# Getting Started

## 1 Implemented Algorithms

## 2 API Cheatsheet & Reference

### 2.1 Installation

### 2.2 Examples

### 2.3 Benchmarks

### 2.4 API CheatSheet

### 2.5 API Reference

### 2.6 Known Issues & Warnings

### 2.7 Outlier Detection 101

### 2.8 Citations & Achievements

### 2.9 Frequently Asked Questions

### 2.10 About us

## Bibliography

## Python Module Index

## Index
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 [AZNL19] has been successfully used in various academic researches and commercial products [ARSLS19][AKW19][AZH18b][AZNHL19]. It is also well acknowledged by the machine learning community with various dedicated posts/tutorials, including Analytics Vidhya, KDnuggets, Computer Vision News, and awesome-machine-learning.

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.

Note on Python 2.7: The maintenance of Python 2.7 will be stopped by January 1, 2020 (see official announcement) To be consistent with the Python change and PyOD’s dependent libraries, e.g., scikit-learn, we will stop supporting Python 2.7 in the near future (dates are still to be decided). We encourage you to use Python 3.5 or newer for the latest functions and bug fixes. More information can be found at Moving to require Python 3.

API Demo:

```python
# train the KNN detector
from pyod.models.knn import KNN
clf = KNN()
clf.fit(X_train)

# get outlier scores
y_train_scores = clf.decision_scores_  # raw outlier scores
y_test_scores = clf.decision_function(X_test)  # outlier scores
```
Citing PyOD:

PyOD paper is accepted at JMLR (machine learning open-source software track). If you use PyOD in a scientific publication, we would appreciate citations to the following paper:

```@article{zhao2019pyod,
    author = {Zhao, Yue and Nasrullah, Zain and Li, Zheng},
    title = {PyOD: A Python Toolbox for Scalable Outlier Detection},
    journal = {Journal of Machine Learning Research},
    year = {2019},
    volume = {20},
    number = {96},
    pages = {1-7},
    url = {http://jmlr.org/papers/v20/19-011.html}
}
```

or:


Key Links and Resources:

- View the latest codes on Github
- Execute Interactive Jupyter Notebooks
- Anomaly Detection Resources
PyOD toolkit consists of three major functional groups:

(i) Individual Detection Algorithms:

1. Linear Models for Outlier Detection:
<table>
<thead>
<tr>
<th>Type</th>
<th>Abbr</th>
<th>Algorithm</th>
<th>Year</th>
<th>Class</th>
<th>Ref</th>
</tr>
</thead>
<tbody>
<tr>
<td>Linear Model</td>
<td>PCA</td>
<td>Principal Component Analysis (the sum of weighted projected distances to the eigenvector hyperplanes)</td>
<td>2003</td>
<td>pyod.models.pca.PCA</td>
<td>[ASCSC03]</td>
</tr>
<tr>
<td>Linear Model</td>
<td>MCD</td>
<td>Minimum Covariance Determinant (use the mahalanobis distances as the outlier scores)</td>
<td>1999</td>
<td>pyod.models.mcd.MCD</td>
<td>[ARD99][AHR04]</td>
</tr>
<tr>
<td>Linear Model</td>
<td>OCSVM</td>
<td>MOne-Class Support Vector Machines</td>
<td>2001</td>
<td>pyod.models.ocsvm.OCSVM</td>
<td>[AScholkopfPST+01]</td>
</tr>
<tr>
<td>Proximity Based</td>
<td>LOF</td>
<td>Local Outlier Factor</td>
<td>2000</td>
<td>pyod.models.lof.LOF</td>
<td>[ABKNS00]</td>
</tr>
<tr>
<td>Proximity Based</td>
<td>COF</td>
<td>Connectivity-Based Outlier Factor</td>
<td>2002</td>
<td>pyod.models.cof.COF</td>
<td>[ATCFC02]</td>
</tr>
<tr>
<td>Proximity Based</td>
<td>CBLOF</td>
<td>Clustering-Based Local Outlier Factor</td>
<td>2003</td>
<td>pyod.models.cblof.CBLOF</td>
<td>[AHXD03]</td>
</tr>
<tr>
<td>Proximity Based</td>
<td>LOCI</td>
<td>LOCI: Fast outlier detection using the local correlation integral</td>
<td>2003</td>
<td>pyod.models.loci.LOCI</td>
<td>[APKGF03]</td>
</tr>
<tr>
<td>Proximity Based</td>
<td>HBOS</td>
<td>Histogram-based Outlier Score</td>
<td>2012</td>
<td>pyod.models.hbos.HBOS</td>
<td>[AGD12]</td>
</tr>
<tr>
<td>Proximity Based</td>
<td>kNN</td>
<td>k Nearest Neighbors (use the distance to the kth nearest neighbor as the outlier score)</td>
<td>2000</td>
<td>pyod.models.knn.kNN</td>
<td>[ARRS00][AAP02]</td>
</tr>
<tr>
<td>Proximity Based</td>
<td>AvgKNN</td>
<td>Average kNN (use the average distance to k nearest neighbors as the outlier score)</td>
<td>2002</td>
<td>pyod.models.knn.kNN</td>
<td>[ARRS00][AAP02]</td>
</tr>
<tr>
<td>Proximity Based</td>
<td>Med-KNN</td>
<td>Median kNN (use the median distance to k nearest neighbors as the outlier score)</td>
<td>2002</td>
<td>pyod.models.knn.kNN</td>
<td>[ARRS00][AAP02]</td>
</tr>
<tr>
<td>Proximity Based</td>
<td>SOD</td>
<td>Subspace Outlier Detection</td>
<td>2009</td>
<td>pyod.models.sod.SOD</td>
<td>[BKKrogerSZ09]</td>
</tr>
<tr>
<td>Probabilistic</td>
<td>ABOD</td>
<td>Angle-Based Outlier Detection</td>
<td>2008</td>
<td>pyod.models.abod.ABOD</td>
<td>[AKZ+08]</td>
</tr>
<tr>
<td>Probabilistic</td>
<td>FastABOD</td>
<td>Fast Angle-Based Outlier Detection using approximation</td>
<td>2008</td>
<td>pyod.models.abod.ABOD</td>
<td>[AKZ+08]</td>
</tr>
<tr>
<td>Probabilistic</td>
<td>SOS</td>
<td>Stochastic Outlier Selection</td>
<td>2012</td>
<td>pyod.models.sos.SOS</td>
<td>[AJHuszarPvdH12]</td>
</tr>
<tr>
<td>Outlier Ensembles</td>
<td>IForest</td>
<td>Isolation Forest</td>
<td>2008</td>
<td>pyod.models.ifeast.IForest</td>
<td>[ALTZ08][ALTZ12]</td>
</tr>
<tr>
<td>Outlier Ensembles</td>
<td>LSCP</td>
<td>LSCP: Locally Selective Combination of Parallel Outlier Ensembles</td>
<td>2019</td>
<td>pyod.models.lscp.LSCP</td>
<td>[AZNHL19]</td>
</tr>
<tr>
<td>Outlier Ensembles</td>
<td>XGBOD</td>
<td>Extreme Boosting Based Outlier Detection (Supervised)</td>
<td>2018</td>
<td>pyod.models.xgbod.XGBOD</td>
<td>[AZH18a]</td>
</tr>
<tr>
<td>Neural Networks</td>
<td>AutoEncoder</td>
<td>Fully connected AutoEncoder (use reconstruction error as the outlier score)</td>
<td>2015</td>
<td>pyod.models.auto_encoder.AutoEncoder</td>
<td>[AAgg15]</td>
</tr>
<tr>
<td>Neural Networks</td>
<td>SO_GAAL</td>
<td>Single-Objective Generative Adversarial Active Learning</td>
<td>2019</td>
<td>pyod.models.so_gaal.SO_GAAL</td>
<td>[ALLZ+19]</td>
</tr>
<tr>
<td>Neural Networks</td>
<td>MO_GAAL</td>
<td>Multiple-Objective Generative Adversarial Active Learning</td>
<td>2019</td>
<td>pyod.models.mo_gaal.MO_GAAL</td>
<td>[ALLZ+19]</td>
</tr>
</tbody>
</table>
(ii) Outlier Ensembles & Outlier Detector Combination Frameworks:

