. Principal Component Analysis (PCA) (errors: 6)
12. Locally Selective Combination (LSCP) (errors: 2)

CHAPTER 2

Key APIs & Attributes

The following APIs are applicable for all detector models for easy use.

- *pyod.models.base.BaseDetector.fit()*: Fit detector. y is optional for unsupervised methods.
- *pyod.models.base.BaseDetector.fit_predict()*: Fit detector first and then predict whether a particular sample is an outlier or not.
- *pyod.models.base.BaseDetector.fit_predict_score()*: Fit the detector, predict on samples, and evaluate the model by predefined metrics, e.g., ROC.
- *pyod.models.base.BaseDetector.decision_function()*: Predict raw anomaly score of X using the fitted detector.
- *pyod.models.base.BaseDetector.predict()*: Predict if a particular sample is an outlier or not using the fitted detector.
- *pyod.models.base.BaseDetector.predict_proba()*: Predict the probability of a sample being outlier using the fitted detector.

Key Attributes of a fitted model:

- *pyod.models.base.BaseDetector.decision_scores_*: The outlier scores of the training data. The higher, the more abnormal. Outliers tend to have higher scores.
- *pyod.models.base.BaseDetector.labels_*: The binary labels of the training data. 0 stands for inliers and 1 for outliers/anomalies.

2.1 Installation

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

```
pip install pyod
pip install --upgrade pyod # make sure the latest version is installed!
```

Alternatively, install from github directly (**NOT Recommended**)

```
git clone https://github.com/yzhao062/pyod.git
python setup.py install
```

Required Dependencies:

- Python 2.7, 3.5, 3.6, or 3.7
- numpy>=1.13
- numba>=0.35
- scipy>=0.19.1
- scikit_learn>=0.19.1

Optional Dependencies (see details below):

- Keras (optional, required for AutoEncoder)
- Matplotlib (optional, required for running examples)
- Tensorflow (optional, required for AutoEncoder, other backend works)
- XGBoost (optional, required for XGBOD)

Warning: Running examples needs Matplotlib, which may throw errors in conda virtual environment on macOS. See mac_matplotlib.

Warning: Keras and/or TensorFlow are listed as optional. However, they are both required if you want to use neural network based models, such as AutoEncoder. See reasons and solutions neural-net FAQ.

Warning: xgboost is listed as optional. However, it is required to run XGBOD. Users are expected to install **xgboost** to use XGBOD model.

2.2 Examples

2.2.1 List of Examples

examples.abod_example module

Example of using Angle-base outlier detection (ABOD) for outlier detection

examples.abod_example.**visualize**(*clf_name*, *X_train*, *y_train*, *X_test*, *y_test*, *y_train_pred*, *y_test_pred*, *show_figure=True*, *save_figure=False*)

Utility function for visualizing the results in examples. Internal use only.

Parameters

- **clf_name** (*str*) – The name of the detector.
- **X_train** (*numpy array of shape (n_samples, n_features)*) – The training samples.

- **y_train** (*list or array of shape (n_samples,)*) – The ground truth of training samples.
- **X_test** (*numpy array of shape (n_samples, n_features)*) – The test samples.
- **y_test** (*list or array of shape (n_samples,)*) – The ground truth of test samples.
- **y_train_pred** (*numpy array of shape (n_samples, n_features)*) – The predicted binary labels of the training samples.
- **y_test_pred** (*numpy array of shape (n_samples, n_features)*) – The predicted binary labels of the test samples.
- **show_figure** (*bool, optional (default=True)*) – If set to True, show the figure.
- **save_figure** (*bool, optional (default=False)*) – If set to True, save the figure to the local.

examples.comb_example module

Example of combining multiple base outlier scores. Four combination frameworks are demonstrated:

1. Average: take the average of all base detectors
2. maximization : take the maximum score across all detectors as the score
3. Average of Maximum (AOM)
4. Maximum of Average (MOA)

examples.feature_bagging_example module

Example of using Feature Bagging for outlier detection

examples.feature_bagging_example.**visualize**(*clf_name*, *X_train*, *y_train*, *X_test*, *y_test*, *y_train_pred*, *y_test_pred*, *show_figure=True*, *save_figure=False*)

Utility function for visualizing the results in examples. Internal use only.

Parameters

- **clf_name** (*str*) – The name of the detector.
- **X_train** (*numpy array of shape (n_samples, n_features)*) – The training samples.
- **y_train** (*list or array of shape (n_samples,)*) – The ground truth of training samples.
- **X_test** (*numpy array of shape (n_samples, n_features)*) – The test samples.
- **y_test** (*list or array of shape (n_samples,)*) – The ground truth of test samples.
- **y_train_pred** (*numpy array of shape (n_samples, n_features)*) – The predicted binary labels of the training samples.

- **y_test_pred** (*numpy array of shape (n_samples, n_features)*) – The predicted binary labels of the test samples.
- **show_figure** (*bool, optional (default=True)*) – If set to True, show the figure.
- **save_figure** (*bool, optional (default=False)*) – If set to True, save the figure to the local.

examples.hbos_example module

Example of using Histogram- based outlier detection (HBOS) for outlier detection

examples.hbos_example.**visualize**(*clf_name*, *X_train*, *y_train*, *X_test*, *y_test*, *y_train_pred*, *y_test_pred*, *show_figure=True*, *save_figure=False*)

Utility function for visualizing the results in examples. Internal use only.

Parameters

- **clf_name** (*str*) – The name of the detector.
- **X_train** (*numpy array of shape (n_samples, n_features)*) – The training samples.
- **y_train** (*list or array of shape (n_samples,)*) – The ground truth of training samples.
- **X_test** (*numpy array of shape (n_samples, n_features)*) – The test samples.
- **y_test** (*list or array of shape (n_samples,)*) – The ground truth of test samples.
- **y_train_pred** (*numpy array of shape (n_samples, n_features)*) – The predicted binary labels of the training samples.
- **y_test_pred** (*numpy array of shape (n_samples, n_features)*) – The predicted binary labels of the test samples.
- **show_figure** (*bool, optional (default=True)*) – If set to True, show the figure.
- **save_figure** (*bool, optional (default=False)*) – If set to True, save the figure to the local.

examples.iforest_example module

Example of using Isolation Forest for outlier detection

examples.iforest_example.**visualize**(*clf_name*, *X_train*, *y_train*, *X_test*, *y_test*, *y_train_pred*, *y_test_pred*, *show_figure=True*, *save_figure=False*)

Utility function for visualizing the results in examples. Internal use only.

Parameters

- **clf_name** (*str*) – The name of the detector.
- **X_train** (*numpy array of shape (n_samples, n_features)*) – The training samples.
- **y_train** (*list or array of shape (n_samples,)*) – The ground truth of training samples.

- **X_test** (*numpy array of shape (n_samples, n_features)*) – The test samples.
- **y_test** (*list or array of shape (n_samples,)*) – The ground truth of test samples.
- **y_train_pred** (*numpy array of shape (n_samples, n_features)*) – The predicted binary labels of the training samples.
- **y_test_pred** (*numpy array of shape (n_samples, n_features)*) – The predicted binary labels of the test samples.
- **show_figure** (*bool, optional (default=True)*) – If set to True, show the figure.
- **save_figure** (*bool, optional (default=False)*) – If set to True, save the figure to the local.

examples.knn_example module

Example of using kNN for outlier detection

examples.knn_example.**visualize**(*clf_name*, *X_train*, *y_train*, *X_test*, *y_test*, *y_train_pred*, *y_test_pred*, *show_figure=True*, *save_figure=False*)

Utility function for visualizing the results in examples. Internal use only.

Parameters

- **clf_name** (*str*) – The name of the detector.
- **X_train** (*numpy array of shape (n_samples, n_features)*) – The training samples.
- **y_train** (*list or array of shape (n_samples,)*) – The ground truth of training samples.
- **X_test** (*numpy array of shape (n_samples, n_features)*) – The test samples.
- **y_test** (*list or array of shape (n_samples,)*) – The ground truth of test samples.
- **y_train_pred** (*numpy array of shape (n_samples, n_features)*) – The predicted binary labels of the training samples.
- **y_test_pred** (*numpy array of shape (n_samples, n_features)*) – The predicted binary labels of the test samples.
- **show_figure** (*bool, optional (default=True)*) – If set to True, show the figure.
- **save_figure** (*bool, optional (default=False)*) – If set to True, save the figure to the local.

examples.lof_example module

Example of using LOF for outlier detection

examples.lof_example.**visualize**(*clf_name*, *X_train*, *y_train*, *X_test*, *y_test*, *y_train_pred*, *y_test_pred*, *show_figure=True*, *save_figure=False*)

Utility function for visualizing the results in examples. Internal use only.

Parameters

- **clf_name** (*str*) – The name of the detector.
- **X_train** (*numpy array of shape (n_samples, n_features)*) – The training samples.
- **y_train** (*list or array of shape (n_samples,)*) – The ground truth of training samples.
- **X_test** (*numpy array of shape (n_samples, n_features)*) – The test samples.
- **y_test** (*list or array of shape (n_samples,)*) – The ground truth of test samples.
- **y_train_pred** (*numpy array of shape (n_samples, n_features)*) – The predicted binary labels of the training samples.
- **y_test_pred** (*numpy array of shape (n_samples, n_features)*) – The predicted binary labels of the test samples.
- **show_figure** (*bool, optional (default=True)*) – If set to True, show the figure.
- **save_figure** (*bool, optional (default=False)*) – If set to True, save the figure to the local.

examples.ocsvm_example module

Example of using one class SVM for outlier detection

examples.ocsvm_example.**visualize**(*clf_name*, *X_train*, *y_train*, *X_test*, *y_test*, *y_train_pred*, *y_test_pred*, *show_figure=True*, *save_figure=False*)

Utility function for visualizing the results in examples. Internal use only.

Parameters

- **clf_name** (*str*) – The name of the detector.
- **X_train** (*numpy array of shape (n_samples, n_features)*) – The training samples.
- **y_train** (*list or array of shape (n_samples,)*) – The ground truth of training samples.
- **X_test** (*numpy array of shape (n_samples, n_features)*) – The test samples.
- **y_test** (*list or array of shape (n_samples,)*) – The ground truth of test samples.
- **y_train_pred** (*numpy array of shape (n_samples, n_features)*) – The predicted binary labels of the training samples.
- **y_test_pred** (*numpy array of shape (n_samples, n_features)*) – The predicted binary labels of the test samples.
- **show_figure** (*bool, optional (default=True)*) – If set to True, show the figure.
- **save_figure** (*bool, optional (default=False)*) – If set to True, save the figure to the local.

examples.pca_example module

Example of using PCA for outlier detection

examples.pca_example.**visualize**(*clf_name*, *X_train*, *y_train*, *X_test*, *y_test*, *y_train_pred*, *y_test_pred*, *show_figure=True*, *save_figure=False*)

Utility function for visualizing the results in examples. Internal use only.

Parameters

- **clf_name** (*str*) – The name of the detector.
- **X_train** (*numpy array of shape (n_samples, n_features)*) – The training samples.
- **y_train** (*list or array of shape (n_samples,)*) – The ground truth of training samples.
- **X_test** (*numpy array of shape (n_samples, n_features)*) – The test samples.
- **y_test** (*list or array of shape (n_samples,)*) – The ground truth of test samples.
- **y_train_pred** (*numpy array of shape (n_samples, n_features)*) – The predicted binary labels of the training samples.
- **y_test_pred** (*numpy array of shape (n_samples, n_features)*) – The predicted binary labels of the test samples.
- **show_figure** (*bool, optional (default=True)*) – If set to True, show the figure.
- **save_figure** (*bool, optional (default=False)*) – If set to True, save the figure to the local.

2.2.2 kNN Example

Full example: knn_example.py

1. Import models

```
from pyod.models.knn import KNN   # kNN detector
```

2. Generate sample data with *pyod.utils.data.generate_data()*:

```
contamination = 0.1  # percentage of outliers
n_train = 200  # number of training points
n_test = 100  # number of testing points

X_train, y_train, X_test, y_test = generate_data(
    n_train=n_train, n_test=n_test, contamination=contamination)
```

3. Initialize a *pyod.models.knn.KNN* detector, fit the model, and make the prediction:

```
# train kNN detector
clf_name = 'KNN'
clf = KNN()
clf.fit(X_train)
```

(continues on next page)

(continued from previous page)

```
# get the prediction labels and outlier scores of the training data
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
```

4. Evaluate the prediction using ROC and Precision@rank n *pyod.utils.data.evaluate_print()*:

```
# 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)
```

5. See sample outputs on both training and test data:

```
On Training Data:
KNN ROC:1.0, precision @ rank n:1.0

On Test Data:
KNN ROC:0.9989, precision @ rank n:0.9
```

6. Generate the visualizations by visualize function included in all examples:

```
visualize(clf_name, X_train, y_train, X_test, y_test, y_train_pred,
          y_test_pred, show_figure=True, save_figure=False)
```

2.2.3 Model Combination Example

comb_example.py is a quick demo for showing the API for combining multiple algorithms. Given we have *n* individual outlier detectors, each of them generates an individual score for all samples. The task is to combine the outputs from these detectors effectivelly.