<table>
<thead>
<tr>
<th>Type</th>
<th>Abbr</th>
<th>Algorithm</th>
<th>Year</th>
<th>Ref</th>
</tr>
</thead>
<tbody>
<tr>
<td>Outlier Ensembles</td>
<td></td>
<td>Feature Bagging</td>
<td>2005</td>
<td><code>pyod.models.feature_bagging.FeatureBagging</code> [ALK05]</td>
</tr>
<tr>
<td>Outlier Ensembles</td>
<td></td>
<td>LSCP: Locally Selective Combination of Parallel Outlier Ensembles</td>
<td>2019</td>
<td><code>pyod.models.lscp.LSCP</code> [AZNHL19]</td>
</tr>
<tr>
<td>Combination</td>
<td></td>
<td>Simple combination by averaging the scores</td>
<td>2015</td>
<td><code>pyod.models.combination.average()</code> [AAS15]</td>
</tr>
<tr>
<td>Combination</td>
<td></td>
<td>Simple combination by averaging the scores with detector weights</td>
<td>2015</td>
<td><code>pyod.models.combination.average()</code> [AAS15]</td>
</tr>
<tr>
<td>Combination</td>
<td></td>
<td>Simple combination by taking the maximum scores</td>
<td>2015</td>
<td><code>pyod.models.combination.maximization()</code> [AAS15]</td>
</tr>
<tr>
<td>Combination</td>
<td></td>
<td>Average of Maximum</td>
<td>2015</td>
<td><code>pyod.models.combination.aom()</code> [AAS15]</td>
</tr>
<tr>
<td>Combination</td>
<td></td>
<td>Maximum of Average</td>
<td>2015</td>
<td><code>pyod.models.combination.moa()</code> [AAS15]</td>
</tr>
</tbody>
</table>

(iii) Utility Functions:

<table>
<thead>
<tr>
<th>Type</th>
<th>Name</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>Data</td>
<td><code>pyod.utils.data.generate_data()</code></td>
<td>Synthesized data generation; normal data is generated by a multivariate Gaussian and outliers are generated by a uniform distribution</td>
</tr>
<tr>
<td>Data</td>
<td><code>pyod.utils.data.generate_data_clusters()</code></td>
<td>Synthesized data generation in clusters; more complex data patterns can be created with multiple clusters</td>
</tr>
<tr>
<td>Stat</td>
<td><code>pyod.utils.stat_models.wpearsonr()</code></td>
<td>Calculate the weighted Pearson correlation of two samples</td>
</tr>
<tr>
<td>Utility</td>
<td><code>pyod.utils.utility.get_label_n()</code></td>
<td>Turn raw outlier scores into binary labels by assign 1 to top n outlier scores</td>
</tr>
<tr>
<td>Utility</td>
<td><code>pyod.utils.utility.precision_n_scores()</code></td>
<td>calculate precision @ rank n</td>
</tr>
</tbody>
</table>

The comparison among of implemented models is made available below (Figure, compare_all_models.py, Interactive Jupyter Notebooks). For Jupyter Notebooks, please navigate to “/notebooks/Compare All Models.ipynb”.

Check the latest benchmark. You could replicate this process by running benchmark.py.
Chapter 1. Implemented Algorithms
API Cheatsheet & Reference

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.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.
- `pyod.models.base.BaseDetector.fit_predict()`: [Deprecated in V0.6.9] Fit detector first and then predict whether a particular sample is an outlier or not.
- `pyod.models.base.BaseDetector.fit_predict_score()`: [Deprecated in V0.6.9] Fit the detector, predict on samples, and evaluate the model by predefined metrics, e.g., ROC.

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.

Note: `fit_predict()` and `fit_predict_score()` are deprecated in V0.6.9 due to consistency issue and will be removed in V0.7.2. To get the binary labels of the training data X_train, one should call clf.fit(X_train) and use `pyod.models.base.BaseDetector.labels_`, instead of calling clf.predict(X_train).
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     # normal install
pip install --upgrade pyod  # or update if needed
pip install --pre pyod    # or include pre-release version for new features
```

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

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

**Warning:** The maintenance of Python 2.7 will be stopped by January 1, 2020 (see official announcement). To be consistent with the Python change and PyOD’s dependent libraries, e.g., scikit-learn, we will stop supporting Python 2.7 in the near future (dates are still to be decided). We encourage you to use Python 3.5 or newer for the latest functions and bug fixes. More information can be found at Moving to require Python 3.

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)
- pandas (optional, required for running benchmark)
- tensorflow (optional, required for AutoEncoder, other backend works)
- xgboost (optional, required for XGBOD)

**Warning:** PyOD has multiple neural network based models, e.g., AutoEncoders, which are implemented in Keras. However, PyOD does NOT install keras and/or tensorflow for you. This reduces the risk of interfering with your local copies. If you want to use neural-net based models, please make sure Keras and a backend library, e.g., TensorFlow, are installed. Instructions are provided: neural-net FAQ. Similarly, models depending on xgboost, e.g., XGBOD, would NOT enforce xgboost installation by default.

**Warning:** Running examples needs matplotlib, which may throw errors in conda virtual environment on mac OS. See reasons and solutions mac_matplotlib.
Warning: PyOD contains multiple models that also exist in scikit-learn. However, these two libraries’ API is not exactly the same—it is recommended to use only one of them for consistency but not mix the results. Refer scikit-learn and PyOD for more information.

2.2 Examples

2.2.1 Featured Tutorials

PyOD has been well acknowledged by the machine learning community with a few featured posts and tutorials.

Analytics Vidhya: An Awesome Tutorial to Learn Outlier Detection in Python using PyOD Library

KDnuggets: Intuitive Visualization of Outlier Detection Methods

Computer Vision News (March 2019): Python Open Source Toolbox for Outlier Detection

awesome-machine-learning: General-Purpose Machine Learning

2.2.2 kNN Example

Full example: knn_example.py

1. Import models

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

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

```python
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.

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

# 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()`.
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

Outlier detection often suffers from model instability due to its unsupervised nature. Thus, it is recommended to combine various detector outputs, e.g., by averaging, to improve its robustness. Detector combination is a subfield of outlier ensembles; refer [BKalayciE18] for more information.

Four score combination mechanisms are shown in this demo:

1. **Average**: average scores of all detectors.
2. **maximization**: maximum score across all detectors.
3. **Average of Maximum (AOM)**: divide base detectors into subgroups and take the maximum score for each subgroup. The final score is the average of all subgroup scores.
4. **Maximum of Average (MOA)**: divide base detectors into subgroups and take the average score for each subgroup. The final score is the maximum of all subgroup scores.

“examples/comb_example.py” illustrates the API for combining the output of multiple base detectors (comb_example.py, Jupyter Notebooks). For Jupyter Notebooks, please navigate to “/notebooks/Model Combination.ipynb”

1. Import models and generate sample data.

```python
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. Initialize 20 kNN outlier detectors with different k (10 to 200), and get the outlier scores.

```python
# 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])

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 scores are standardized into zero average and unit std before combination. This step is crucial to adjust the detector outputs to the same scale.

```python
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. Four different combination algorithms are applied as described above:
5. Finally, all four combination methods are evaluated by ROC and Precision @ Rank n:

```python
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
```

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

References

2.3 Benchmarks

2.3.1 Introduction

A benchmark is supplied for select algorithms to provide an overview of the implemented models. In total, 17 benchmark datasets are used for comparison, which can 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 random splits. The mean of 10 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

You could replicate this process by running `benchmark.py`.

We also provide the hardware specification for reference.

<table>
<thead>
<tr>
<th>Specification</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Platform</td>
<td>PC</td>
</tr>
<tr>
<td>OS</td>
<td>Microsoft Windows 10 Enterprise</td>
</tr>
<tr>
<td>CPU</td>
<td>Intel i7-6820HQ @ 2.70GHz</td>
</tr>
<tr>
<td>RAM</td>
<td>32GB</td>
</tr>
<tr>
<td>Software</td>
<td>PyCharm 2018.02</td>
</tr>
<tr>
<td>Python</td>
<td>Python 3.6.2</td>
</tr>
<tr>
<td>Core</td>
<td>Single core (no parallelization)</td>
</tr>
</tbody>
</table>
## 2.3.2 ROC Performance

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

<table>
<thead>
<tr>
<th>Data</th>
<th>#Samples</th>
<th># Dimensions</th>
<th>Outlier Perc</th>
<th>ABOD</th>
<th>CBLOF</th>
<th>FFB</th>
<th>HBOS</th>
<th>IForest</th>
<th>KNN</th>
<th>LOF</th>
<th>MCD</th>
<th>OCSVM</th>
<th>PCA</th>
</tr>
</thead>
<tbody>
<tr>
<td>arrhythmia</td>
<td>452</td>
<td>274</td>
<td>14.6018</td>
<td>0.7688</td>
<td>0.7835</td>
<td>0.7781</td>
<td>0.8219</td>
<td>0.8005</td>
<td>0.7877</td>
<td>0.7799</td>
<td>0.7812</td>
<td>0.7815</td>
<td></td>
</tr>
<tr>
<td>cardio</td>
<td>1831</td>
<td>21</td>
<td>9.6122</td>
<td>0.5692</td>
<td>0.9276</td>
<td>0.7835</td>
<td>0.7835</td>
<td>0.7835</td>
<td>0.7835</td>
<td>0.7835</td>
<td>0.7835</td>
<td>0.7835</td>
<td></td>
</tr>
<tr>
<td>glass</td>
<td>214</td>
<td>9</td>
<td>4.2056</td>
<td>0.7951</td>
<td>0.8504</td>
<td>0.7835</td>
<td>0.7835</td>
<td>0.7835</td>
<td>0.7835</td>
<td>0.7835</td>
<td>0.7835</td>
<td>0.7835</td>
<td></td>
</tr>
<tr>
<td>ionosphere</td>
<td>351</td>
<td>33</td>
<td>35.8974</td>
<td>0.9248</td>
<td>0.8134</td>
<td>0.8750</td>
<td>0.8750</td>
<td>0.8750</td>
<td>0.8750</td>
<td>0.8750</td>
<td>0.8750</td>
<td>0.8750</td>
<td></td>
</tr>
<tr>
<td>letter</td>
<td>1600</td>
<td>32</td>
<td>6.2500</td>
<td>0.8785</td>
<td>0.5070</td>
<td>0.8785</td>
<td>0.8785</td>
<td>0.8785</td>
<td>0.8785</td>
<td>0.8785</td>
<td>0.8785</td>
<td>0.8785</td>
<td></td>
</tr>
<tr>
<td>lymph</td>
<td>148</td>
<td>18</td>
<td>4.0541</td>
<td>0.9110</td>
<td>0.9728</td>
<td>0.9728</td>
<td>0.9728</td>
<td>0.9728</td>
<td>0.9728</td>
<td>0.9728</td>
<td>0.9728</td>
<td>0.9728</td>
<td></td>
</tr>
<tr>
<td>mnist</td>
<td>7603</td>
<td>100</td>
<td>9.2069</td>
<td>0.7815</td>
<td>0.8009</td>
<td>0.7815</td>
<td>0.7815</td>
<td>0.7815</td>
<td>0.7815</td>
<td>0.7815</td>
<td>0.7815</td>
<td>0.7815</td>
<td></td>
</tr>
<tr>
<td>musk</td>
<td>3062</td>
<td>166</td>
<td>3.1679</td>
<td>0.1844</td>
<td>0.9879</td>
<td>0.5263</td>
<td>0.5263</td>
<td>0.5263</td>
<td>0.5263</td>
<td>0.5263</td>
<td>0.5263</td>
<td>0.5263</td>
<td></td>
</tr>
<tr>
<td>optdigits</td>
<td>5216</td>
<td>64</td>
<td>2.8758</td>
<td>0.4667</td>
<td>0.5089</td>
<td>0.4434</td>
<td>0.4434</td>
<td>0.4434</td>
<td>0.4434</td>
<td>0.4434</td>
<td>0.4434</td>
<td>0.4434</td>
<td></td>
</tr>
<tr>
<td>pendigits</td>
<td>6870</td>
<td>16</td>
<td>2.2707</td>
<td>0.6878</td>
<td>0.9486</td>
<td>0.4596</td>
<td>0.4596</td>
<td>0.4596</td>
<td>0.4596</td>
<td>0.4596</td>
<td>0.4596</td>
<td>0.4596</td>
<td></td>
</tr>
<tr>
<td>pima</td>
<td>768</td>
<td>8</td>
<td>34.8958</td>
<td>0.6794</td>
<td>0.7348</td>
<td>0.6235</td>
<td>0.6235</td>
<td>0.6235</td>
<td>0.6235</td>
<td>0.6235</td>
<td>0.6235</td>
<td>0.6235</td>
<td></td>
</tr>
<tr>
<td>satellite</td>
<td>6435</td>
<td>36</td>
<td>31.6395</td>
<td>0.5714</td>
<td>0.6693</td>
<td>0.5572</td>
<td>0.5572</td>
<td>0.5572</td>
<td>0.5572</td>
<td>0.5572</td>
<td>0.5572</td>
<td>0.5572</td>
<td></td>
</tr>
<tr>
<td>satimage2</td>
<td>5803</td>
<td>36</td>
<td>1.2235</td>
<td>0.8190</td>
<td>0.9917</td>
<td>0.4570</td>
<td>0.4570</td>
<td>0.4570</td>
<td>0.4570</td>
<td>0.4570</td>
<td>0.4570</td>
<td>0.4570</td>
<td></td>
</tr>
<tr>
<td>shuttle</td>
<td>49097</td>
<td>9</td>
<td>7.1511</td>
<td>0.6234</td>
<td>0.6272</td>
<td>0.4724</td>
<td>0.4724</td>
<td>0.4724</td>
<td>0.4724</td>
<td>0.4724</td>
<td>0.4724</td>
<td>0.4724</td>
<td></td>
</tr>
<tr>
<td>vertebral</td>
<td>240</td>
<td>6</td>
<td>12.5000</td>
<td>0.4262</td>
<td>0.3486</td>
<td>0.4166</td>
<td>0.4166</td>
<td>0.4166</td>
<td>0.4166</td>
<td>0.4166</td>
<td>0.4166</td>
<td>0.4166</td>
<td></td>
</tr>
<tr>
<td>vowels</td>
<td>1456</td>
<td>12</td>
<td>3.4341</td>
<td>0.9605</td>
<td>0.5856</td>
<td>0.9425</td>
<td>0.9425</td>
<td>0.9425</td>
<td>0.9425</td>
<td>0.9425</td>
<td>0.9425</td>
<td>0.9425</td>
<td></td>
</tr>
<tr>
<td>wbc</td>
<td>378</td>
<td>30</td>
<td>5.5556</td>
<td>0.9047</td>
<td>0.9227</td>
<td>0.9325</td>
<td>0.9325</td>
<td>0.9325</td>
<td>0.9325</td>
<td>0.9325</td>
<td>0.9325</td>
<td>0.9325</td>
<td></td>
</tr>
</tbody>
</table>
## 2.3.3 P@N Performance

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

<table>
<thead>
<tr>
<th>Data</th>
<th>#Samples</th>
<th># Dimensions</th>
<th>Outlier Perc</th>
<th>ABOD</th>
<th>CBLOF</th>
<th>FBF</th>
<th>HBOS</th>
<th>IForest</th>
<th>KNN</th>
<th>LOF</th>
<th>MCD</th>
<th>OCSVM</th>
<th>PCA</th>
</tr>
</thead>
<tbody>
<tr>
<td>arrhythmia</td>
<td>452</td>
<td>274</td>
<td>14.6018</td>
<td>0.3808</td>
<td>0.4539</td>
<td>0.4230</td>
<td>0.496</td>
<td>0.446</td>
<td>0.3995</td>
<td>0.4613</td>
<td>0.4613</td>
<td></td>
<td></td>
</tr>
<tr>
<td>cardio</td>
<td>1831</td>
<td>21</td>
<td>9.6122</td>
<td>0.2374</td>
<td>0.5876</td>
<td>0.169</td>
<td>0.4476</td>
<td>0.504</td>
<td>0.3323</td>
<td>0.1541</td>
<td>0.4317</td>
<td>0.5011</td>
<td>0.6099</td>
</tr>
<tr>
<td>glass</td>
<td>214</td>
<td>9</td>
<td>4.2056</td>
<td>0.1702</td>
<td>0.2374</td>
<td>0.1476</td>
<td>0.0000</td>
<td>0.0726</td>
<td>0.0726</td>
<td>0.0726</td>
<td>0.0726</td>
<td>0.0726</td>
<td></td>
</tr>
<tr>
<td>ionosphere</td>
<td>351</td>
<td>33</td>
<td>35.8974</td>
<td>0.8442</td>
<td>0.6088</td>
<td>0.7056</td>
<td>0.3295</td>
<td>0.6369</td>
<td>0.8602</td>
<td>0.7000</td>
<td>0.5720</td>
<td></td>
<td></td>
</tr>
<tr>
<td>letter</td>
<td>1600</td>
<td>32</td>
<td>6.2500</td>
<td>0.3801</td>
<td>0.3424</td>
<td>0.3642</td>
<td>0.0715</td>
<td>0.1003</td>
<td>0.0301</td>
<td>0.0234</td>
<td>0.0000</td>
<td>0.0000</td>
<td>0.0876</td>
</tr>
<tr>
<td>lymph</td>
<td>148</td>
<td>18</td>
<td>4.0541</td>
<td>0.4483</td>
<td>0.7517</td>
<td>0.7517</td>
<td>0.8467</td>
<td>0.926</td>
<td>0.7517</td>
<td>0.7517</td>
<td>0.7517</td>
<td>0.7517</td>
<td></td>
</tr>
<tr>
<td>mnist</td>
<td>7603</td>
<td>100</td>
<td>9.2069</td>
<td>0.3555</td>
<td>0.3348</td>
<td>0.3299</td>
<td>0.1188</td>
<td>0.313</td>
<td>0.4204</td>
<td>0.3462</td>
<td>0.3962</td>
<td>0.3462</td>
<td>0.3846</td>
</tr>
<tr>
<td>musk</td>
<td>3062</td>
<td>166</td>
<td>3.1679</td>
<td>0.0507</td>
<td>0.7766</td>
<td>0.223</td>
<td>0.9788</td>
<td>0.968</td>
<td>0.2733</td>
<td>0.1695</td>
<td>0.9742</td>
<td>1.0000</td>
<td>0.9799</td>
</tr>
<tr>
<td>optdigits</td>
<td>5216</td>
<td>64</td>
<td>2.8758</td>
<td>0.0060</td>
<td>0.0000</td>
<td>0.0244</td>
<td>0.2194</td>
<td>0.0301</td>
<td>0.0000</td>
<td>0.0234</td>
<td>0.0000</td>
<td>0.0000</td>
<td>0.0000</td>
</tr>
<tr>
<td>pendigits</td>
<td>6870</td>
<td>16</td>
<td>2.2707</td>
<td>0.0812</td>
<td>0.2768</td>
<td>0.0658</td>
<td>0.2979</td>
<td>0.3422</td>
<td>0.0984</td>
<td>0.0653</td>
<td>0.0895</td>
<td>0.3287</td>
<td>0.3187</td>
</tr>
<tr>
<td>pima</td>
<td>768</td>
<td>8</td>
<td>34.8958</td>
<td>0.5195</td>
<td>0.5413</td>
<td>0.4480</td>
<td>0.5428</td>
<td>0.511</td>
<td>0.5413</td>
<td>0.4555</td>
<td>0.4962</td>
<td>0.4704</td>
<td>0.4943</td>
</tr>
<tr>
<td>satellite</td>
<td>6435</td>
<td>36</td>
<td>31.6395</td>
<td>0.3902</td>
<td>0.4152</td>
<td>0.3902</td>
<td>0.5690</td>
<td>0.5676</td>
<td>0.4994</td>
<td>0.3893</td>
<td>0.6848</td>
<td>0.5346</td>
<td>0.4784</td>
</tr>
<tr>
<td>satimage2</td>
<td>5803</td>
<td>36</td>
<td>1.2235</td>
<td>0.2130</td>
<td>0.8844</td>
<td>0.0555</td>
<td>0.6939</td>
<td>0.8754</td>
<td>0.3809</td>
<td>0.0555</td>
<td>0.6481</td>
<td>0.9356</td>
<td>0.8041</td>
</tr>
<tr>
<td>shuttle</td>
<td>49097</td>
<td>9</td>
<td>7.1511</td>
<td>0.1977</td>
<td>0.2943</td>
<td>0.0695</td>
<td>0.9551</td>
<td>0.9546</td>
<td>0.2184</td>
<td>0.1424</td>
<td>0.7505</td>
<td>0.9542</td>
<td>0.9501</td>
</tr>
<tr>
<td>vertebral</td>
<td>240</td>
<td>6</td>
<td>12.5000</td>
<td>0.0601</td>
<td>0.0000</td>
<td>0.0644</td>
<td>0.0071</td>
<td>0.0341</td>
<td>0.0238</td>
<td>0.0506</td>
<td>0.0071</td>
<td>0.0238</td>
<td>0.0225</td>
</tr>
<tr>
<td>vowels</td>
<td>1456</td>
<td>12</td>
<td>3.4341</td>
<td>0.5710</td>
<td>0.8831</td>
<td>0.3224</td>
<td>0.1297</td>
<td>0.1875</td>
<td>0.5093</td>
<td>0.3551</td>
<td>0.2185</td>
<td>0.2791</td>
<td>0.1364</td>
</tr>
<tr>
<td>wbc</td>
<td>378</td>
<td>30</td>
<td>5.5556</td>
<td>0.3060</td>
<td>0.5055</td>
<td>0.5188</td>
<td>0.5817</td>
<td>0.5083</td>
<td>0.4952</td>
<td>0.5188</td>
<td>0.4577</td>
<td>0.5125</td>
<td>0.4767</td>
</tr>
</tbody>
</table>
2.3.4 Execution Time

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

<table>
<thead>
<tr>
<th>Data</th>
<th>#Samples</th>
<th>#Dimensions</th>
<th>Outlier Perc</th>
<th>ABOF</th>
<th>CBLOF</th>
<th>FFB</th>
<th>HBOSIForest</th>
<th>KNN</th>
<th>LOF</th>
<th>MCD</th>
<th>OCSVM</th>
<th>MPCAA</th>
</tr>
</thead>
<tbody>
<tr>
<td>arrhythmia</td>
<td>452</td>
<td>274</td>
<td>14.6018</td>
<td>0.3667</td>
<td>0.2123</td>
<td>0.565</td>
<td>0.1383</td>
<td>0.1075</td>
<td>0.0743</td>
<td>1.4165</td>
<td>0.0473</td>
<td>0.0596</td>
</tr>
<tr>
<td>cardio</td>
<td>1831</td>
<td>21</td>
<td>9.6122</td>
<td>0.3824</td>
<td>0.1253</td>
<td>0.7744</td>
<td>0.0053</td>
<td>0.2249</td>
<td>0.0909</td>
<td>0.5418</td>
<td>0.0833</td>
<td>0.0033</td>
</tr>
<tr>
<td>glass</td>
<td>214</td>
<td>9</td>
<td>4.2056</td>
<td>0.0353</td>
<td>0.0359</td>
<td>0.0317</td>
<td>0.0173</td>
<td>0.0173</td>
<td>0.0025</td>
<td>0.0325</td>
<td>0.0010</td>
<td>0.0011</td>
</tr>
<tr>
<td>ionosphere</td>
<td>351</td>
<td>33</td>
<td>35.8974</td>
<td>0.0645</td>
<td>0.0459</td>
<td>0.0728</td>
<td>0.0080</td>
<td>0.2086</td>
<td>0.0070</td>
<td>0.0718</td>
<td>0.0048</td>
<td>0.0018</td>
</tr>
<tr>
<td>letter</td>
<td>1600</td>
<td>32</td>
<td>6.2500</td>
<td>0.3435</td>
<td>0.1014</td>
<td>0.7361</td>
<td>0.0080</td>
<td>0.1882</td>
<td>0.0935</td>
<td>1.1942</td>
<td>0.0888</td>
<td>0.0044</td>
</tr>
<tr>
<td>lympho</td>
<td>148</td>
<td>18</td>
<td>4.0541</td>
<td>0.0277</td>
<td>0.0353</td>
<td>0.0266</td>
<td>0.0037</td>
<td>0.1712</td>
<td>0.0010</td>
<td>0.0327</td>
<td>0.0014</td>
<td>0.0012</td>
</tr>
<tr>
<td>mnist</td>
<td>7603</td>
<td>100</td>
<td>9.2069</td>
<td>7.4192</td>
<td>1.1319</td>
<td>48.2710</td>
<td>0.1348</td>
<td>0.0193</td>
<td>0.0265</td>
<td>4.7448</td>
<td>5.0203</td>
<td>0.1569</td>
</tr>
<tr>
<td>musk</td>
<td>3062</td>
<td>166</td>
<td>3.1679</td>
<td>2.3860</td>
<td>0.4134</td>
<td>13.8610</td>
<td>0.0580</td>
<td>7.2176</td>
<td>0.0070</td>
<td>25.5511</td>
<td>0.3774</td>
<td>0.1637</td>
</tr>
<tr>
<td>optdigits</td>
<td>5216</td>
<td>64</td>
<td>2.8758</td>
<td>2.7279</td>
<td>0.4977</td>
<td>14.2399</td>
<td>0.0303</td>
<td>0.7783</td>
<td>0.1882</td>
<td>0.9698</td>
<td>0.0888</td>
<td>0.0044</td>
</tr>
<tr>
<td>pendigits</td>
<td>6870</td>
<td>16</td>
<td>2.2707</td>
<td>1.4339</td>
<td>0.2847</td>
<td>3.8185</td>
<td>0.0090</td>
<td>0.5875</td>
<td>0.9666</td>
<td>2.2209</td>
<td>0.9666</td>
<td>0.0062</td>
</tr>
<tr>
<td>pima</td>
<td>768</td>
<td>8</td>
<td>34.8958</td>
<td>0.1357</td>
<td>0.0698</td>
<td>0.0906</td>
<td>0.0019</td>
<td>0.0925</td>
<td>0.0590</td>
<td>0.0102</td>
<td>0.0474</td>
<td>0.0087</td>
</tr>
<tr>
<td>satellite</td>
<td>6435</td>
<td>36</td>
<td>31.6395</td>
<td>1.7970</td>
<td>0.4269</td>
<td>7.5566</td>
<td>0.0160</td>
<td>0.6449</td>
<td>1.2580</td>
<td>0.9868</td>
<td>2.6916</td>
<td>0.1625</td>
</tr>
<tr>
<td>satimage2</td>
<td>5803</td>
<td>36</td>
<td>1.2235</td>
<td>1.5209</td>
<td>0.3705</td>
<td>5.6561</td>
<td>0.0148</td>
<td>0.5529</td>
<td>1.0587</td>
<td>0.7525</td>
<td>2.3935</td>
<td>1.1114</td>
</tr>
<tr>
<td>shuttle</td>
<td>49097</td>
<td>9</td>
<td>7.1511</td>
<td>14.36</td>
<td>11.2524</td>
<td>59.2113</td>
<td>0.0953</td>
<td>3.3906</td>
<td>9.4958</td>
<td>11.1500</td>
<td>0.1214</td>
<td>0.4944</td>
</tr>
<tr>
<td>vertebral</td>
<td>240</td>
<td>6</td>
<td>12.5000</td>
<td>0.0529</td>
<td>0.0444</td>
<td>0.0339</td>
<td>0.0014</td>
<td>0.1786</td>
<td>0.0160</td>
<td>0.0025</td>
<td>0.0446</td>
<td>0.0015</td>
</tr>
<tr>
<td>vowels</td>
<td>1456</td>
<td>12</td>
<td>3.4341</td>
<td>0.3380</td>
<td>0.0889</td>
<td>0.3132</td>
<td>0.0042</td>
<td>0.2751</td>
<td>0.1125</td>
<td>0.0367</td>
<td>0.0175</td>
<td>0.0023</td>
</tr>
<tr>
<td>wbc</td>
<td>378</td>
<td>30</td>
<td>5.5556</td>
<td>0.1014</td>
<td>0.0691</td>
<td>0.0771</td>
<td>0.0063</td>
<td>0.2030</td>
<td>0.0287</td>
<td>0.0700</td>
<td>0.0864</td>
<td>0.0062</td>
</tr>
</tbody>
</table>

2.4 API CheatSheet

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.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.
- `pyod.models.base.BaseDetector.fit_predict()`: [Deprecated in V0.6.9] Fit detector first and then predict whether a particular sample is an outlier or not.
• `pyod.models.base.BaseDetector.fit_predict_score()` [Deprecated in V0.6.9] Fit the detector, predict on samples, and evaluate the model by predefined metrics, e.g., ROC.

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.

**Note**: `fit_predict()` and `fit_predict_score()` are deprecated in V0.6.9 due to consistency issue and will be removed in V0.7.2. To get the binary labels of the training data `X_train`, one should call `clf.fit(X_train)` and use `pyod.models.base.BaseDetector.labels_`, instead of calling `clf.predict(X_train)`.

See base class definition below:

### 2.4.1 pyod.models.base module

Base class for all outlier detector models

```python
class pyod.models.base.BaseDetector(contamination=0.1)
```

Abstract class for all outlier detection algorithms.

`pyod` would stop supporting Python 2 in the future. Consider move to Python 3.5+.

**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_**
- The outlier scores of the training data. The higher, the more abnormal. Outliers tend to have higher scores. This value is available once the detector is fitted.
  
  **Type** numpy array of shape (n_samples,)

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

**labels_**
- The binary labels of the training data. 0 stands for inliers and 1 for outliers/anomalies. It is generated by applying `threshold_` on `decision_scores_`.
  
  **Type** int, either 0 or 1

**decision_function** (X)
- Predict raw anomaly scores of X using the fitted detector.
  - The anomaly score of an input sample is computed based on the fitted detector. For consistency, outliers are assigned with higher anomaly scores.

  **Parameters**
  
  - `X` (numpy array of shape (n_samples, n_features)) – The 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 \( (X, y=None) \)  
DEPRECATED

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

- \( X \) [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).

outlier_labels [numpy array of shape \((n\_samples,\))] 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.

Deprecated since version 0.6.9: fit_predict will be removed in pyod 0.7.2.; it will be replaced by calling fit function first and then accessing labels_ attribute for consistency.

fit_predict_score \( (X, y, scoring='roc\_auc\_score') \)  
DEPRECATED

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

- \( X \) [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

score : float

Deprecated since version 0.6.9: fit_predict_score will be removed in pyod 0.7.2.; it will be replaced by calling fit function first and then accessing labels_ attribute for consistency. Scoring could be done by calling an evaluation method, e.g., AUC ROC.

get_params \( (deep=True) \)  
Get parameters for this estimator.


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.


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

  Type  numpy array of shape (n_samples,)

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

  Type  float

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

  Type  int, either 0 or 1

**decision_function** (X)
Predict raw anomaly 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)
DEPRECATED

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

  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).

**outlier_labels** [numpy array of shape (n_samples,)] 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.

Deprecated since version 0.6.9: `fit_predict` will be removed in pyod 0.7.2.; it will be replaced by calling `fit` function first and then accessing `labels_` attribute for consistency.

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

DEPRECATED

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

**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

**score** : float

Deprecated since version 0.6.9: `fit_predict_score` will be removed in pyod 0.7.2.; it will be replaced by calling `fit` function first and then accessing `labels_` attribute for consistency. Scoring could be done by calling an evaluation method, e.g., AUC ROC.

**get_params** *(deep=True)*

Get parameters for this estimator.


**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*,)