Model combination example is made available below (Code, Jupyter Notebooks):

For Jupyter Notebooks, please navigate to **"/notebooks/Model Combination.ipynb"**

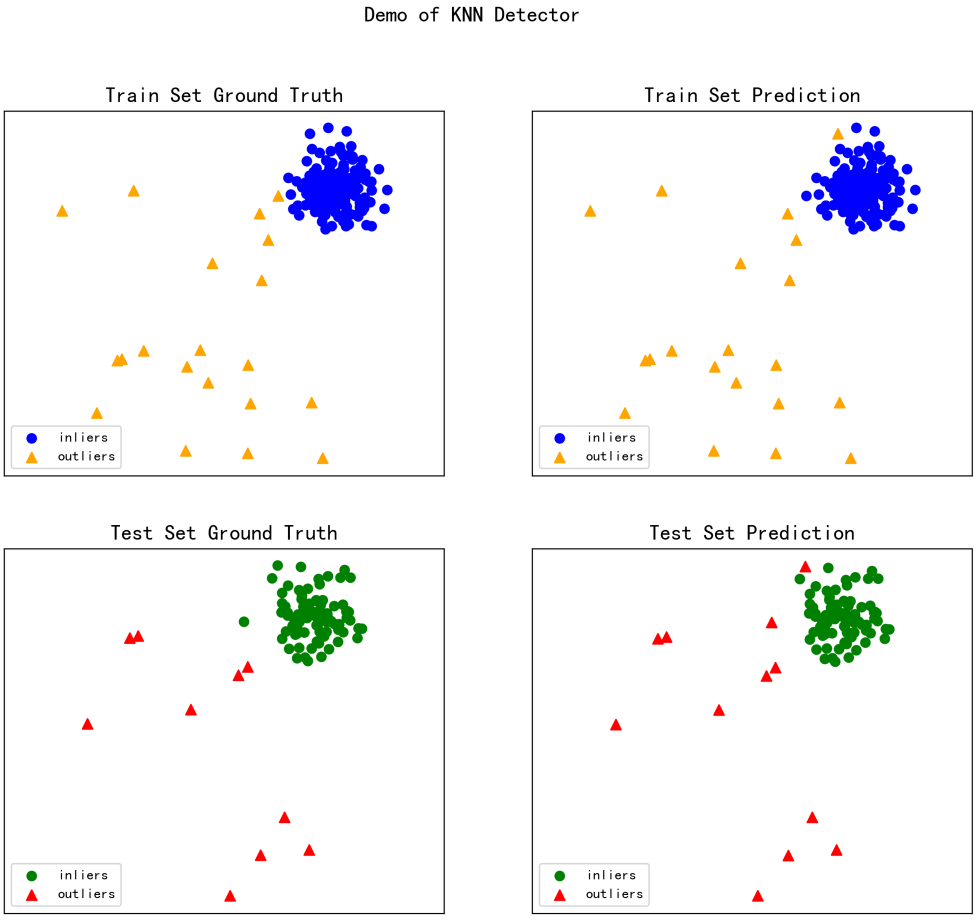
1. Import models and generate sample data:

```
from pyod.models.knn import KNN  # kNN detector
from pyod.models.combination import aom, moa, average, maximization
from pyod.utils.data import generate_data

X, y= generate_data(train_only=True)  # load data
```

2. First initialize 20 kNN outlier detectors with different k (10 to 200), and get the outlier scores:

```
# initialize 20 base detectors for combination
k_list = [10, 20, 30, 40, 50, 60, 70, 80, 90, 100, 110, 120, 130, 140,
          150, 160, 170, 180, 190, 200]

train_scores = np.zeros([X_train.shape[0], n_clf])
test_scores = np.zeros([X_test.shape[0], n_clf])
```

(continues on next page)


Demo of KNN Detector
Train Set Ground Truth
inliers
outliers
Train Set Prediction
inliers
outliers
Test Set Ground Truth
inliers
outliers
Test Set Prediction
inliers
outliers

(continued from previous page)

```
for i in range(n_clf):
    k = k_list[i]

    clf = KNN(n_neighbors=k, method='largest')
    clf.fit(X_train_norm)

    train_scores[:, i] = clf.decision_scores_
    test_scores[:, i] = clf.decision_function(X_test_norm)
```

3. Then the output codes are standardized into zero average and unit std before combination:

```
from pyod.utils.utility import standardizer

# scores have to be normalized before combination
train_scores_norm, test_scores_norm = standardizer(train_scores, test_
↪scores)
```

4. Then four different combination algorithms are applied as described above:

```
comb_by_average = average(test_scores_norm)
comb_by_maximization = maximization(test_scores_norm)
comb_by_aom = aom(test_scores_norm, 5) # 5 groups
comb_by_moa = moa(test_scores_norm, 5)) # 5 groups
```

5. Finally, all four combination methods are evaluated with ROC and Precision @ Rank n:

```
Combining 20 kNN detectors
Combination by Average ROC:0.9194, precision @ rank n:0.4531
Combination by Maximization ROC:0.9198, precision @ rank n:0.4688
Combination by AOM ROC:0.9257, precision @ rank n:0.4844
Combination by MOA ROC:0.9263, precision @ rank n:0.4688
```

2.3 Benchmarks

2.3.1 Introduction

To provide an overview and guidance of the implemented models, benchmark is supplied below.

In total, 16 benchmark data are used for comparision, all datasets could be downloaded at ODDS.

For each dataset, it is first split into 60% for training and 40% for testing. All experiments are repeated 10 times independently with different splits. The mean of 20 trials is regarded as the final result. Three evaluation metrics are provided:

- The area under receiver operating characteristic (ROC) curve
- Precision @ rank n (P@N)
- Execution time

Note: LSCP is a combination framework. In this benchmark it is based on 5 LOF detector (n_neighbors=[10,. . . ,50]), so it is only meaningful to compare LSCP with LOF, instead of other detection algorithms.

You are welcome to replicate this process by running: benchmark.py

2.3.2 ROC Performance

Table 1: ROC Performances (average of 10 independent trials)

Data	#Samples	# Dimensions	Outlier Perc	ABOD	CBLOF	FB	HBOS	IForest	KNN	LOF	MCD	OCSVM	PCA	LSCP
arrhythmia	452	274	14.6018	0.7688	0.7854	0.7768	0.8219	0.8111	0.7861	0.7787	0.7791	0.7812	0.7815	0.7825
cardio	1831	21	9.6122	0.5692	0.9269	0.6034	0.8351	0.9257	0.7236	0.5736	0.8267	0.9348	0.9504	0.7078
glass	214	9	4.2056	0.7951	0.7953	0.8379	0.7389	0.7258	0.8508	0.8644	0.7877	0.6324	0.6747	0.8389
ionosphere	351	33	35.8974	0.9248	0.8122	0.8677	0.5614	0.8487	0.9267	0.8753	0.9553	0.8419	0.7962	0.882
letter	1600	32	6.25	0.8783	0.5364	0.8654	0.5927	0.6209	0.8766	0.8594	0.8083	0.6118	0.5283	0.8363
lympho	148	18	4.0541	0.9111	0.9712	0.9768	0.9957	0.9911	0.9745	0.9771	0.9159	0.9759	0.9847	0.9771
mnist	7603	100	9.2069	0.7815	0.8757	0.7152	0.5742	0.8003	0.8481	0.7161	0.8461	0.8529	0.8527	0.7652
musk	3062	166	3.1679	0.1844	0.9911	0.5279	1	0.9995	0.7986	0.5287	0.9999	1	1	0.5029
optdigits	5216	64	2.8758	0.4667	0.5081	0.4492	0.8732	0.7097	0.3708	0.45	0.3824	0.4997	0.5086	0.419
pendigits	6870	16	2.2707	0.6878	0.9425	0.4699	0.9238	0.9434	0.7486	0.4698	0.8341	0.9303	0.9352	0.5036
pima	768	8	34.8958	0.6794	0.7285	0.6271	0.7	0.6731	0.7078	0.6271	0.6753	0.6215	0.6481	0.6501
satellite	6435	36	31.6395	0.5714	0.619	0.5567	0.7581	0.7075	0.6836	0.5573	0.8031	0.6622	0.5988	0.5722
satimage-2	5803	36	1.2235	0.819	0.9827	0.4586	0.9804	0.9951	0.9536	0.4577	0.9967	0.9978	0.9822	0.6852
vertebral	240	6	12.5	0.4262	0.3964	0.4101	0.3263	0.3916	0.3817	0.4081	0.3971	0.4431	0.4027	0.3806
vowels	1456	12	3.4341	0.9606	0.5993	0.9388	0.6727	0.7421	0.968	0.941	0.7912	0.7802	0.6027	0.944
wbc	378	30	5.5556	0.9047	0.9264	0.9335	0.9516	0.9247	0.9366	0.9349	0.9169	0.9319	0.9159	0.9315

2.3.3 P@N Performance

Table 2: Precision @ N Performances (average of 10 independent trials)

Data	#Samples	# Dimensions	Outlier Perc	ABOD	CBLOF	FB	HBOS	IForest	KNN	LOF	MCD	OCSVM	PCA	LSCP
arrhythmia	452	274	14.6018	0.3808	0.4471	0.415	0.5111	0.5125	0.4464	0.4333	0.3995	0.4614	0.461	0.4234
cardio	1831	21	9.6122	0.2374	0.5747	0.1673	0.4476	0.509	0.3323	0.1541	0.4479	0.5011	0.609	0.176
glass	214	9	4.2056	0.1702	0.0726	0.1059	0	0.0726	0.0726	0.1476	0	0.1726	0.0726	0.1226
ionosphere	351	33	35.8974	0.8442	0.5815	0.696	0.3295	0.6477	0.8602	0.7061	0.8811	0.7	0.5729	0.7184
letter	1600	32	6.25	0.3801	0.0554	0.377	0.0715	0.092	0.3312	0.3641	0.1904	0.151	0.0875	0.3165
lympho	148	18	4.0541	0.4483	0.7517	0.7517	0.8467	0.7517	0.7517	0.7517	0.7517	0.435	0.7517	0.7517	0.7517
mnist	7603	100	9.2069	0.3555	0.4271	0.3317	0.1188	0.2897	0.4204	0.3343	0.2485	0.3962	0.3846	0.3502
musk	3062	166	3.1679	0.0507	0.8481	0.2	0.978	0.9667	0.2733	0.1695	0.977	1	0.979	0.1358
optdigits	5216	64	2.8758	0.006	0	0.0248	0.2194	0.0217	0	0.0234	0	0	0	0.022
pendigits	6870	16	2.2707	0.0812	0.2801	0.0615	0.2979	0.3394	0.0984	0.0653	0.0881	0.3287	0.3187	0.0613
pima	768	8	34.8958	0.5193	0.5558	0.4476	0.5424	0.5031	0.5413	0.4555	0.4954	0.4704	0.4943	0.4754
satellite	6435	36	31.6395	0.3902	0.3951	0.3917	0.569	0.5694	0.4994	0.3893	0.6848	0.5346	0.4784	0.4071
satimage-2	5803	36	1.2235	0.213	0.7846	0.0666	0.6939	0.853	0.3809	0.0555	0.6481	0.9356	0.804	0.0508
vertebral	240	6	12.5	0.0601	0.0183	0.0496	0.0071	0.031	0.0238	0.0506	0.0071	0.0238	0.0226	0.0506
vowels	1456	12	3.4341	0.571	0.0731	0.323	0.1297	0.1802	0.5093	0.3551	0.1715	0.2791	0.1364	0.335
wbc	378	30	5.5556	0.306	0.4985	0.5188	0.5817	0.5132	0.4952	0.5188	0.4568	0.5125	0.4767	0.5188

2.3.4 Execution Time

Table 3: Time Complexity in Seconds (average of 10 independent trials)

Data	#Samples	# Dimensions	Outlier Perc	ABOD	CBLOF	FB	HBOS	IForest	KNN	LOF	MCD	OCSVM	PCA	LSCP
arrhythmia	452	274	14.6018	0.374	0.1409	0.5566	0.1469	0.2572	0.0976	0.0723	0.6016	0.0476	0.059	61.8041
cardio	1831	21	9.6122	0.3637	0.0217	0.8016	0.0056	0.2867	0.2226	0.105	0.5143	0.0914	0.003	83.3281
glass	214	9	4.2056	0.035	0.0128	0.0278	0.0016	0.1567	0.0127	0.0022	0.0383	0.0015	0.001	0.2012
ionosphere	351	33	35.8974	0.058	0.0169	0.066	0.0078	0.1849	0.0228	0.006	0.0578	0.0051	0.002	0.3703
letter	1600	32	6.25	0.3319	0.0221	0.7389	0.008	0.2572	0.1637	0.0924	0.9713	0.0873	0.004	53.1145
lympho	148	18	4.0541	0.0284	0.0171	0.027	0.0043	0.1667	0.0087	0.002	0.0338	0.0012	0.001	0.1388
mnist	7603	100	9.2069	7.0412	0.0475	45.7463	0.0478	1.9339	6.7763	6.4203	2.5654	4.7525	0.151	3146.2224
musk	3062	166	3.1679	2.0537	0.0455	11.7174	0.0493	0.9907	1.7191	1.5493	9.68	1.212	0.147	538.6129
optdigits	5216	64	2.8758	2.1879	0.0318	11.8058	0.0255	0.6474	1.7063	1.4831	0.9539	1.3605	0.044	538.288
pendigits	6870	16	2.2707	1.2799	0.0326	3.6209	0.008	0.5069	0.6721	0.536	1.566	0.8901	0.005	415.3573
pima	768	8	34.8958	0.1135	0.0153	0.0837	0.0018	0.1811	0.0443	0.0096	0.0502	0.0104	0.001	0.6555
satellite	6435	36	31.6395	1.6487	0.0333	7.0337	0.0163	0.6374	1.0924	0.9317	1.8058	1.288	0.024	723.2239
satimage2	5803	36	1.2235	1.6925	0.0406	6.12	0.0174	0.6441	1.0753	0.8087	1.9241	1.1575	0.016	923.5951
vertebral	240	6	12.5	0.0448	0.0157	0.0324	0.0017	0.1779	0.0167	0.003	0.0496	0.0018	0.001	10.2316
vowels	1456	12	3.4341	0.268	0.0191	0.2979	0.0037	0.258	0.1093	0.0364	0.7747	0.0424	0.002	1.8441
wbc	378	30	5.5556	0.0695	0.0216	0.0806	0.0079	0.2084	0.0275	0.0074	0.0639	0.0059	0.002	20.4293

2.3.5 Conclusion

TO ADD

2.4 API CheatSheet

- `pyod.models.base.BaseDetector.fit()`: Fit detector. y is optional for unsupervised methods.
- `pyod.models.base.BaseDetector.fit_predict()`: Fit detector first and then predict whether a particular sample is an outlier or not.
- `pyod.models.base.BaseDetector.fit_predict_score()`: Fit the detector, predict on samples, and evaluate the model by predefined metrics, e.g., ROC.
- `pyod.models.base.BaseDetector.decision_function()`: Predict raw anomaly score of X using the fitted detector.