```python
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.


Returns self

Return type object

**pyod.models.auto_encoder module**

Using Auto Encoder with Outlier Detection

```python
class pyod.models.auto_encoder.AutoEncoder
```

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/](https://keras.io/activations/)
- **output_activation** (*str*, *optional* *default='sigmoid'*) – Activation function to use for output layer. See [https://keras.io/activations/](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/](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_**
The number of neurons in the encoding layer.

  Type int

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

  Type float

**model_**
The underlying AutoEncoder in Keras.

  Type Keras Object

**history_**
The AutoEncoder training history.

  Type Keras Object

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

  Type numpy array of shape (n_samples,)
threshold_
The threshold is based on contamination. It is the \( n_{\text{samples}} \times \text{contamination} \) most abnormal samples in \texttt{decision_scores_}. The threshold is calculated for generating binary outlier labels.

Type float

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

Type int, either 0 or 1

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

- \( X \) (\texttt{numpy array of shape \((n_{\text{samples}}, n_{\text{features}})\)}) – The training input samples. Sparse matrices are accepted only if they are supported by the base estimator.

Returns \texttt{anomaly_scores} – The anomaly score of the input samples.

Return type \texttt{numpy array of shape \((n_{\text{samples}},)\)}

fit (X, y=None)
Fit detector. y is optional for unsupervised methods.

Parameters

- \( X \) (\texttt{numpy array of shape \((n_{\text{samples}}, n_{\text{features}})\)}) – The input samples.
- \( y \) (\texttt{numpy array of shape \((n_{\text{samples}},)\), optional (default=None)}) – The ground truth of the input samples (labels).

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

Parameters

- \( X \) [\texttt{numpy array of shape \((n_{\text{samples}}, n_{\text{features}})\)}] The input samples.
- \( y \) [\texttt{numpy array of shape \((n_{\text{samples}},)\), optional (default=None)}] The ground truth of the input samples (labels).

outlier_labels [\texttt{numpy array of shape \((n_{\text{samples}},)\)}] 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.

Depreciated since version 0.6.9: \texttt{fit_predict} will be removed in pyod 0.7.2; it will be replaced by calling \texttt{fit} function first and then accessing \texttt{labels_} attribute for consistency.

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

Parameters

- \( X \) [\texttt{numpy array of shape \((n_{\text{samples}}, n_{\text{features}})\)}] The input samples.
- \( y \) [\texttt{numpy array of shape \((n_{\text{samples}},)\), 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

score : float
  Deprecated since version 0.6.9: fit_predict_score will be removed in pyod 0.7.2; it will be replaced by calling fit function first and then accessing labels_ attribute for consistency. Scoring could be done by calling an evaluation method, e.g., AUC ROC.

get_params (deep=True)
  Get parameters for this estimator.


  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.

Returns `self`

Return type `object`

**pyod.models.cblof module**

Clustering Based Local Outlier Factor (CBLOF)

```python
class pyod.models.cblof.CBLOF(n_clusters=8, contamination=0.1, clustering_estimator=None, alpha=0.9, beta=5, use_weights=False, check_estimator=False, random_state=None, n_jobs=1)
```

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, kMeans 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.


- `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 \(|C_1|, |C_2|, \ldots, |C_n|\), \(\beta = |C_k|/|C_{k-1}|\)

- `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=False))` – If set to True, check whether the base estimator is consistent with sklearn standard.
Warning: `check_estimator` may throw errors with scikit-learn 0.20 above.

- **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`.

- **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.