- *pyod.models.base.BaseDetector.predict()*: Predict if a particular sample is an outlier or not using the fitted detector.
- *pyod.models.base.BaseDetector.predict_proba()*: Predict the probability of a sample being outlier using the fitted detector.

See base class definition below:

2.4.1 pyod.models.base module

Base class for all outlier detector models

class pyod.models.base.**BaseDetector**(*contamination=0.1*)
Bases: object

Abstract class for all outlier detection algorithms.

Parameters **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.

decision_scores_
numpy array of shape (n_samples,) – 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.

threshold_
float – 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.

labels_
int, either 0 or 1 – 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_`.

decision_function(*X*)
Predict raw anomaly score of X using the fitted detector.

The anomaly score of an input sample is computed based on different detector algorithms. For consistency, outliers are assigned with larger anomaly scores.

Parameters **X** (*numpy array of shape (n_samples, n_features)*) – The training 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 detector first and then predict whether a particular sample is an outlier or not.

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 **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,)

fit_predict_score(*X*, *y*, *scoring='roc_auc_score'*)

Fit the detector, predict on samples, and evaluate the model by predefined metrics, e.g., ROC.

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).
- **scoring** (*str, optional (default='roc_auc_score')*) – Evaluation metric:
  - 'roc_auc_score': ROC score
  - 'prc_n_score': Precision @ rank n score

Returns **score**

Return type float

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*, *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.
- **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

2.5 API Reference

2.5.1 All Models

pyod.models.abod module

Angle-based Outlier Detector (ABOD)

class pyod.models.abod.**ABOD**(*contamination=0.1*, *n_neighbors=5*, *method='fast'*)

Bases: *pyod.models.base.BaseDetector*

ABOD class for Angle-base Outlier Detection. For an observation, the variance of its weighted cosine scores to all neighbors could be viewed as the outlying score. See *[BKZ+08]* for details.

Two version of ABOD are supported:

- Fast ABOD: use k nearest neighbors to approximate.
- Original ABOD: consider all training points with high time complexity at O(n^3).

Parameters

- **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.
- **n_neighbors** (*int, optional (default=10)*) – Number of neighbors to use by default for k neighbors queries.
- **method** (*str, optional (default='fast')*) – Valid values for metric are:
  - 'fast': fast ABOD. Only consider n_neighbors of training points
  - 'default': original ABOD with all training points, which could be slow

decision_scores_
numpy array of shape (n_samples,) – 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.

threshold_
float – 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.

labels_
int, either 0 or 1 – 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_`.

decision_function(*X*)
Predict raw anomaly score of X using the fitted detector.

The anomaly score of an input sample is computed based on different detector algorithms. For consistency, outliers are assigned with larger anomaly scores.

Parameters **X** (*numpy array of shape (n_samples, n_features)*) – The training 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 detector first and then predict whether a particular sample is an outlier or not.

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 **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,)

fit_predict_score(*X*, *y*, *scoring='roc_auc_score'*)
Fit the detector, predict on samples, and evaluate the model by predefined metrics, e.g., ROC.

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).

- **scoring** (*str, optional (default='roc_auc_score')*) – Evaluation metric:
  - 'roc_auc_score': ROC score
  - 'prc_n_score': Precision @ rank n score

Returns **score**

Return type float

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*, *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.
- **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

pyod.models.auto_encoder module

Using Auto Encoder with Outlier Detection

class pyod.models.auto_encoder.**AutoEncoder**(*hidden_neurons=None*, *hidden_activation='relu'*, *output_activation='sigmoid'*, *loss=<function mean_squared_error>*, *optimizer='adam'*, *epochs=100*, *batch_size=32*, *dropout_rate=0.2*, *l2_regularizer=0.1*, *validation_size=0.1*, *preprocessing=True*, *verbose=1*, *random_state=None*, *contamination=0.1*)

Bases: *pyod.models.base.BaseDetector*

Auto Encoder (AE) is a type of neural networks for learning useful data representations unsupervisedly. Similar to PCA, AE could be used to detect outlying objects in the data by calculating the reconstruction errors. See *[BAgg15]* Chapter 3 for details.

Parameters

- **hidden_neurons** (*list, optional (default=[64, 32, 32, 64])*) – The number of neurons per hidden layers.
- **hidden_activation** (*str, optional (default='relu')*) – Activation function to use for hidden layers. All hidden layers are forced to use the same type of activation. See https://keras.io/activations/
- **output_activation** (*str, optional (default='sigmoid')*) – Activation function to use for output layer. See https://keras.io/activations/
- **loss** (*str or obj, optional (default=keras.losses.mean_squared_error)*) – String (name of objective function) or objective function. See https://keras.io/losses/
- **optimizer** (*str, optional (default='adam')*) – String (name of optimizer) or optimizer instance. See https://keras.io/optimizers/
- **epochs** (*int, optional (default=100)*) – Number of epochs to train the model.
- **batch_size** (*int, optional (default=32)*) – Number of samples per gradient update.
- **dropout_rate** (*float in (0., 1), optional (default=0.2)*) – The dropout to be used across all layers.
- **l2_regularizer** (*float in (0., 1), optional (default=0.1)*) – The regularization strength of activity_regularizer applied on each layer. By default, l2 regularizer is used. See https://keras.io/regularizers/
- **validation_size** (*float in (0., 1), optional (default=0.1)*) – The percentage of data to be used for validation.
- **preprocessing** (*bool, optional (default=True)*) – If True, apply standardization on the data.
- **verbose** (*int, optional (default=1)*) – Verbosity mode.
  - 0 = silent

- 1 = progress bar
- 2 = one line per epoch.

- **random_state** (*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*.
- **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. When fitting this is used to define the threshold on the decision function.

encoding_dim_
int – The number of neurons in the encoding layer.

compression_rate_
float – The ratio between the original feature and the number of neurons in the encoding layer.

model_
Keras Object – The underlying AutoEncoder in Keras.

history_
Keras Object – The AutoEncoder training history.

decision_scores_
numpy array of shape (n_samples,) – 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.

threshold_
float – 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.

labels_
int, either 0 or 1 – 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_`.

decision_function(*X*)
Predict raw anomaly score of X using the fitted detector.

The anomaly score of an input sample is computed based on different detector algorithms. For consistency, outliers are assigned with larger anomaly scores.

Parameters **X** (*numpy array of shape (n_samples, n_features)*) – The training 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 detector first and then predict whether a particular sample is an outlier or not.

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 **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,)

fit_predict_score(*X*, *y*, *scoring='roc_auc_score'*)
Fit the detector, predict on samples, and evaluate the model by predefined metrics, e.g., ROC.

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).
- **scoring** (*str, optional (default='roc_auc_score')*) – Evaluation metric:
  - 'roc_auc_score': ROC score
  - 'prc_n_score': Precision @ rank n score

Returns **score**

Return type float

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*, *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.
- **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

pyod.models.cblof module

Clustering Based Local Outlier Factor (CBLOF)

class pyod.models.cblof.**CBLOF**(*n_clusters=8*, *contamination=0.1*, *clustering_estimator=None*, *alpha=0.9*, *beta=5*, *use_weights=False*, *check_estimator=True*, *random_state=None*)

Bases: *pyod.models.base.BaseDetector*

The CBLOF operator calculates the outlier score based on cluster-based local outlier factor.

CBLOF takes as an input the data set and the cluster model that was generated by a clustering algorithm. It classifies the clusters into small clusters and large clusters using the parameters alpha and beta. The anomaly score is then calculated based on the size of the cluster the point belongs to as well as the distance to the nearest large cluster.

Use weighting for outlier factor based on the sizes of the clusters as proposed in the original publication. Since this might lead to unexpected behavior (outliers close to small clusters are not found), it is disabled by default.Outliers scores are solely computed based on their distance to the closest large cluster center.

By default, MiniBatchKMeans is used for clustering algorithm instead of Squeezer algorithm mentioned in the original paper for multiple reasons.

See *[BHXD03]* for details.

Parameters

- **n_clusters** (*int, optional (default=8)*) – The number of clusters to form as well as the number of centroids to generate.

- **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.
- **clustering_estimator** (*Estimator, optional (default=None)*) – The base clustering algorithm for performing data clustering. A valid clustering algorithm should be passed in. The estimator should have standard sklearn APIs, fit() and predict(). The estimator should have attributes `labels_` and `cluster_centers_`. If `cluster_centers_` is not in the attributes once the model is fit, it is calculated as the mean of the samples in a cluster.

  If not set, CBLOF uses MiniBatchKMeans for scalability. See http://scikit-learn.org/stable/modules/generated/sklearn.cluster.MiniBatchKMeans.html
- **alpha** (*float in (0.5, 1), optional (default=0.9)*) – Coefficient for deciding small and large clusters. The ratio of the number of samples in large clusters to the number of samples in small clusters.
- **beta** (*int or float in (1,), optional (default=5)*) – Coefficient for deciding small and large clusters. For a list sorted clusters by size *|C1|, |C2|, ..., |Cn|, beta = |Ck|/|Ck-1|*
- **use_weights** (*bool, optional (default=False)*) – If set to True, the size of clusters are used as weights in outlier score calculation.
- **check_estimator** (*bool, optional (default=True)*) – If set to True, check whether the base estimator is consistent with sklearn standard.
- **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*.

clustering_estimator_
Estimator, sklearn instance – Base estimator for clustering.

cluster_labels_
list of shape (n_samples,) – Cluster assignment for the training samples.

cluster_sizes_
list of shape (n_clusters,) – The size of each cluster once fitted with the training data.

decision_scores_
numpy array of shape (n_samples,) – 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.

cluster_centers_
numpy array of shape (n_clusters, n_features) – The center of each cluster.

small_cluster_labels_
list of clusters numbers – The cluster assignments belonging to small clusters.

large_cluster_labels_
list of clusters numbers – The cluster assignments belonging to large clusters.

threshold_
float – 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.

labels_
int, either 0 or 1 – 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_`.

decision_function(*X*)
Predict raw anomaly score of X using the fitted detector.

The anomaly score of an input sample is computed based on different detector algorithms. For consistency, outliers are assigned with larger anomaly scores.

Parameters **X** (*numpy array of shape (n_samples, n_features)*) – The training 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 detector first and then predict whether a particular sample is an outlier or not.

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 **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,)

fit_predict_score(*X*, *y*, *scoring='roc_auc_score'*)
Fit the detector, predict on samples, and evaluate the model by predefined metrics, e.g., ROC.

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).
- **scoring** (*str, optional (default='roc_auc_score')*) – Evaluation metric:
  - 'roc_auc_score': ROC score
  - 'prc_n_score': Precision @ rank n score

Returns **score**

Return type float

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*, *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.
- **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

pyod.models.combination module

A collection of model combination functionalities.

pyod.models.combination.**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 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 outlier scores.

Return type Numpy array of shape (n_samples,)

pyod.models.combination.**average**(*scores*, *estimator_weight=None*)

Combination method to merge the outlier 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_weight** (*list of shape (1, n_estimators)*) – If specified, using weighted average

Returns **combined_scores** – The combined outlier scores.

Return type numpy array of shape (n_samples,)

pyod.models.combination.**maximization**(*scores*)

Combination method to merge the outlier 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 outlier scores.

Return type numpy array of shape (n_samples,)

pyod.models.combination.**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 outlier scores.

Return type Numpy array of shape (n_samples,)

pyod.models.feature_bagging module

Feature bagging detector

class pyod.models.feature_bagging.**FeatureBagging**(*base_estimator=None*, *n_estimators=10*, *contamination=0.1*, *max_features=1.0*, *bootstrap_features=False*, *check_estimator=True*, *n_jobs=1*, *random_state=None*, *combination='average'*, *verbose=0*, *estimator_params=None*)

Bases: *pyod.models.base.BaseDetector*

A feature bagging detector is a meta estimator that fits a number of base detectors on various sub-samples of the dataset and use averaging or other combination methods to improve the predictive accuracy and control over-fitting.

The sub-sample size is always the same as the original input sample size but the features are randomly sampled from half of the features to all features.

By default, LOF is used as the base estimator. However, any estimator could be used as the base estimator, such as kNN and ABOD.

Feature bagging first construct n subsamples by random selecting a subset of features, which induces the diversity of base estimators.

Finally, the prediction score is generated by averaging/taking the maximum of all base detectors. See *[BLK05]* for details.

Parameters

- **base_estimator** (*object or None, optional (default=None)*) – The base estimator to fit on random subsets of the dataset. If None, then the base estimator is a LOF detector.
- **n_estimators** (*int, optional (default=10)*) – The number of base estimators in the ensemble.
- **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.

- **max_features** (*int or float, optional (default=1.0)*) – The number of features to draw from X to train each base estimator.
  - If int, then draw *max_features* features.
  - If float, then draw *max_features * X.shape[1]* features.
- **bootstrap_features** (*bool, optional (default=False)*) – Whether features are drawn with replacement.
- **check_estimator** (*bool, optional (default=True)*) – If set to True, check whether the base estimator is consistent with sklearn standard.
- **n_jobs** (*optional (default=1)*) – The number of jobs to run in parallel for both *fit* and *predict*. If -1, then the number of jobs is set to the number of cores.
- **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*.
- **combination** (*str, optional (default='average')*) – the method of combination:
  - if 'average': take the average of all detectors
  - if 'max': take the maximum scores of all detectors
- **verbose** (*int, optional (default=0)*) – Controls the verbosity of the building process.
- **estimator_params** (*dict, optional (default=None)*) – The list of attributes to use as parameters when instantiating a new base estimator. If none are given, default parameters are used.

decision_scores_
numpy array of shape (n_samples,) – 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.

threshold_
float – 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.

labels_
int, either 0 or 1 – 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_`.

decision_function(*X*)
Predict raw anomaly score of X using the fitted detector.

The anomaly score of an input sample is computed based on different detector algorithms. For consistency, outliers are assigned with larger anomaly scores.

Parameters **X** (*numpy array of shape (n_samples, n_features)*) – The training 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 detector first and then predict whether a particular sample is an outlier or not.

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 **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,)

fit_predict_score(*X*, *y*, *scoring='roc_auc_score'*)
Fit the detector, predict on samples, and evaluate the model by predefined metrics, e.g., ROC.

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).
- **scoring** (*str, optional (default='roc_auc_score')*) – Evaluation metric:
  - 'roc_auc_score': ROC score
  - 'prc_n_score': Precision @ rank n score

Returns **score**

Return type float

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*, *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.
- **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

pyod.models.hbos module

Histogram-based Outlier Detection (HBOS)

class pyod.models.hbos.**HBOS**(*n_bins=10*, *alpha=0.1*, *tol=0.5*, *contamination=0.1*)

Bases: *pyod.models.base.BaseDetector*

Histogram- based outlier detection (HBOS) is an efficient unsupervised method. It assumes the feature independence and calculates the degree of outlyingness by building histograms. See *[BGD12]* for details.

Parameters

- **n_bins** (*int, optional (default=10)*) – The number of bins.
- **alpha** (*float in (0, 1), optional (default=0.1)*) – The regularizer for preventing overflow.
- **tol** (*float in (0, 1), optional (default=0.1)*) – The parameter to decide the flexibility while dealing the samples falling outside the bins.
- **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.

bin_edges_
numpy array of shape (n_bins + 1, n_features) – The edges of the bins.

hist_
numpy array of shape (n_bins, n_features) – The density of each histogram.

decision_scores_
numpy array of shape (n_samples,) – 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.

threshold_
float – 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.

labels_
int, either 0 or 1 – 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_`.

decision_function(*X*)
Predict raw anomaly score of X using the fitted detector.

The anomaly score of an input sample is computed based on different detector algorithms. For consistency, outliers are assigned with larger anomaly scores.

Parameters **X** (*numpy array of shape (n_samples, n_features)*) – The training 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 detector first and then predict whether a particular sample is an outlier or not.

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 **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,)

fit_predict_score(*X*, *y*, *scoring='roc_auc_score'*)
Fit the detector, predict on samples, and evaluate the model by predefined metrics, e.g., ROC.

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).
- **scoring** (*str, optional (default='roc_auc_score')*) – Evaluation metric:
  - 'roc_auc_score': ROC score
  - 'prc_n_score': Precision @ rank n score

Returns **score**

Return type float

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*, *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.
- **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

pyod.models.iforest module

IsolationForest Outlier Detector. Implemented on scikit-learn library.

class pyod.models.iforest.**IForest**(*n_estimators=100*, *max_samples='auto'*, *contamination=0.1*, *max_features=1.0*, *bootstrap=False*, *n_jobs=1*, *random_state=None*, *verbose=0*)

Bases: *pyod.models.base.BaseDetector*

Wrapper of scikit-learn Isolation Forest with more functionalities.

The IsolationForest 'isolates' observations by randomly selecting a feature and then randomly selecting a split value between the maximum and minimum values of the selected feature. See *[BLTZ08][BLTZ12]* for details.

Since recursive partitioning can be represented by a tree structure, the number of splittings required to isolate a sample is equivalent to the path length from the root node to the terminating node.

This path length, averaged over a forest of such random trees, is a measure of normality and our decision function.

Random partitioning produces noticeably shorter paths for anomalies. Hence, when a forest of random trees collectively produce shorter path lengths for particular samples, they are highly likely to be anomalies.

Parameters

- **n_estimators** (*int, optional (default=100)*) – The number of base estimators in the ensemble.
- **max_samples** (*int or float, optional (default="auto")*) – The number of samples to draw from X to train each base estimator.
  - If int, then draw *max_samples* samples.
  - If float, then draw *max_samples * X.shape[0]* samples.
  - If "auto", then *max_samples=min(256, n_samples)*.

  If max_samples is larger than the number of samples provided, all samples will be used for all trees (no sampling).
- **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.
- **max_features** (*int or float, optional (default=1.0)*) – The number of features to draw from X to train each base estimator.
  - If int, then draw *max_features* features.
  - If float, then draw *max_features * X.shape[1]* features.
- **bootstrap** (*boolean, optional (default=False)*) – If True, individual trees are fit on random subsets of the training data sampled with replacement. If False, sampling without replacement is performed.

- **n_jobs** (*integer, optional (default=1)*) – The number of jobs to run in parallel for both *fit* and *predict*. If -1, then the number of jobs is set to the number of cores.
- **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*.
- **verbose** (*int, optional (default=0)*) – Controls the verbosity of the tree building process.

estimators_
list of DecisionTreeClassifier – The collection of fitted sub-estimators.

estimators_samples_
list of arrays – The subset of drawn samples (i.e., the in-bag samples) for each base estimator.

max_samples_
integer – The actual number of samples

decision_scores_
numpy array of shape (n_samples,) – 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.

threshold_
float – 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.

labels_
int, either 0 or 1 – 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_`.

decision_function(*X*)
Predict raw anomaly score of X using the fitted detector.

The anomaly score of an input sample is computed based on different detector algorithms. For consistency, outliers are assigned with larger anomaly scores.

Parameters **X** (*numpy array of shape (n_samples, n_features)*) – The training 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 detector first and then predict whether a particular sample is an outlier or not.

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 **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,)

fit_predict_score(*X*, *y*, *scoring='roc_auc_score'*)

Fit the detector, predict on samples, and evaluate the model by predefined metrics, e.g., ROC.

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).
- **scoring** (*str, optional (default='roc_auc_score')*) – Evaluation metric:
  - 'roc_auc_score': ROC score
  - 'prc_n_score': Precision @ rank n score

Returns **score**

Return type float

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*, *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.
- **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

pyod.models.knn module

k-Nearest Neighbors Detector (kNN)

class pyod.models.knn.**KNN**(*contamination=0.1*, *n_neighbors=5*, *method='largest'*, *radius=1.0*, *algorithm='auto'*, *leaf_size=30*, *metric='minkowski'*, *p=2*, *metric_params=None*, *n_jobs=1*, ***kwargs*)

Bases: *pyod.models.base.BaseDetector*

kNN class for outlier detection. For an observation, its distance to its kth nearest neighbor could be viewed as the outlying score. It could be viewed as a way to measure the density. See *[BRRS00][BAP02]* for details.

Three kNN detectors are supported: largest: use the distance to the kth neighbor as the outlier score mean: use the average of all k neighbors as the outlier score median: use the median of the distance to k neighbors as the outlier score

Parameters

- **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.
- **n_neighbors** (*int, optional (default = 5)*) – Number of neighbors to use by default for k neighbors queries.
- **method** (*str, optional (default='largest')*) – {'largest', 'mean', 'median'}
  - 'largest': use the distance to the kth neighbor as the outlier score
  - 'mean': use the average of all k neighbors as the outlier score
  - 'median': use the median of the distance to k neighbors as the outlier score
- **radius** (*float, optional (default = 1.0)*) – Range of parameter space to use by default for *radius_neighbors* queries.
- **algorithm** (*{'auto', 'ball_tree', 'kd_tree', 'brute'}, optional*) – Algorithm used to compute the nearest neighbors:

- 'ball_tree' will use BallTree
- 'kd_tree' will use KDTree
- 'brute' will use a brute-force search.
- 'auto' will attempt to decide the most appropriate algorithm based on the values passed to *fit()* method.

Note: fitting on sparse input will override the setting of this parameter, using brute force.

- **leaf_size** (*int, optional (default = 30)*) – Leaf size passed to BallTree or KDTree. This can affect the speed of the construction and query, as well as the memory required to store the tree. The optimal value depends on the nature of the problem.
- **metric** (*string or callable, default 'minkowski'*) – metric to use for distance computation. Any metric from scikit-learn or scipy.spatial.distance can be used.

  If metric is a callable function, it is called on each pair of instances (rows) and the resulting value recorded. The callable should take two arrays as input and return one value indicating the distance between them. This works for Scipy's metrics, but is less efficient than passing the metric name as a string.

  Distance matrices are not supported.

  Valid values for metric are:

  - from scikit-learn: ['cityblock', 'cosine', 'euclidean', 'l1', 'l2', 'manhattan']
  - from scipy.spatial.distance: ['braycurtis', 'canberra', 'chebyshev', 'correlation', 'dice', 'hamming', 'jaccard', 'kulsinski', 'mahalanobis', 'matching', 'minkowski', 'rogerstanimoto', 'russellrao', 'seuclidean', 'sokalmichener', 'sokalsneath', 'sqeuclidean', 'yule']

  See the documentation for scipy.spatial.distance for details on these metrics.
- **p** (*integer, optional (default = 2)*) – Parameter for the Minkowski metric from sklearn.metrics.pairwise.pairwise_distances. When p = 1, this is equivalent to using manhattan_distance (l1), and euclidean_distance (l2) for p = 2. For arbitrary p, minkowski_distance (l_p) is used. See http://scikit-learn.org/stable/modules/generated/sklearn.metrics.pairwise.pairwise_distances
- **metric_params** (*dict, optional (default = None)*) – Additional keyword arguments for the metric function.
- **n_jobs** (*int, optional (default = 1)*) – The number of parallel jobs to run for neighbors search. If `-1`, then the number of jobs is set to the number of CPU cores. Affects only kneighbors and kneighbors_graph methods.

decision_scores_
: *numpy array of shape (n_samples,)* – 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.

threshold_
: *float* – 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.

labels_
: *int, either 0 or 1* – 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_`.

decision_function(*X*)
: Predict raw anomaly score of X using the fitted detector.

The anomaly score of an input sample is computed based on different detector algorithms. For consistency, outliers are assigned with larger anomaly scores.

Parameters **X** (*numpy array of shape (n_samples, n_features)*) – The training 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 detector first and then predict whether a particular sample is an outlier or not.

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 **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,)

fit_predict_score(*X*, *y*, *scoring='roc_auc_score'*)
Fit the detector, predict on samples, and evaluate the model by predefined metrics, e.g., ROC.

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).
- **scoring**(*str, optional (default='roc_auc_score')*) – Evaluation metric:
  - 'roc_auc_score': ROC score
  - 'prc_n_score': Precision @ rank n score

Returns **score**

Return type float

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*, *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.
- **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

pyod.models.lof module

Local Outlier Factor (LOF). Implemented on scikit-learn library.

class pyod.models.lof.**LOF**(*n_neighbors=20*, *algorithm='auto'*, *leaf_size=30*, *metric='minkowski'*, *p=2*, *metric_params=None*, *contamination=0.1*, *n_jobs=1*)

Bases: *pyod.models.base.BaseDetector*

Wrapper of scikit-learn LOF Class with more functionalities. Unsupervised Outlier Detection using Local Outlier Factor (LOF).

The anomaly score of each sample is called Local Outlier Factor. It measures the local deviation of density of a given sample with respect to its neighbors. It is local in that the anomaly score depends on how isolated the object is with respect to the surrounding neighborhood. More precisely, locality is given by k-nearest neighbors, whose distance is used to estimate the local density. By comparing the local density of a sample to the local densities of its neighbors, one can identify samples that have a substantially lower density than their neighbors. These are considered outliers. See *[BBKNS00]* for details.

Parameters

- **n_neighbors** (*int, optional (default=20)*) – Number of neighbors to use by default for *kneighbors* queries. If n_neighbors is larger than the number of samples provided, all samples will be used.
- **algorithm** (*{'auto', 'ball_tree', 'kd_tree', 'brute'}, optional*) – Algorithm used to compute the nearest neighbors:
  - 'ball_tree' will use BallTree
  - 'kd_tree' will use KDTree
  - 'brute' will use a brute-force search.
  - 'auto' will attempt to decide the most appropriate algorithm based on the values passed to *fit()* method.

  Note: fitting on sparse input will override the setting of this parameter, using brute force.
- **leaf_size** (*int, optional (default=30)*) – Leaf size passed to *BallTree* or *KDTree*. This can affect the speed of the construction and query, as well as the memory required to store the tree. The optimal value depends on the nature of the problem.
- **metric** (*string or callable, default 'minkowski'*) – metric used for the distance computation. Any metric from scikit-learn or scipy.spatial.distance can be used.

  If 'precomputed', the training input X is expected to be a distance matrix.

  If metric is a callable function, it is called on each pair of instances (rows) and the resulting value recorded. The callable should take two arrays as input and return one value indicating the distance between them. This works for Scipy's metrics, but is less efficient than passing the metric name as a string.