**clustering_estimator_**
- **Type** Estimator, sklearn instance

**cluster_labels_**
- **Type** list of shape (n_samples,)

**n_clusters_**
- **Type** int

**cluster_sizes_**
- **Type** list of shape (n_clusters_,)

**decision_scores_**
- **Type** numpy array of shape (n_samples,)

**cluster_centers_**
- **Type** numpy array of shape (n_clusters_, n_features)

**small_cluster_labels_**
- **Type** list of clusters numbers

**large_cluster_labels_**
- **Type** list of clusters numbers

**threshold_**
- **Type** float

**labels_**
- **Type** int, either 0 or 1
**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).

**Returns**

- **outlier_labels** (*numpy array of shape (n_samples,)*) – 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.

Deprecation since version 0.6.9: **fit_predict** will be removed in pyod 0.7.2.; it will be replaced by calling **fit** function first and then accessing **labels_** attribute for consistency.

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

DEPRECATED

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

- **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

**score** : float

Deprecation since version 0.6.9: **fit_predict_score** will be removed in pyod 0.7.2.; it will be replaced by calling **fit** function first and then accessing **labels_** attribute for consistency. Scoring could be done by calling an evaluation method, e.g., AUC ROC.
get_params (deep=True)

Get parameters for this estimator.


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.


Returns self

Return type object

pyod.models.cof module

Connectivity-Based Outlier Factor (COF) Algorithm
class pyod.models.cof.COF (contamination=0.1, n_neighbors=20)

Bases: pyod.models.base.BaseDetector

Connectivity-Based Outlier Factor (COF) COF uses the ratio of average chaining distance of data point and the average of average chaining distance of k nearest neighbor of the data point, as the outlier score for observations. See [BTCFC02] 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.

• n_neighbors (int, optional (default=20)) – Number of neighbors to use by default for k neighbors queries. Note that n_neighbors should be less than the number of samples. If n_neighbors is larger than the number of samples provided, all samples will be used.

decision_scores_

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

Type numpy array of shape (n_samples,)

threshold_

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

Type float

labels_

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

Type int, either 0 or 1

n_neighbors_

Number of neighbors to use by default for k neighbors queries.

Type int

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)*

DEPRECATED

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

- **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).

**outlier_labels** [numpy array of shape (n_samples,)] 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.

Deprecated since version 0.6.9: *fit_predict* will be removed in *pyod 0.7.2,*; it will be replaced by calling *fit* function first and then accessing *labels_* attribute for consistency.

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

DEPRECATED

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

- **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

**score** : float

Deprecated since version 0.6.9: *fit_predict_score* will be removed in *pyod 0.7.2,*; it will be replaced by calling *fit* function first and then accessing *labels_* attribute for consistency. Scoring could be done by calling an evaluation method, e.g., AUC ROC.

**get_params** *(deep=True)*

Get parameters for this estimator.


- **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.


**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*)


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_detector=True, check_estimator=False, n_jobs=1, random_state=None, contamination='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_detector** *(bool, optional (default=True))* — If set to True, check whether the base estimator is consistent with pyod standard.

- **check_estimator** *(bool, optional (default=False))* — If set to True, check whether the base estimator is consistent with sklearn standard.

  Deprecated since version 0.6.9: check_estimator will be removed in pyod 0.7.2.; it will be replaced by check_detector.

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

  Type  numpy array of shape (n_samples,)

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

  Type  float

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

  Type  int, either 0 or 1

decision_function(X)
  Predict raw anomaly 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)
  DEPRECATED
Fit detector first and then predict whether a particular sample is an outlier or not.

- **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).

**outlier_labels** [numpy array of shape (n_samples,)] 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.

Depreciated since version 0.6.9: fit_predict will be removed in pyod 0.7.2.; it will be replaced by calling fit function first and then accessing labels_ attribute for consistency.

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

**get_params**(deep=True)

Get parameters for this estimator.

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.

**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.

**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 [scikit-learn.org](http://scikit-learn.org) and [sklearn/base.py](http://scikit-learn.org) 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 [BDG12] 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_**

- **The edges of the bins.**

- **Type**
- **numpy array of shape (n_bins + 1, n_features )**

**hist_**

- **The density of each histogram.**

- **Type**
- **numpy array of shape (n_bins, n_features)**
**decision_scores_**
The outlier scores of the training data. The higher, the more abnormal. Outliers tend to have higher scores. This value is available once the detector is fitted.

Type: numpy array of shape (n_samples,)

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

Type: float

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

Type: int, either 0 or 1

**decision_function(X)**
Predict raw anomaly 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)**
DEPRECATED

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).

outlier_labels [numpy array of shape (n_samples,)] 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.

Deprecated since version 0.6.9: fit_predict will be removed in pyod 0.7.2.; it will be replaced by calling fit function first and then accessing labels_ attribute for consistency.

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

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

2.5. API Reference
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

score : float
Deprecated since version 0.6.9: fit_predict_score will be removed in pyod 0.7.2.; it will be replaced by calling fit function first and then accessing labels_ attribute for consistency. Scoring could be done by calling an evaluation method, e.g., AUC ROC.

get_params (deep=True)
Get parameters for this estimator.

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.


Returns self

Return type object

**pyod.models.iforest module**

IsolationForest Outlier Detector. Implemented on scikit-learn library.

```python
class pyod.models.iforest.IForest(n_estimators=100, max_samples='auto', contamination=0.1, max_features=1.0, bootstrap=False, n_jobs=1, behaviour='old', 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.

• **behaviour (str, default='old')** – Behaviour of the `decision_function` which can be either ‘old’ or ‘new’. Passing `behaviour='new'` makes the `decision_function` change to match other anomaly detection algorithm API which will be the default behaviour in the future. As explained in details in the `offset_` attribute documentation, the `decision_function` becomes dependent on the contamination parameter, in such a way that 0 becomes its natural threshold to detect outliers.

New in version 0.7.0: `behaviour` is added in 0.7.0 for back-compatibility purpose.

Deprecated since version 0.20: `behaviour='old'` is deprecated in sklearn 0.20 and will not be possible in 0.22.

Deprecated since version 0.22: `behaviour` parameter will be deprecated in sklearn 0.22 and removed in 0.24.

| Warning: | Only applicable for sklearn 0.20 above.

• **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_**
  - The collection of fitted sub-estimators.
  - Type list of DecisionTreeClassifier

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

- **max_samples_**
  - The actual number of samples
  - Type integer

- **decision_scores_**
  - The outlier scores of the training data. The higher, the more abnormal. Outliers tend to have higher scores. This value is available once the detector is fitted.
  - Type numpy array of shape (n_samples,)

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

- **labels_**
  - The binary labels of the training data. 0 stands for inliers and 1 for outliers/anomalies. It is generated by applying `threshold_` on `decision_scores_`.
  - Type int, either 0 or 1
**decision_function** (*X*)

Predict raw anomaly 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*)

DEPRECATED

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

- **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).

**outlier_labels** [(*numpy array of shape (n_samples,)*)] 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.

Deprecated since version 0.6.9: *fit_predict* will be removed in pyod 0.7.2.; it will be replaced by calling *fit* function first and then accessing *labels_* attribute for consistency.

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

DEPRECATED

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

- **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

**score** : float

Deprecated since version 0.6.9: *fit_predict_score* will be removed in pyod 0.7.2.; it will be replaced by calling *fit* function first and then accessing *labels_* attribute for consistency. Scoring could be done by calling an evaluation method, e.g., AUC ROC.
**get_params** *(deep=True)*  
Get parameters for this estimator.


**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.


**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']

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

- \text{metric_params} (dict, optional (default = None)) – Additional keyword arguments for the metric function.

- \text{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.

\text{decision_scores_}

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

Type numpy array of shape (n_samples,)

\text{threshold_}

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

Type float

\text{labels_}

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

Type int, either 0 or 1

\text{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,)

\text{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).

\text{fit_predict} (X, y=None)

DEPRECATED

Chapter 2. API Cheatsheet & Reference
Fit detector first and then predict whether a particular sample is an outlier or not.

- **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).
- **outlier_labels**: [numpy array of shape (n_samples,)] 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.

Deprecation since version 0.6.9: `fit_predict` will be removed in pyod 0.7.2.; it will be replaced by calling `fit` function first and then accessing `labels_` attribute for consistency.

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

**DEPRECATED**

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

- **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
- **score**: float

Deprecation since version 0.6.9: `fit_predict_score` will be removed in pyod 0.7.2.; it will be replaced by calling `fit` function first and then accessing `labels_` attribute for consistency. Scoring could be done by calling an evaluation method, e.g., AUC ROC.

**get_params** *(deep=True)*

Get parameters for this estimator.


- **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.


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']

    See the documentation for scipy.spatial.distance for details on these metrics: [http://docs.scipy.org/doc/scipy/reference/spatial.distance.html](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](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_**
  The actual number of neighbors used for kneighbors queries.
  
  **decision_scores_**
  The outlier scores of the training data. The higher, the more abnormal. Outliers tend to have higher scores. This value is available once the detector is fitted.

  **threshold_**
  The threshold is based on contamination. It is the n_samples * contamination most abnormal samples in decision_scores_. The threshold is calculated for generating binary outlier labels.
labels_
The binary labels of the training data. 0 stands for inliers and 1 for outliers/anomalies. It is generated by applying `threshold_` on `decision_scores_`.

Type int, either 0 or 1
decision_function(X)
Predict raw anomaly 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,)

class
fit
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
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).

outlier_labels
-numpy array of shape (n_samples,) 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.

Deprecation: Since version 0.6.9: `fit_predict` will be removed in pyod 0.7.2.; it will be replaced by calling `fit` function first and then accessing `labels_` attribute for consistency.

fit_predict_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
score : float

Deprecated since version 0.6.9: `fit_predict_score` will be removed in pyod 0.7.2.; it will be replaced by calling `fit` function first and then accessing `labels_` attribute for consistency. Scoring could be done by calling an evaluation method, e.g., AUC ROC.

**get_params**(deep=True)

Get parameters for this estimator.


**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.


**Returns**

- **self**

**Return type**

object
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_**
- The outlier scores of the training data. The higher, the more abnormal. Outliers tend to have higher scores. This value is available once the detector is fitted.
  
  Type  numpy array of shape (n_samples,)

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

**labels_**
- The binary labels of the training data. 0 stands for inliers and 1 for outliers/anomalies. It is generated by applying threshold_ on decision_scores_.
  
  Type  int, either 0 or 1

Examples