  Valid values for metric are:
  - from scikit-learn: ['cityblock', 'cosine', 'euclidean', 'l1', 'l2', 'manhattan']
  - from scipy.spatial.distance: ['braycurtis', 'canberra', 'chebyshev', 'correlation', 'dice', 'hamming', 'jaccard', 'kulsinski', 'mahalanobis', 'matching', 'minkowski', 'rogerstanimoto', 'russellrao', 'seuclidean', 'sokalmichener', 'sokalsneath', 'sqeuclidean', 'yule']

  See the documentation for scipy.spatial.distance for details on these metrics: http://docs.scipy.org/doc/scipy/reference/spatial.distance.html
- **p** (*integer, optional (default = 2)*) – Parameter for the Minkowski metric from sklearn.metrics.pairwise.pairwise_distances. When p = 1, this is equivalent to using manhattan_distance (l1), and euclidean_distance (l2) for p = 2. For arbitrary p, minkowski_distance (l_p) is used. See http://scikit-learn.org/stable/modules/generated/sklearn.metrics.pairwise.pairwise_distances
- **metric_params** (*dict, optional (default = None)*) – Additional keyword arguments for the metric function.

- **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. When fitting this is used to define the threshold on the decision function.
- **n_jobs** (*int, optional (default = 1)*) – The number of parallel jobs to run for neighbors search. If `-1`, then the number of jobs is set to the number of CPU cores. Affects only kneighbors and kneighbors_graph methods.

n_neighbors_
int – The actual number of neighbors used for *kneighbors* queries.

decision_scores_
numpy array of shape (n_samples,) – 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.

threshold_
float – 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.

labels_
int, either 0 or 1 – 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_`.

decision_function(*X*)
Predict raw anomaly score of X using the fitted detector.

The anomaly score of an input sample is computed based on different detector algorithms. For consistency, outliers are assigned with larger anomaly scores.

Parameters **X** (*numpy array of shape (n_samples, n_features)*) – The training 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 detector first and then predict whether a particular sample is an outlier or not.

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 **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,)

fit_predict_score(*X*, *y*, *scoring='roc_auc_score'*)
Fit the detector, predict on samples, and evaluate the model by predefined metrics, e.g., ROC.

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).
- **scoring** (*str, optional (default='roc_auc_score')*) – Evaluation metric:
  - 'roc_auc_score': ROC score
  - 'prc_n_score': Precision @ rank n score

Returns **score**

Return type float

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*, *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.
- **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

pyod.models.loci module

Local Correlation Integral (LOCI). Part of the codes are adapted from https://github.com/Cloudy10/loci

class pyod.models.loci.**LOCI**(*contamination=0.1*, *alpha=0.5*, *k=3*)
Bases: *pyod.models.base.BaseDetector*

Local Correlation Integral.

LOCI is highly effective for detecting outliers and groups of outliers (a.k.a.micro-clusters), which offers the following advantages and novelties: (a) It provides an automatic, data-dictated cut-off to determine whether a point is an outlier—in contrast, previous methods force users to pick cut-offs, without any hints as to what cut-off value is best for a given dataset. (b) It can provide a LOCI plot for each point; this plot summarizes a wealth of information about the data in the vicinity of the point, determining clusters, micro-clusters, their diameters and their inter-cluster distances. None of the existing outlier-detection methods can match this feature, because they output only a single number for each point: its outlierness score.(c) It can be computed as quickly as the best previous methods Read more in the *[BPKGF03]*.

Parameters

- **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.
- **alpha** (*int, default = 0.5*) – The neighbourhood parameter measures how large of a neighbourhood should be considered "local".
- **k** (*int, default = 3*) – An outlier cutoff threshold for determine whether or not a point should be considered an outlier.

decision_scores_
numpy array of shape (n_samples,) – 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.

threshold_
float – 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.

labels_
int, either 0 or 1 – 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_`.

Examples

```
>>> from pyod.models.loci import LOCI
>>> from pyod.utils.data import generate_data
>>> n_train = 50
>>> n_test = 50
>>> contamination = 0.1
>>> X_train, y_train, X_test, y_test = generate_data(
...     n_train=n_train, n_test=n_test,
...     contamination=contamination, random_state=42)
>>>
>>> clf = LOCI()
>>> clf.fit(X_train)
>>> print(clf.decision_scores_)
```

decision_function(*X*)
Predict raw anomaly score of X using the fitted detector.

The anomaly score of an input sample is computed based on different detector algorithms. For consistency, outliers are assigned with larger anomaly scores.

Parameters **X** (*numpy array of shape (n_samples, n_features)*) – The training 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 the model using X as training data.

Parameters **X** (*array, shape (n_samples, n_features)*) – Training data.

Returns **self**

Return type object

fit_predict(*X*, *y=None*)
Fit detector first and then predict whether a particular sample is an outlier or not.

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 **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,)

fit_predict_score(*X*, *y*, *scoring='roc_auc_score'*)
Fit the detector, predict on samples, and evaluate the model by predefined metrics, e.g., ROC.

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).
- **scoring** (*str, optional (default='roc_auc_score')*) – Evaluation metric:
  - 'roc_auc_score': ROC score
  - 'prc_n_score': Precision @ rank n score

Returns **score**

Return type float

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*, *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.
- **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

pyod.models.lscp module

Locally Selective Combination of Parallel Outlier Ensembles (LSCP) Adapted from the original implementation:

class pyod.models.lscp.**LSCP**(*detector_list*, *local_region_size=30*, *local_max_features=1.0*, *n_bins=10*, *random_state=None*, *contamination=0.1*)

Bases: *pyod.models.base.BaseDetector*

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 *[BZHNL19]* for details.

Parameters

- **detector_list** (*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).

decision_scores_

numpy array of shape (n_samples,) – 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.

threshold_
float – 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.

labels_
int, either 0 or 1 – 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_`.

Examples

```
>>> from pyod.utils.data import generate_data
>>> from pyod.utils.utility import standardizer
>>> from pyod.models.lscp import LSCP
>>> from pyod.models.lof import LOF
>>> X_train, y_train, X_test, y_test = generate_data(
...     n_train=50, n_test=50,
...     contamination=0.1, random_state=42)
>>> X_train, X_test = standardizer(X_train, X_test)
>>> detector_list = [LOF(), LOF()]
>>> clf = LSCP(detector_list)
>>> clf.fit(X_train)
>>> print(clf.decision_scores_)
```

decision_function(*X*)
Predict raw anomaly score of X using the fitted detector.

The anomaly score of an input sample is computed based on different detector algorithms. For consistency, outliers are assigned with larger anomaly scores.

Parameters **X** (*numpy array of shape (n_samples, n_features)*) – The training 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 detector first and then predict whether a particular sample is an outlier or not.

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 **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,)

fit_predict_score(*X*, *y*, *scoring='roc_auc_score'*)
Fit the detector, predict on samples, and evaluate the model by predefined metrics, e.g., ROC.

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).
- **scoring** (*str, optional (default='roc_auc_score')*) – Evaluation metric:
  - 'roc_auc_score': ROC score
  - 'prc_n_score': Precision @ rank n score

Returns **score**

Return type float

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*, *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.
- **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

pyod.models.mcd module

Outlier Detection with Minimum Covariance Determinant (MCD)

class pyod.models.mcd.**MCD**(*contamination=0.1*, *store_precision=True*, *assume_centered=False*, *support_fraction=None*, *random_state=None*)

Bases: *pyod.models.base.BaseDetector*

Detecting outliers in a Gaussian distributed dataset using Minimum Covariance Determinant (MCD): robust estimator of covariance.

The Minimum Covariance Determinant covariance estimator is to be applied on Gaussian-distributed data, but could still be relevant on data drawn from a unimodal, symmetric distribution. It is not meant to be used with multi-modal data (the algorithm used to fit a MinCovDet object is likely to fail in such a case). One should consider projection pursuit methods to deal with multi-modal datasets.

First fit a minimum covariance determinant model and then compute the Mahalanobis distance as the outlier degree of the data

See *[BRD99][BHR04]* for details.

Parameters

- **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.
- **store_precision** (*bool*) – Specify if the estimated precision is stored.
- **assume_centered** (*Boolean*) – If True, the support of the robust location and the covariance estimates is computed, and a covariance estimate is recomputed from it, without centering the data. Useful to work with data whose mean is significantly equal to zero but is not exactly zero. If False, the robust location and covariance are directly computed with the FastMCD algorithm without additional treatment.
- **support_fraction** (*float, 0 < support_fraction < 1*) – The proportion of points to be included in the support of the raw MCD estimate. Default is None, which implies that the minimum value of support_fraction will be used within the algorithm: [n_sample + n_features + 1] / 2
- **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*.

raw_location_
array-like, shape (n_features,) – The raw robust estimated location before correction and re-weighting.

raw_covariance_
array-like, shape (n_features, n_features) – The raw robust estimated covariance before correction and re-weighting.

raw_support_
array-like, shape (n_samples,) – A mask of the observations that have been used to compute the raw robust estimates of location and shape, before correction and re-weighting.

location_
array-like, shape (n_features,) – Estimated robust location

covariance_
array-like, shape (n_features, n_features) – Estimated robust covariance matrix

precision_
array-like, shape (n_features, n_features) – Estimated pseudo inverse matrix. (stored only if store_precision is True)

support_
array-like, shape (n_samples,) – A mask of the observations that have been used to compute the robust estimates of location and shape.

decision_scores_
numpy array of shape (n_samples,) – 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. Mahalanobis distances of the training set (on which *:meth:'fit* is called) observations.

threshold_
float – 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.

labels_
int, either 0 or 1 – 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_`.

decision_function(*X*)
Predict raw anomaly score of X using the fitted detector.

The anomaly score of an input sample is computed based on different detector algorithms. For consistency, outliers are assigned with larger anomaly scores.

Parameters **X** (*numpy array of shape (n_samples, n_features)*) – The training 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 detector first and then predict whether a particular sample is an outlier or not.

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 **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,)

fit_predict_score(*X*, *y*, *scoring='roc_auc_score'*)
Fit the detector, predict on samples, and evaluate the model by predefined metrics, e.g., ROC.

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).
- **scoring** (*str, optional (default='roc_auc_score')*) – Evaluation metric:
  - 'roc_auc_score': ROC score
  - 'prc_n_score': Precision @ rank n score

Returns **score**

Return type float

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*, *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.
- **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

pyod.models.mo_gaal module

Multiple-Objective Generative Adversarial Active Learning. Part of the codes are adapted from https://github.com/leibinghe/GAAL-based-outlier-detection

class pyod.models.mo_gaal.**MO_GAAL**(*k*, *stop_epochs=20*, *lr_d=0.01*, *lr_g=0.0001*, *decay=1e-06*, *momentum=0.9*, *contamination=0.1*)

Bases: *pyod.models.base.BaseDetector*

Multi-Objective Generative Adversarial Active Learning.

MO_GAAL directly generates informative potential outliers to assist the classifier in describing a boundary that can separate outliers from normal data effectively. Moreover, to prevent the generator from falling into the mode collapsing problem, the network structure of SO-GAAL is expanded from a single generator (SO-GAAL) to multiple generators with different objectives (MO-GAAL) to generate a reasonable reference distribution for the whole dataset. Read more in the *[BLLZ+18]*.

Parameters

- **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.
- **stop_epochs** (*int, default = 20*) – The number of epochs of training.
- **lr_d** (*float, default = 0.01*) – The learn rate of the discriminator.
- **lr_g** (*float, default = 0.0001*) – The learn rate of the generator.
- **decay** (*int, default = 1e-6*) – The decay parameter for SGD.
- **momentum** (*float, default = 0.9*) – The momentum parameter for SGD.

decision_scores_
numpy array of shape (n_samples,) – 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.

threshold_
float – 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.

labels_
int, either 0 or 1 – 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_`.

Examples

```
>>> from pyod.models.mo_gaal import MO_GAAL
>>> from pyod.utils.data import generate_data
>>> n_train = 20000
>>> n_test = 2000
>>> contamination = 0.1
>>> n_features = 300
>>> X_train, y_train, X_test, y_test = generate_data(
...     n_train=n_train, n_test=n_test, n_features=n_features,
...     contamination=contamination, random_state=42)
>>>
>>> clf = MO_GAAL()
>>> clf.fit(X_train)
>>> print(clf.decision_scores_)
```

decision_function(*X*)
Predict raw anomaly score of X using the fitted detector.

The anomaly score of an input sample is computed based on different detector algorithms. For consistency, outliers are assigned with larger anomaly scores.

Parameters **X** (*numpy array of shape (n_samples, n_features)*) – The training 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 detector first and then predict whether a particular sample is an outlier or not.

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 **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,)

fit_predict_score(*X*, *y*, *scoring='roc_auc_score'*)

Fit the detector, predict on samples, and evaluate the model by predefined metrics, e.g., ROC.

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).
- **scoring** (*str, optional (default='roc_auc_score')*) – Evaluation metric:
  - 'roc_auc_score': ROC score
  - 'prc_n_score': Precision @ rank n score

Returns **score**

Return type float

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*, *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.
- **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

pyod.models.ocsvm module

One-class SVM detector. Implemented on scikit-learn library.

class pyod.models.ocsvm.**OCSVM**(*kernel='rbf'*, *degree=3*, *gamma='auto'*, *coef0=0.0*, *tol=0.001*, *nu=0.5*, *shrinking=True*, *cache_size=200*, *verbose=False*, *max_iter=-1*, *contamination=0.1*, *random_state=None*)

Bases: *pyod.models.base.BaseDetector*

Wrapper of scikit-learn one-class SVM Class with more functionalities. Unsupervised Outlier Detection.

Estimate the support of a high-dimensional distribution.

The implementation is based on libsvm. See http://scikit-learn.org/stable/modules/svm.html#svm-outlier-detection and *[BMP03]*.

Parameters

- **kernel** (*string, optional (default='rbf')*) – Specifies the kernel type to be used in the algorithm. It must be one of 'linear', 'poly', 'rbf', 'sigmoid', 'precomputed' or a callable. If none is given, 'rbf' will be used. If a callable is given it is used to precompute the kernel matrix.
- **nu** (*float, optional*) – An upper bound on the fraction of training errors and a lower bound of the fraction of support vectors. Should be in the interval (0, 1]. By default 0.5 will be taken.
- **degree** (*int, optional (default=3)*) – Degree of the polynomial kernel function ('poly'). Ignored by all other kernels.
- **gamma** (*float, optional (default='auto')*) – Kernel coefficient for 'rbf', 'poly' and 'sigmoid'. If gamma is 'auto' then 1/n_features will be used instead.
- **coef0** (*float, optional (default=0.0)*) – Independent term in kernel function. It is only significant in 'poly' and 'sigmoid'.
- **tol** (*float, optional*) – Tolerance for stopping criterion.
- **shrinking** (*boolean, optional*) – Whether to use the shrinking heuristic.

- **cache_size** (*float, optional*) – Specify the size of the kernel cache (in MB).
- **verbose** (*bool, default: False*) – Enable verbose output. Note that this setting takes advantage of a per-process runtime setting in libsvm that, if enabled, may not work properly in a multithreaded context.
- **max_iter** (*int, optional (default=-1)*) – Hard limit on iterations within solver, or -1 for no limit.
- **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.
- **random_state** (*int, RandomState instance or None, optional (default=None)*) – The seed of the pseudo random number generator to use when shuffling the data. 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*.

support_
array-like, shape = [n_SV] – Indices of support vectors.

support_vectors_
array-like, shape = [nSV, n_features] – Support vectors.

dual_coef_
array, shape = [1, n_SV] – Coefficients of the support vectors in the decision function.

coef_
array, shape = [1, n_features] – Weights assigned to the features (coefficients in the primal problem). This is only available in the case of a linear kernel.

*coef_* is readonly property derived from *dual_coef_* and *support_vectors_*

intercept_
array, shape = [1,] – Constant in the decision function.

decision_scores_
numpy array of shape (n_samples,) – 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.

threshold_
float – 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.

labels_
int, either 0 or 1 – 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_`.

decision_function(*X*)
Predict raw anomaly score of X using the fitted detector.

The anomaly score of an input sample is computed based on different detector algorithms. For consistency, outliers are assigned with larger anomaly scores.

Parameters **X** (*numpy array of shape (n_samples, n_features)*) – The training 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*, *sample_weight=None*, ***params*)
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 detector first and then predict whether a particular sample is an outlier or not.

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 **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,)

fit_predict_score(*X*, *y*, *scoring='roc_auc_score'*)
Fit the detector, predict on samples, and evaluate the model by predefined metrics, e.g., ROC.

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).
- **scoring** (*str, optional (default='roc_auc_score')*) – Evaluation metric:
  - 'roc_auc_score': ROC score
  - 'prc_n_score': Precision @ rank n score

Returns **score**

Return type float

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*, *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.
- **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

pyod.models.pca module

Principal Component Analysis (PCA) Outlier Detector

class pyod.models.pca.**PCA**(*n_components=None*, *n_selected_components=None*, *contamination=0.1*, *copy=True*, *whiten=False*, *svd_solver='auto'*, *tol=0.0*, *iterated_power='auto'*, *random_state=None*, *weighted=True*, *standardization=True*)

Bases: *pyod.models.base.BaseDetector*

Principal component analysis (PCA) can be used in detecting outliers. PCA is a linear dimensionality reduction using Singular Value Decomposition of the data to project it to a lower dimensional space.

In this procedure, covariance matrix of the data can be decomposed to orthogonal vectors, called eigenvectors, associated with eigenvalues. The eigenvectors with high eigenvalues capture most of the variance in the data.

Therefore, a low dimensional hyperplane constructed by k eigenvectors can capture most of the variance in the data. However, outliers are different from normal data points, which is more obvious on the hyperplane constructed by the eigenvectors with small eigenvalues.

Therefore, outlier scores can be obtained as the sum of the projected distance of a sample on all eigenvectors. See *[BSCSC03][BAgg15]* for details.

Score(X) = Sum of weighted euclidean distance between each sample to the hyperplane constructed by the selected eigenvectors

Parameters

- **n_components** (*int, float, None or string*) – Number of components to keep. if n_components is not set all components are kept:

```
n_components == min(n_samples, n_features)
```

  if n_components == 'mle' and svd_solver == 'full', Minka's MLE is used to guess the dimension if `0 < n_components < 1` and svd_solver == 'full', select the number of components such that the amount of variance that needs to be explained is greater than the percentage specified by n_components n_components cannot be equal to n_features for svd_solver == 'arpack'.

- **n_selected_components** (*int, optional (default=None)*) – Number of selected principal components for calculating the outlier scores. It is not necessarily equal to the total number of the principal components. If not set, use all principal components.
- **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.
- **copy** (*bool (default True)*) – If False, data passed to fit are overwritten and running fit(X).transform(X) will not yield the expected results, use fit_transform(X) instead.
- **whiten** (*bool, optional (default False)*) – When True (False by default) the *components_* vectors are multiplied by the square root of n_samples and then divided by the singular values to ensure uncorrelated outputs with unit component-wise variances.

  Whitening will remove some information from the transformed signal (the relative variance scales of the components) but can sometime improve the predictive accuracy of the downstream estimators by making their data respect some hard-wired assumptions.

- **svd_solver** (*string {'auto', 'full', 'arpack', 'randomized'}*) –

  auto : the solver is selected by a default policy based on *X.shape* and *n_components*: if the input data is larger than 500x500 and the number of components to extract is lower than 80% of the smallest dimension of the data, then the more efficient 'randomized' method is enabled. Otherwise the exact full SVD is computed and optionally truncated afterwards.

  full : run exact full SVD calling the standard LAPACK solver via *scipy.linalg.svd* and select the components by postprocessing

  arpack : run SVD truncated to n_components calling ARPACK solver via *scipy.sparse.linalg.svds*. It requires strictly 0 < n_components < X.shape[1]

  randomized : run randomized SVD by the method of Halko et al.

- **tol** (*float >= 0, optional (default .0)*) – Tolerance for singular values computed by svd_solver == 'arpack'.
- **iterated_power** (*int >= 0, or 'auto', (default 'auto')*) – Number of iterations for the power method computed by svd_solver == 'randomized'.
- **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*. Used when `svd_solver` == 'arpack' or 'randomized'.

- **weighted** (*bool, optional (default=True)*) – If True, the eigenvalues are used in score computation. The eigenvectors with samll eigenvalues comes with more importance in outlier score calculation.
- **standardization** (*bool, optional (default=True)*) – If True, perform standardization first to convert data to zero mean and unit variance. See http://scikit-learn.org/stable/auto_examples/preprocessing/plot_scaling_importance.html

components_
array, shape (n_components, n_features) – Principal axes in feature space, representing the directions of maximum variance in the data. The components are sorted by `explained_variance_`.

explained_variance_
array, shape (n_components,) – The amount of variance explained by each of the selected components.

Equal to n_components largest eigenvalues of the covariance matrix of X.

explained_variance_ratio_
array, shape (n_components,) – Percentage of variance explained by each of the selected components.

If `n_components` is not set then all components are stored and the sum of explained variances is equal to 1.0.

singular_values_
array, shape (n_components,) – The singular values corresponding to each of the selected components. The singular values are equal to the 2-norms of the `n_components` variables in the lower-dimensional space.

mean_
array, shape (n_features,) – Per-feature empirical mean, estimated from the training set.

Equal to *X.mean(axis=0)*.

n_components_
int – The estimated number of components. When n_components is set to 'mle' or a number between 0 and 1 (with svd_solver == 'full') this number is estimated from input data. Otherwise it equals the parameter n_components, or n_features if n_components is None.

noise_variance_
float – The estimated noise covariance following the Probabilistic PCA model from Tipping and Bishop 1999. See "Pattern Recognition and Machine Learning" by C. Bishop, 12.2.1 p. 574 or http://www.miketipping.com/papers/met-mppca.pdf. It is required to computed the estimated data covariance and score samples.

Equal to the average of (min(n_features, n_samples) - n_components) smallest eigenvalues of the covariance matrix of X.

decision_scores_
numpy array of shape (n_samples,) – 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.

threshold_
float – 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.

labels_
int, either 0 or 1 – 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_`.

decision_function(*X*)
Predict raw anomaly score of X using the fitted detector.

The anomaly score of an input sample is computed based on different detector algorithms. For consistency, outliers are assigned with larger anomaly scores.

Parameters **X** (*numpy array of shape (n_samples, n_features)*) – The training 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,)

explained_variance_
The amount of variance explained by each of the selected components.

Equal to n_components largest eigenvalues of the covariance matrix of X.

Decorator for scikit-learn PCA attributes.

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 detector first and then predict whether a particular sample is an outlier or not.

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 **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,)

fit_predict_score(*X*, *y*, *scoring='roc_auc_score'*)
Fit the detector, predict on samples, and evaluate the model by predefined metrics, e.g., ROC.

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).
- **scoring** (*str, optional (default='roc_auc_score')*) – Evaluation metric:
  - 'roc_auc_score': ROC score
  - 'prc_n_score': Precision @ rank n score

Returns **score**

Return type float

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

noise_variance_
The estimated noise covariance following the Probabilistic PCA model from Tipping and Bishop 1999. See "Pattern Recognition and Machine Learning" by C. Bishop, 12.2.1 p. 574 or http://www.miketipping.com/papers/met-mppca.pdf. It is required to computed the estimated data covariance and score samples.

Equal to the average of (min(n_features, n_samples) - n_components) smallest eigenvalues of the covariance matrix of X.

Decorator for scikit-learn PCA attributes.

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*, *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.
- **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

pyod.models.so_gaal module

Single-Objective Generative Adversarial Active Learning. Part of the codes are adapted from https://github.com/leibinghe/GAAL-based-outlier-detection

class pyod.models.so_gaal.**SO_GAAL**(*stop_epochs=20*, *lr_d=0.01*, *lr_g=0.0001*, *decay=1e-06*, *momentum=0.9*, *contamination=0.1*)

Bases: *pyod.models.base.BaseDetector*

Single-Objective Generative Adversarial Active Learning.

SO-GAAL directly generates informative potential outliers to assist the classifier in describing a boundary that can separate outliers from normal data effectively. Moreover, to prevent the generator from falling into the mode collapsing problem, the network structure of SO-GAAL is expanded from a single generator (SO-GAAL) to multiple generators with different objectives (MO-GAAL) to generate a reasonable reference distribution for the whole dataset. Read more in the *[BLLZ+18]*.

Parameters

- **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.
- **stop_epochs** (*int, default = 20*) – The number of epochs of training.
- **lr_d** (*float, default = 0.01*) – The learn rate of the discriminator.
- **lr_g** (*float, default = 0.0001*) – The learn rate of the generator.
- **decay** (*int, default = 1e-6*) – The decay parameter for SGD.
- **momentum** (*float, default = 0.9*) – The momentum parameter for SGD.

decision_scores_
numpy array of shape (n_samples,) – 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.

threshold_
float – 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.

labels_
int, either 0 or 1 – 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_`.

Examples

```
>>> from pyod.models.so_gaal import SO_GAAL
>>> from pyod.utils.data import generate_data
>>> n_train = 10000
>>> n_test = 5000
```

(continues on next page)

(continued from previous page)

```
>>> contamination = 0.1
>>> n_features = 300
>>> X_train, y_train, X_test, y_test = generate_data(
...     n_train=n_train, n_test=n_test, n_features=n_features,
...     contamination=contamination, random_state=42)
>>> clf = SO_GAAL()
>>> clf.fit(X_train)
>>> print(clf.decision_scores_)
```

decision_function(*X*)
Predict raw anomaly score of X using the fitted detector.

The anomaly score of an input sample is computed based on different detector algorithms. For consistency, outliers are assigned with larger anomaly scores.

Parameters **X** (*numpy array of shape (n_samples, n_features)*) – The training 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 detector first and then predict whether a particular sample is an outlier or not.

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 **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,)

fit_predict_score(*X*, *y*, *scoring='roc_auc_score'*)
Fit the detector, predict on samples, and evaluate the model by predefined metrics, e.g., ROC.

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).
- **scoring** (*str, optional (default='roc_auc_score')*) – Evaluation metric:

- 'roc_auc_score': ROC score
- 'prc_n_score': Precision @ rank n score

Returns **score**

Return type float

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*, *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.
- **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

pyod.models.sos module

Stochastic Outlier Selection (SOS). Part of the codes are adapted from https://github.com/jeroenjanssens/scikit-sos

class pyod.models.sos.**SOS**(*contamination=0.1*, *perplexity=30*, *metric='euclidean'*, *eps=1e-05*)

Bases: *pyod.models.base.BaseDetector*

Stochastic Outlier Selection.

SOS employs the concept of affinity to quantify the relationship from one data point to another data point. Affinity is proportional to the similarity between two data points. So, a data point has little affinity with a dissimilar data point. A data point is selected as an outlier when all the other data points have insufficient affinity with it. Read more in the *[BJHuszarPvdH12]*.

Parameters

- **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.
- **perplexity** (*int*) – The perplexity parameter is a smooth measure the effective number of neighbours
- **metric** (*str, default 'euclidean'*) – Metric used for the distance computation. Any metric from scipy.spatial.distance can be used.

  Valid values for metric are:

  – 'euclidean'

  – from scipy.spatial.distance: ['braycurtis', 'canberra', 'chebyshev', 'correlation', 'dice', 'hamming', 'jaccard', 'kulsinski', 'mahalanobis', 'matching', 'minkowski', 'rogerstanimoto', 'russellrao', 'seuclidean', 'sokalmichener', 'sokalsneath', 'sqeuclidean', 'yule']

  See the documentation for scipy.spatial.distance for details on these metrics: http://docs.scipy.org/doc/scipy/reference/spatial.distance.html
- **eps** (*float, optional (default = 1e-5)*) – Tolerance threshold for floating point errors.

decision_scores_

numpy array of shape (n_samples,) – 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.

threshold_

float – 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.

labels_

int, either 0 or 1 – 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_`.