```python
>>> 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()
```
decision_function(X)

Predict raw anomaly scores of X using the fitted detector.

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

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

DEPRECATED

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

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).

outlier_labels [numpy array of shape (n_samples,)] 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.

Deprecated since version 0.6.9: fit_predict will be removed in pyod 0.7.2.; it will be replaced by calling fit function first and then accessing labels_attribute for consistency.

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

DEPRECATED

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

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

score : float
Deprecated since version 0.6.9: `fit_predict_score` will be removed in pyod 0.7.2.; it will be replaced by calling `fit` function first and then accessing `labels_` attribute for consistency. Scoring could be done by calling an evaluation method, e.g., AUC ROC.

**get_params**(deep=True)
Get parameters for this estimator.


- **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.


- **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 [BZNHL19] 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_**

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

Type  numpy array of shape (n_samples,)

**threshold_**

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

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

Type: int, either 0 or 1

**Examples**

```python
>>> 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)
```

**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)*
DEPRECATED

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).

- **outlier_labels** *(numpy array of shape (n_samples,))* – 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.
Deprecated since version 0.6.9: `fit_predict` will be removed in pyod 0.7.2.; it will be replaced by calling `fit` function first and then accessing `labels_` attribute for consistency.

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

DEPRECATED

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

- $X$ [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
- `score` : float

Deprecated since version 0.6.9: `fit_predict_score` will be removed in pyod 0.7.2.; it will be replaced by calling `fit` function first and then accessing `labels_` attribute for consistency. Scoring could be done by calling an evaluation method, e.g., AUC ROC.

**get_params** $(deep=True)$

Get parameters for this estimator.


- **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.

**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.

**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’. 

2.5. API Reference 55
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.


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

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

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

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

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

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

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

decision_scores_
The outlier scores of the training data. The higher, the more abnormal. Outliers tend to have higher scores. This value is available once the detector is fitted. Mahalanobis distances of the training set (on which :meth:`fit` is called) observations.
Type numpy array of shape (n_samples,)

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

labels_
The binary labels of the training data. 0 stands for inliers and 1 for outliers/anomalies. It is generated by applying threshold on decision_scores_.
Type int, either 0 or 1

decision_function (X)
Predict raw anomaly 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)
DEPRECATED

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

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).

outlier_labels [numpy array of shape (n_samples,)] 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.

Deprecated since version 0.6.9: fit_predict will be removed in pyod 0.7.2.; it will be replaced by calling fit function first and then accessing labels_ attribute for consistency.

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

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

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

score : float

Deprecated since version 0.6.9: fit_predict_score will be removed in pyod 0.7.2.; it will be replaced by calling fit function first and then accessing labels_ attribute for consistency. Scoring could be done by calling an evaluation method, e.g., AUC ROC.

get_params (deep=True)
Get parameters for this estimator.


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.


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 ($k=10$, 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+19].
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.

• `k (int, optional (default=10))` – The number of sub generators.

• `stop_epochs (int, optional (default=20))` – The number of epochs of training.

• `lr_d (float, optional (default=0.01))` – The learn rate of the discriminator.

• `lr_g (float, optional (default=0.0001))` – The learn rate of the generator.

• `decay (float, optional (default=1e-6))` – The decay parameter for SGD.

• `momentum (float, optional (default=0.9))` – The momentum parameter for SGD.

decision_scores_

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

Type: `numpy array of shape (n_samples,)`

threshold_

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

Type: `float`

labels_

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

Type: `int`, either 0 or 1

decision_function(X)

Predict raw anomaly 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)

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

- **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).

**outlier_labels** [numpy array of shape (n_samples,)] 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.

Deprecated since version 0.6.9: **fit_predict** will be removed in pyod 0.7.2.; it will be replaced by calling **fit** function first and then accessing **labels_** attribute for consistency.

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

DEPRECATED

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

- **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

**score : float**

Deprecated since version 0.6.9: **fit_predict_score** will be removed in pyod 0.7.2.; it will be replaced by calling **fit** function first and then accessing **labels_** attribute for consistency. Scoring could be done by calling an evaluation method, e.g., AUC ROC.

**get_params**(deep=True)

Get parameters for this estimator.


**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.


**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)*


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

Estimate the support of a high-dimensional distribution.


**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.

**support_**
- Indices of support vectors.
  - Type: array-like, shape = [n_SV]

**support_vectors_**
- Support vectors.
  - Type: array-like, shape = [nSV, n_features]

**dual_coef_**
- Coefficients of the support vectors in the decision function.
  - Type: array, shape = [1, n_SV]

**coef_**
- Weights assigned to the features (coefficients in the primal problem). This is only available in the case of a linear kernel.
  - `coef_` is a readonly property derived from `dual_coef_` and `support_vectors_`
  - Type: array, shape = [1, n_features]

**intercept_**
- Constant in the decision function.
  - Type: array, shape = [1,]

**decision_scores_**
- The outlier scores of the training data. The higher, the more abnormal. Outliers tend to have higher scores. This value is available once the detector is fitted.
  - Type: numpy array of shape (n_samples,)

**threshold_**
- The threshold is based on `contamination`. It is the `n_samples + contamination` most abnormal samples in `decision_scores_`. The threshold is calculated for generating binary outlier labels.
  - Type: float

**labels_**
- The binary labels of the training data. 0 stands for inliers and 1 for outliers/anomalies. It is generated by applying `threshold_` on `decision_scores_`.
  - Type: int, either 0 or 1

**decision_function (X)**
- Predict raw anomaly 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)*

DEPRECATED

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

- **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).

**outlier_labels** (*numpy array of shape (n_samples,)*) 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.

Deprecation since version 0.6.9: **fit_predict** will be removed in pyod 0.7.2.; it will be replaced by calling **fit** function first and then accessing **labels_** attribute for consistency.

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

DEPRECATED

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

- **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

**score** : float

Deprecation since version 0.6.9: **fit_predict_score** will be removed in pyod 0.7.2.; it will be replaced by calling **fit** function first and then accessing **labels_** attribute for consistency. Scoring could be done by calling an evaluation method, e.g., AUC ROC.

**get_params** *(deep=True)*

Get parameters for this estimator.

**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.


**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:

  \[ n_{\text{components}} = \min(n_{\text{samples}}, n_{\text{features}}) \]

  If `n_components` is not set all components are kept. If `n_components` is set to ‘mle’ and `svd_solver` is ‘full’, Minka’s MLE is used to guess the dimension if \(0 < n_{\text{components}} < 1\) and `svd_solver` is ‘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` is ‘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 \(0 < n_{\text{components}} < X.\text{shape}[1]\)

  **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 small 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](http://scikit-learn.org/stable/auto_examples/preprocessing/plot_scaling_importance.html)

components_

Principal axes in feature space, representing the directions of maximum variance in the data. The components are sorted by explained_variance_.

Type array, shape (n_components, n_features)

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.

Type array, shape (n_components,)

explained_variance_ratio_

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.

Type array, shape (n_components,)

singular_values_

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.

Type array, shape (n_components,)

mean_

Per-feature empirical mean, estimated from the training set.

Equal to X.mean(axis=0).

Type array, shape (n_features,)

n_components_

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.

Type int

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 \( X \).

**Type**  float

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

**Type**  numpy array of shape (n_samples,)

**threshold_**
The threshold is based on contamination. It is the \( n\_samples \times \text{contamination} \) most abnormal samples in \( \text{decision\_scores\_} \). The threshold is calculated for generating binary outlier labels.

**Type**  float

**labels_**
The binary labels of the training data. 0 stands for inliers and 1 for outliers/anomalies. It is generated by applying \( \text{threshold\_} \) on \( \text{decision\_scores\_} \).

**Type**  int, either 0 or 1

**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)
DEPRECATED

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

**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).

outlier_labels [numpy array of shape (n_samples,)] 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.

Deprecated since version 0.6.9: fit_predict will be removed in pyod 0.7.2.; it will be replaced by calling fit function first and then accessing labels_ attribute for consistency.

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

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

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

score : float
Depreciated since version 0.6.9: fit_predict_score will be removed in pyod 0.7.2.; it will be replaced by calling fit function first and then accessing labels_ attribute for consistency. Scoring could be done by calling an evaluation method, e.g., AUC ROC.

get_params (deep=True)
Get parameters for this estimator.


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  

Returns  
sel

Return type  object

pyod.models.sod module

Subspace Outlier Detection (SOD)

class  
pyod.models.sod.SOD(contamination=0.1, n_neighbors=20, ref_set=10, alpha=0.8)
Bases: pyod.models.base.BaseDetector

Subspace outlier detection (SOD) schema aims to detect outlier in varying subspaces of a high dimensional feature space. For each data object, SOD explores the axis-parallel subspace spanned by the data object’s neighbors and determines how much the object deviates from the neighbors in this subspace.

See  
[BKKrogerSZ09] for details.

Parameters

- **n_neighbors**  
  (int, optional (default=20)) – Number of neighbors to use by default for k neighbors queries.
- **ref_set**  
  (int, optional (default=10)) – specifies the number of shared nearest neighbors to create the reference set. Note that ref_set must be smaller than n_neighbors.
- **alpha**  
  (float in (0., 1.), optional (default=0.8)) – specifies the lower limit for selecting subspace. 0.8 is set as default as suggested in the original paper.
- **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_
   The outlier scores of the training data. The higher, the more abnormal. Outliers tend to have higher scores. This value is available once the detector is fitted.
   
   **Type**  numpy array of shape (n_samples,)

threshold_
   The threshold is based on contamination. It is the n_samples * contamination most abnor-
   mal samples in decision_scores_. The threshold is calculated for generating binary outlier labels.
   
   **Type**  float

labels_
   The binary labels of the training data. 0 stands for inliers and 1 for outliers/anomalies. It is generated by applying threshold_ on decision_scores_.
   
   **Type**  int, either 0 or 1

decision_function(X)
   Predict raw anomaly 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.

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)
   DEPRECATED
   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).

   **outlier_labels** [numpy array of shape (n_samples,)] 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.

   Deprecated since version 0.6.9: fit_predict will be removed in pyod 0.7.2.; it will be replaced by calling fit function first and then accessing labels_ attribute for consistency.

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

score : float
Deprecated since version 0.6.9: fit_predict_score will be removed in pyod 0.7.2.; it will be replaced by calling fit function first and then accessing labels_attribute for consistency. Scoring could be done by calling an evaluation method, e.g., AUC ROC.

get_params (deep=True)
Get parameters for this estimator.

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.

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