Examples

```
>>> from pyod.models.sos import SOS
>>> from pyod.utils.data import generate_data
>>> n_train = 50
>>> n_test = 50
>>> contamination = 0.1
```

(continues on next page)

(continued from previous page)

```
>>> X_train, y_train, X_test, y_test = generate_data(
...     n_train=n_train, n_test=n_test,
...     contamination=contamination, random_state=42)
>>>
>>> clf = SOS()
>>> clf.fit(X_train)
>>> print(clf.decision_scores_)
```

decision_function(*X*)
Predict raw anomaly score of X using the fitted detector.

The anomaly score of an input sample is computed based on different detector algorithms. For consistency, outliers are assigned with larger anomaly scores.

Parameters **X** (*numpy array of shape (n_samples, n_features)*) – The training 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 detector first and then predict whether a particular sample is an outlier or not.

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 **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,)

fit_predict_score(*X*, *y*, *scoring='roc_auc_score'*)
Fit the detector, predict on samples, and evaluate the model by predefined metrics, e.g., ROC.

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).
- **scoring** (*str, optional (default='roc_auc_score')*) – Evaluation metric:

- 'roc_auc_score': ROC score
- 'prc_n_score': Precision @ rank n score

Returns **score**

Return type float

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*, *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.
- **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

pyod.models.xgbod module

XGBOD: Improving Supervised Outlier Detection with Unsupervised Representation Learning. A semi-supervised outlier detection framework.

class pyod.models.xgbod.**XGBOD**(*estimator_list=None*, *standardization_flag_list=None*, *max_depth=3*, *learning_rate=0.1*, *n_estimators=100*, *silent=True*, *objective='binary:logistic'*, *booster='gbtree'*, *n_jobs=1*, *nthread=None*, *gamma=0*, *min_child_weight=1*, *max_delta_step=0*, *subsample=1*, *colsample_bytree=1*, *colsample_bylevel=1*, *reg_alpha=0*, *reg_lambda=1*, *scale_pos_weight=1*, *base_score=0.5*, *random_state=0*, *missing=None*, ***kwargs*)

Bases: *pyod.models.base.BaseDetector*

XGBOD class for outlier detection. It first use the passed in unsupervised outlier detectors to extract richer representation of the data and then concatenate the newly generated features to the original feature for constructing the augmented feature space. An XGBoost classifier is then applied on this augmented feature space. Read more in the *[BZH18]*.

Parameters

- **estimator_list** (*list, optional (default=None)*) – The list of pyod detectors passed in for unsupervised learning
- **standardization_flag_list** (*list, optional (default=None)*) – The list of boolean flags for indicating whether to take standardization for each detector.
- **max_depth** (*int*) – Maximum tree depth for base learners.
- **learning_rate** (*float*) – Boosting learning rate (xgb's "eta")
- **n_estimators** (*int*) – Number of boosted trees to fit.
- **silent** (*boolean*) – Whether to print messages while running boosting.
- **objective** (*string or callable*) – Specify the learning task and the corresponding learning objective or a custom objective function to be used (see note below).
- **booster** (*string*) – Specify which booster to use: gbtree, gblinear or dart.
- **n_jobs** (*int*) – Number of parallel threads used to run xgboost. (replaces `nthread`)
- **gamma** (*float*) – Minimum loss reduction required to make a further partition on a leaf node of the tree.
- **min_child_weight** (*int*) – Minimum sum of instance weight(hessian) needed in a child.
- **max_delta_step** (*int*) – Maximum delta step we allow each tree's weight estimation to be.
- **subsample** (*float*) – Subsample ratio of the training instance.
- **colsample_bytree** (*float*) – Subsample ratio of columns when constructing each tree.
- **colsample_bylevel** (*float*) – Subsample ratio of columns for each split, in each level.
- **reg_alpha** (*float (xgb's alpha)*) – L1 regularization term on weights.
- **reg_lambda** (*float (xgb's lambda)*) – L2 regularization term on weights.
- **scale_pos_weight** (*float*) – Balancing of positive and negative weights.

- **base_score** – The initial prediction score of all instances, global bias.
- **random_state** (*int*) – Random number seed. (replaces seed)
- **missing** (*float, optional*) – Value in the data which needs to be present as a missing value. If None, defaults to np.nan.
- **importance_type** (*string, default "gain"*) – The feature importance type for the feature_importances_ property: either "gain", "weight", "cover", "total_gain" or "total_cover".
- ****kwargs** (*dict, optional*) – Keyword arguments for XGBoost Booster object. Full documentation of parameters can be found here: https://github.com/dmlc/xgboost/blob/master/doc/parameter.rst. Attempting to set a parameter via the constructor args and **kwargs dict simultaneously will result in a TypeError.

  Note: **kwargs is unsupported by scikit-learn. We do not guarantee that parameters passed via this argument will interact properly with scikit-learn.

n_detector_
int – The number of unsupervised of detectors used.

clf_
object – The XGBoost classifier.

decision_scores_
numpy array of shape (n_samples,) – 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.

labels_
int, either 0 or 1 – 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_.

decision_function(*X*)
Predict raw anomaly score of X using the fitted detector.

The anomaly score of an input sample is computed based on different detector algorithms. For consistency, outliers are assigned with larger anomaly scores.

Parameters **X** (*numpy array of shape (n_samples, n_features)*) – The training 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*)
Fit the model using X and y as training data.

Parameters

- **X** (*numpy array of shape (n_samples, n_features)*) – Training data.
- **y** (*numpy array of shape (n_samples,)*) – The ground truth (binary label)
  - 0 : inliers
  - 1 : outliers

Returns **self**

Return type object

fit_predict(*X*, *y*)
Fit detector first and then predict whether a particular sample is an outlier or not.

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 **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,)

fit_predict_score(*X*, *y*, *scoring='roc_auc_score'*)
Fit the detector, predict on samples, and evaluate the model by predefined metrics, e.g., ROC.

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).
- **scoring** (*str, optional (default='roc_auc_score')*) – Evaluation metric:
  - 'roc_auc_score': ROC score
  - 'prc_n_score': Precision @ rank n score

Returns **score**

Return type float

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. Calling xgboost *predict* function.

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*)
Predict the probability of a sample being outlier. Calling xgboost *predict_proba* function.

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. 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

Module contents

References

2.5.2 pyod.utils package

Submodules

pyod.utils.data module

Utility functions for manipulating data

pyod.utils.data.**check_consistent_shape**(*X_train*, *y_train*, *X_test*, *y_test*, *y_train_pred*, *y_test_pred*)

Internal shape to check input data shapes are consistent.

Parameters

- **X_train** (*numpy array of shape (n_samples, n_features)*) – The training samples.
- **y_train** (*list or array of shape (n_samples,)*) – The ground truth of training samples.
- **X_test** (*numpy array of shape (n_samples, n_features)*) – The test samples.
- **y_test** (*list or array of shape (n_samples,)*) – The ground truth of test samples.
- **y_train_pred** (*numpy array of shape (n_samples, n_features)*) – The predicted binary labels of the training samples.
- **y_test_pred** (*numpy array of shape (n_samples, n_features)*) – The predicted binary labels of the test samples.

Returns

- **X_train** (*numpy array of shape (n_samples, n_features)*) – The training samples.
- **y_train** (*list or array of shape (n_samples,)*) – The ground truth of training samples.

- **X_test** (*numpy array of shape (n_samples, n_features)*) – The test samples.
- **y_test** (*list or array of shape (n_samples,)*) – The ground truth of test samples.
- **y_train_pred** (*numpy array of shape (n_samples, n_features)*) – The predicted binary labels of the training samples.
- **y_test_pred** (*numpy array of shape (n_samples, n_features)*) – The predicted binary labels of the test samples.

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

Utility function for evaluating and printing the results for examples. Default metrics include ROC and Precision @ n

Parameters

- **clf_name** (*str*) – The name of the detector.
- **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.

pyod.utils.data.**generate_data**(*n_train=1000*, *n_test=500*, *n_features=2*, *contamination=0.1*, *train_only=False*, *offset=10*, *random_state=None*)

Utility function to generate synthesized data. Normal data is generated by a multivariate Gaussian distribution and outliers are generated by a uniform distribution.

Parameters

- **n_train** (*int, (default=1000)*) – The number of training points to generate.
- **n_test** (*int, (default=500)*) – The number of test points to generate.
- **n_features** (*int, optional (default=2)*) – The number of features (dimensions).
- **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.
- **train_only** (*bool, optional (default=False)*) – If true, generate train data only.
- **offset** (*int, optional (default=10)*) – Adjust the value range of Gaussian and Uniform.
- **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

- **X_train** (*numpy array of shape (n_train, n_features)*) – Training data.
- **y_train** (*numpy array of shape (n_train,)*) – Training ground truth.
- **X_test** (*numpy array of shape (n_test, n_features)*) – Test data.
- **y_test** (*numpy array of shape (n_test,)*) – Test ground truth.

pyod.utils.data.**get_color_codes**(*y*)

Internal function to generate color codes for inliers and outliers. Inliers (0): blue; Outlier (1): red.

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

Returns **c** – Color codes.

Return type numpy array of shape (n_samples,)

pyod.utils.data.**get_outliers_inliers**(*X*, *y*)

Internal method to separate inliers from outliers.

Parameters

- **X** (*numpy array of shape (n_samples, n_features)*) – The input samples
- **y** (*list or array of shape (n_samples,)*) – The ground truth of input samples.

Returns

- **X_outliers** (*numpy array of shape (n_samples, n_features)*) – Outliers.
- **X_inliers** (*numpy array of shape (n_samples, n_features)*) – Inliers.

pyod.utils.stat_models module

A collection of statistical models

pyod.utils.stat_models.**pairwise_distances_no_broadcast**(*X*, *Y*)

Utility function to calculate row-wise euclidean distance of two matrix. Different from pair-wise calculation, this function would not broadcast.

For instance, X and Y are both (4,3) matrices, the function would return a distance vector with shape (4,), instead of (4,4).

Parameters

- **X** (*array of shape (n_samples, n_features)*) – First input samples
- **Y** (*array of shape (n_samples, n_features)*) – Second input samples

Returns **distance** – Row-wise euclidean distance of X and Y

Return type array of shape (n_samples,)

pyod.utils.stat_models.**pearsonr_mat**(*mat*, *w=None*)

Utility function to calculate pearson matrix (row-wise).

Parameters

- **mat** (*numpy array of shape (n_samples, n_features)*) – Input matrix.
- **w** (*numpy array of shape (n_features,)*) – Weights.

Returns **pear_mat** – Row-wise pearson score matrix.

Return type numpy array of shape (n_samples, n_samples)

pyod.utils.stat_models.**wpearsonr**(*x*, *y*, *w=None*)

Utility function to calculate the weighted Pearson correlation of two samples.

See https://stats.stackexchange.com/questions/221246/such-thing-as-a-weighted-correlation for more information

Parameters

- **x** (*array, shape (n,)*) – Input x.

- **y** (*array, shape (n,)*) – Input y.
- **w** (*array, shape (n,)*) – Weights w.

Returns **scores** – Weighted Pearson Correlation between x and y.

Return type float in range of [-1,1]

pyod.utils.utility module

A set of utility functions to support outlier detection.

pyod.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,)

pyod.utils.utility.**check_detector**(*detector*)
Checks if fit and decision_function methods exist for given detector

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

pyod.utils.utility.**check_parameter**(*param*, *low=-2147483647*, *high=2147483647*, *param_name=''*, *include_left=False*, *include_right=False*)
Check if an input is within the defined range.

Parameters

- **param** (*int, float*) – The input parameter to check.
- **low** (*int, float*) – The lower bound of the range.
- **high** (*int, float*) – The higher bound of the range.
- **param_name** (*str, optional (default='')*) – The name of the parameter.
- **include_left** (*bool, optional (default=False)*) – Whether includes the lower bound (lower bound <=).
- **include_right** (*bool, optional (default=False)*) – Whether includes the higher bound (<= higher bound).

Returns **within_range** – Whether the parameter is within the range of (low, high)

Return type bool or raise errors

pyod.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,)

pyod.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,)

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

pyod.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]
```

pyod.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

pyod.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,)

pyod.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

pyod.utils.**check_parameter**(*param*, *low=-2147483647*, *high=2147483647*, *param_name=''*, *include_left=False*, *include_right=False*)

Check if an input is within the defined range.

Parameters

- **param** (*int, float*) – The input parameter to check.
- **low** (*int, float*) – The lower bound of the range.
- **high** (*int, float*) – The higher bound of the range.
- **param_name** (*str, optional (default='')*) – The name of the parameter.
- **include_left** (*bool, optional (default=False)*) – Whether includes the lower bound (lower bound <=).
- **include_right** (*bool, optional (default=False)*) – Whether includes the higher bound (<= higher bound).

Returns **within_range** – Whether the parameter is within the range of (low, high)

Return type bool or raise errors

pyod.utils.**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

pyod.utils.**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,)

pyod.utils.**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

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

pyod.utils.**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,)

pyod.utils.**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]
```

pyod.utils.**generate_data**(*n_train=1000*, *n_test=500*, *n_features=2*, *contamination=0.1*, *train_only=False*, *offset=10*, *random_state=None*)

Utility function to generate synthesized data. Normal data is generated by a multivariate Gaussian distribution and outliers are generated by a uniform distribution.

Parameters

- **n_train** (*int, (default=1000)*) – The number of training points to generate.
- **n_test** (*int, (default=500)*) – The number of test points to generate.
- **n_features** (*int, optional (default=2)*) – The number of features (dimensions).
- **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.
- **train_only** (*bool, optional (default=False)*) – If true, generate train data only.
- **offset** (*int, optional (default=10)*) – Adjust the value range of Gaussian and Uniform.
- **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

- **X_train** (*numpy array of shape (n_train, n_features)*) – Training data.
- **y_train** (*numpy array of shape (n_train,)*) – Training ground truth.
- **X_test** (*numpy array of shape (n_test, n_features)*) – Test data.
- **y_test** (*numpy array of shape (n_test,)*) – Test ground truth.

pyod.utils.**evaluate_print**(*clf_name*, *y*, *y_pred*)

Utility function for evaluating and printing the results for examples. Default metrics include ROC and Precision @ n

Parameters

- **clf_name** (*str*) – The name of the detector.
- **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.

pyod.utils.**get_color_codes**(*y*)

Internal function to generate color codes for inliers and outliers. Inliers (0): blue; Outlier (1): red.

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

Returns **c** – Color codes.

Return type numpy array of shape (n_samples,)

pyod.utils.**pairwise_distances_no_broadcast**(*X*, *Y*)

Utility function to calculate row-wise euclidean distance of two matrix. Different from pair-wise calculation, this function would not broadcast.

For instance, X and Y are both (4,3) matrices, the function would return a distance vector with shape (4,), instead of (4,4).

Parameters

- **X** (*array of shape (n_samples, n_features)*) – First input samples
- **Y** (*array of shape (n_samples, n_features)*) – Second input samples

Returns **distance** – Row-wise euclidean distance of X and Y

Return type array of shape (n_samples,)

pyod.utils.**wpearsonr**(*x*, *y*, *w=None*)

Utility function to calculate the weighted Pearson correlation of two samples.

See https://stats.stackexchange.com/questions/221246/such-thing-as-a-weighted-correlation for more information

Parameters

- **x** (*array, shape (n,)*) – Input x.
- **y** (*array, shape (n,)*) – Input y.
- **w** (*array, shape (n,)*) – Weights w.

Returns **scores** – Weighted Pearson Correlation between x and y.

Return type float in range of [-1,1]

pyod.utils.**pearsonr_mat**(*mat*, *w=None*)

Utility function to calculate pearson matrix (row-wise).

Parameters

- **mat** (*numpy array of shape (n_samples, n_features)*) – Input matrix.
- **w** (*numpy array of shape (n_features,)*) – Weights.

Returns **pear_mat** – Row-wise pearson score matrix.

Return type numpy array of shape (n_samples, n_samples)

2.5.3 Module contents

2.6 Known Issues

This is the central place to track known issues:

1. PyOD examples depend on Matplotlib, which would throw errors in conda virtual environment on mac OS. See mac_matplotlib.

2.6.1 Neural Networks

Neural-net installation related questions could be found: FAQ

SO_GAAL and MO_GAAL may only work under Python 3.5+.

2.7 Todo & Contribution Guidance

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
- Fix the bug that numba enabled function may be excluded from code coverage

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

2.7.1 How to Contribute

You are welcome to contribute to this exciting project, and a manuscript at JMLR (Track for open-source software) is under review.

If you are interested in contributing:

- Please first check Issue lists for "help wanted" tag and comment the one you are interested. We will assign the issue to you.
- Fork the master branch and add your improvement/modification/fix.
- Create a pull request and follow the pull request template PR template

To make sure the code has the same style and standard, please refer to models, such as abod.py, hbos.py, or feature bagging for example.

You are also welcome to share your ideas by opening an issue or dropping me an email at yuezhao@cs.toronto.edu :)

2.8 About us

2.8.1 Core Development Team

Yue Zhao (initialized the project in 2017): Homepage

Zain Nasrullah (joined in 2018): LinkedIn (Zain Nasrullah)

Winston (Zheng) Li (joined in 2018): LinkedIn (Winston Li)

2.8.2 Citing PyOD

If you use PyOD in a scientific publication, we would appreciate citations to the following paper:

```
@article{zhao2019pyod,
  title={PyOD: A Python Toolbox for Scalable Outlier Detection},
  author={Yue Zhao and Zain Nasrullah and Zheng Li},
  journal={arXiv preprint arXiv:1901.01588},
  year={2019},
  url={https://arxiv.org/abs/1901.01588}
}
```

It is currently under review at JMLR (machine learning open-source software track). See preprint.

CHAPTER 3

Quick Links

- genindex
- modindex
- search

References

Bibliography

[BAgg15] Charu C Aggarwal. Outlier analysis. In *Data mining*, 75–79. Springer, 2015.

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

[BAP02] Fabrizio Angiulli and Clara Pizzuti. Fast outlier detection in high dimensional spaces. In *European Conference on Principles of Data Mining and Knowledge Discovery*, 15–27. Springer, 2002.

[BBKNS00] Markus M Breunig, Hans-Peter Kriegel, Raymond T Ng, and Jörg Sander. Lof: identifying density-based local outliers. In *ACM sigmod record*, volume 29, 93–104. ACM, 2000.

[BGD12] Markus Goldstein and Andreas Dengel. Histogram-based outlier score (hbos): a fast unsupervised anomaly detection algorithm. *KI-2012: Poster and Demo Track*, pages 59–63, 2012.

[BHR04] Johanna Hardin and David M Rocke. Outlier detection in the multiple cluster setting using the minimum covariance determinant estimator. *Computational Statistics & Data Analysis*, 44(4):625–638, 2004.

[BHXD03] Zengyou He, Xiaofei Xu, and Shengchun Deng. Discovering cluster-based local outliers. *Pattern Recognition Letters*, 24(9-10):1641–1650, 2003.

[BJHuszarPvdH12] JHM Janssens, Ferenc Huszár, EO Postma, and HJ van den Herik. Stochastic outlier selection. Technical Report, Technical report TiCC TR 2012-001, Tilburg University, Tilburg Center for Cognition and Communication, Tilburg, The Netherlands, 2012.

[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.

[BKZ+08] Hans-Peter Kriegel, Arthur Zimek, and others. Angle-based outlier detection in high-dimensional data. In *Proceedings of the 14th ACM SIGKDD international conference on Knowledge discovery and data mining*, 444–452. ACM, 2008.

[BLK05] Aleksandar Lazarevic and Vipin Kumar. Feature bagging for outlier detection. In *Proceedings of the eleventh ACM SIGKDD international conference on Knowledge discovery in data mining*, 157–166. ACM, 2005.

[BLTZ08] Fei Tony Liu, Kai Ming Ting, and Zhi-Hua Zhou. Isolation forest. In *Data Mining, 2008. ICDM'08. Eighth IEEE International Conference on*, 413–422. IEEE, 2008.

[BLTZ12] Fei Tony Liu, Kai Ming Ting, and Zhi-Hua Zhou. Isolation-based anomaly detection. *ACM Transactions on Knowledge Discovery from Data (TKDD)*, 6(1):3, 2012.

[BLLZ+18] Yezheng Liu, Zhe Li, Chong Zhou, Yuanchun Jiang, Jianshan Sun, Meng Wang, and Xiangnan He. Generative adversarial active learning for unsupervised outlier detection. *arXiv preprint arXiv:1809.10816*, 2018.

[BMP03] Junshui Ma and Simon Perkins. Time-series novelty detection using one-class support vector machines. In *Neural Networks, 2003. Proceedings of the International Joint Conference on*, volume 3, 1741–1745. IEEE, 2003.

[BPKGF03] Spiros Papadimitriou, Hiroyuki Kitagawa, Phillip B Gibbons, and Christos Faloutsos. Loci: fast outlier detection using the local correlation integral. In *Data Engineering, 2003. Proceedings. 19th International Conference on*, 315–326. IEEE, 2003.

[BRRS00] Sridhar Ramaswamy, Rajeev Rastogi, and Kyuseok Shim. Efficient algorithms for mining outliers from large data sets. In *ACM Sigmod Record*, volume 29, 427–438. ACM, 2000.

[BRD99] Peter J Rousseeuw and Katrien Van Driessen. A fast algorithm for the minimum covariance determinant estimator. *Technometrics*, 41(3):212–223, 1999.

[BSCSC03] Mei-Ling Shyu, Shu-Ching Chen, Kanoksri Sarinnapakorn, and LiWu Chang. A novel anomaly detection scheme based on principal component classifier. Technical Report, MIAMI UNIV CORAL GABLES FL DEPT OF ELECTRICAL AND COMPUTER ENGINEERING, 2003.

[BZH18] Yue Zhao and Maciej K Hryniewicki. Xgbod: improving supervised outlier detection with unsupervised representation learning. In *Neural Networks, 2018. Proceedings of the International Joint Conference on*. IEEE, 2018.

[BZHNL19] Yue Zhao, Maciej K Hryniewicki, Zain Nasrullah, and Zheng Li. LSCP: locally selective combination in parallel outlier ensembles. In *SIAM International Conference on Data Mining (SDM)*. Calgary, Canada, May 2019. Society for Industrial and Applied Mathematics.

[AAgg15] Charu C Aggarwal. Outlier analysis. In *Data mining*, 75–79. Springer, 2015.

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

[AAP02] Fabrizio Angiulli and Clara Pizzuti. Fast outlier detection in high dimensional spaces. In *European Conference on Principles of Data Mining and Knowledge Discovery*, 15–27. Springer, 2002.

[ABKNS00] Markus M Breunig, Hans-Peter Kriegel, Raymond T Ng, and Jörg Sander. Lof: identifying density-based local outliers. In *ACM sigmod record*, volume 29, 93–104. ACM, 2000.

[AGD12] Markus Goldstein and Andreas Dengel. Histogram-based outlier score (hbos): a fast unsupervised anomaly detection algorithm. *KI-2012: Poster and Demo Track*, pages 59–63, 2012.

[AHR04] Johanna Hardin and David M Rocke. Outlier detection in the multiple cluster setting using the minimum covariance determinant estimator. *Computational Statistics & Data Analysis*, 44(4):625–638, 2004.

[AHXD03] Zengyou He, Xiaofei Xu, and Shengchun Deng. Discovering cluster-based local outliers. *Pattern Recognition Letters*, 24(9-10):1641–1650, 2003.

[AJHuszarPvdH12] JHM Janssens, Ferenc Huszár, EO Postma, and HJ van den Herik. Stochastic outlier selection. Technical Report, Technical report TiCC TR 2012-001, Tilburg University, Tilburg Center for Cognition and Communication, Tilburg, The Netherlands, 2012.

[AKZ+08] Hans-Peter Kriegel, Arthur Zimek, and others. Angle-based outlier detection in high-dimensional data. In *Proceedings of the 14th ACM SIGKDD international conference on Knowledge discovery and data mining*, 444–452. ACM, 2008.

[ALK05] Aleksandar Lazarevic and Vipin Kumar. Feature bagging for outlier detection. In *Proceedings of the eleventh ACM SIGKDD international conference on Knowledge discovery in data mining*, 157–166. ACM, 2005.

[ALTZ08] Fei Tony Liu, Kai Ming Ting, and Zhi-Hua Zhou. Isolation forest. In *Data Mining, 2008. ICDM'08. Eighth IEEE International Conference on*, 413–422. IEEE, 2008.

[ALTZ12] Fei Tony Liu, Kai Ming Ting, and Zhi-Hua Zhou. Isolation-based anomaly detection. *ACM Transactions on Knowledge Discovery from Data (TKDD)*, 6(1):3, 2012.

[ALLZ+18] Yezheng Liu, Zhe Li, Chong Zhou, Yuanchun Jiang, Jianshan Sun, Meng Wang, and Xiangnan He. Generative adversarial active learning for unsupervised outlier detection. *arXiv preprint arXiv:1809.10816*, 2018.

[AMP03] Junshui Ma and Simon Perkins. Time-series novelty detection using one-class support vector machines. In *Neural Networks, 2003. Proceedings of the International Joint Conference on*, volume 3, 1741–1745. IEEE, 2003.

[APKGF03] Spiros Papadimitriou, Hiroyuki Kitagawa, Phillip B Gibbons, and Christos Faloutsos. Loci: fast outlier detection using the local correlation integral. In *Data Engineering, 2003. Proceedings. 19th International Conference on*, 315–326. IEEE, 2003.

[ARRS00] Sridhar Ramaswamy, Rajeev Rastogi, and Kyuseok Shim. Efficient algorithms for mining outliers from large data sets. In *ACM Sigmod Record*, volume 29, 427–438. ACM, 2000.

[ARD99] Peter J Rousseeuw and Katrien Van Driessen. A fast algorithm for the minimum covariance determinant estimator. *Technometrics*, 41(3):212–223, 1999.

[ASCSC03] Mei-Ling Shyu, Shu-Ching Chen, Kanoksri Sarinnapakorn, and LiWu Chang. A novel anomaly detection scheme based on principal component classifier. Technical Report, MIAMI UNIV CORAL GABLES FL DEPT OF ELECTRICAL AND COMPUTER ENGINEERING, 2003.

[AZH18a] Yue Zhao and Maciej K Hryniewicki. Xgbod: improving supervised outlier detection with unsupervised representation learning. In *Neural Networks, 2018. Proceedings of the International Joint Conference on*. IEEE, 2018.

[AZHNL19] Yue Zhao, Maciej K Hryniewicki, Zain Nasrullah, and Zheng Li. LSCP: locally selective combination in parallel outlier ensembles. In *SIAM International Conference on Data Mining (SDM)*. Calgary, Canada, May 2019. Society for Industrial and Applied Mathematics.

[AZH18b] Yue Zhao and Maciej K. Hryniewicki. Dcso: dynamic combination of detector scores for outlier ensembles. In *ACM SIGKDD Workshop on Outlier Detection De-constructed (ODD v5.0)*. ACM, 2018.

Python Module Index

e

p

Index

E

F

R

S

T

V

W

X