```python
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+19].
```

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

**decision_scores_**

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

Type numpy array of shape (n_samples,)

**threshold_**

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

Type float

**labels_**

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

Type int, either 0 or 1

**decision_function(X)**

Predict raw anomaly 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`)

DEPRECATED

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

**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).

**outlier_labels** (*numpy array of shape (n_samples,)*) 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.

Deprecates since version 0.6.9: *fit_predict* will be removed in pyod 0.7.2.; it will be replaced by calling *fit* function first and then accessing *labels_* attribute for consistency.

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

DEPRECATED

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

**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

**score** : float

Deprecates since version 0.6.9: *fit_predict_score* will be removed in pyod 0.7.2.; it will be replaced by calling *fit* function first and then accessing *labels_* attribute for consistency. Scoring could be done by calling an evaluation method, e.g., AUC ROC.

**get_params** (**deep**=`True`)

Get parameters for this estimator.
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.


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](https://github.com/jeroenjanssens/scikit-sos)

class **pyod.models.sos.SOS** *(contamination=0.1, perplexity=4.5, 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** *(float, optional (default=4.5)) – A smooth measure of the effective number of neighbors. The perplexity parameter is similar to the parameter k in kNN algorithm (the number of nearest neighbors). The range of perplexity can be any real number between 1 and n-1, where n is the number of samples.*

- **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’


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_**

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

Type numpy array of shape (n_samples,)

**threshold_**

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

Type float

**labels_**

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

Type int, either 0 or 1

**Examples**

```python
>>> from pyod.models.sos import SOS
>>> 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 = SOS()
>>> clf.fit(X_train)

SOS(contamination=0.1, eps=1e-05, metric='euclidean', perplexity=4.5)

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\((X, y=None)\)

DEPRECATED

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).

outlier_labels [numpy array of shape \((n\_samples,)\)] 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.

Deprecated since version 0.6.9: fit_predict will be removed in pyod 0.7.2.; it will be replaced by calling fit function first and then accessing labels_ attribute for consistency.

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

DEPRECATED

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
score : float

Deprecated since version 0.6.9: fit_predict_score will be removed in pyod 0.7.2; it will be replaced by calling fit function first and then accessing labels_ attribute for consistency. Scoring could be done by calling an evaluation method, e.g., AUC ROC.

get_params (deep=True)
Get parameters for this estimator.


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 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.


Returns self

Return type object
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.

The number of unsupervised of detectors used.

Type int

The XGBoost classifier.

Type object

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

Type numpy array of shape (n_samples,)

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

Type int, either 0 or 1

Predict raw anomaly scores of X using the fitted detector.

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

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

DEPRECATED

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

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).

outlier_labels [numpy array of shape (n_samples,)] 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.

Deprecated since version 0.6.9: fit_predict will be removed in pyod 0.7.2.; it will be replaced by calling fit function first and then accessing labels_attribute for consistency.

**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.


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.
pyod Documentation, Release 0.7.3

**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.


**Returns** self

**Return type** object

**Module contents**

**References**

**2.5.2 Utility Functions**

**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.

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, behaviour='old', 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.
- **behaviour** (str, default='old') – Behaviour of the returned datasets which can be either ‘old’ or ‘new’. Passing behaviour='new' returns “X_train, y_train, X_test, y_test”, while passing behaviour='old' returns “X_train, X_test, y_train, y_test”.

New in version 0.7.0: behaviour is added in 0.7.0 for back-compatibility purpose.

Deprecated since version 0.7.0: behaviour='old' is deprecated in 0.20 and will not be possible in 0.7.2.

Deprecated since version 0.7.2: behaviour parameter will be deprecated in 0.7.2 and removed in 0.7.4.
**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.

```python
pyod.utils.data.generate_data_clusters(n_train=1000, n_test=500, n_clusters=2,
n_features=2, contamination=0.1, size='same',
density='same', dist=0.25, random_state=None,
return_in_clusters=False)
```

**Utility function to generate synthesized data in clusters.** Generated data can involve the low density pattern problem and global outliers which are considered as difficult tasks for outliers detection algorithms.

**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_clusters** (int, optional (default=2)) – The number of centers (i.e. clusters) to generate.
- **n_features** (int, optional (default=2)) – The number of features for each sample.
- **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.
- **size** (str, optional (default='same')) – Size of each cluster: ‘same’ generates clusters with same size, ‘different’ generate clusters with different sizes.
- **density** (str, optional (default='same')) – Density of each cluster: ‘same’ generates clusters with same density, ‘different’ generate clusters with different densities.
- **dist** (float, optional (default=0.25)) – Distance between clusters. Should be between 0. and 1.0 It is used to avoid clusters overlapping as much as possible. However, if number of samples and number of clusters are too high, it is unlikely to separate them fully even if dist set to 1.0
- **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.
- **return_in_clusters** (bool, optional (default=False)) – If True, the function returns x_train, y_train, x_test, y_test each as a list of numpy arrays where each index represents a cluster. If False, it returns x_train, y_train, x_test, y_test each as numpy array after joining the sequence of clusters arrays.

**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.example module**

Utility functions for running examples

`pyod.utils.example.data_visualize(X_train, y_train, show_figure=True, save_figure=False)`  
Utility function for visualizing the synthetic samples generated by generate_data_cluster function.  

**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.*
- **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.*

`pyod.utils.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.*

**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](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,)

```python
def 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,)
```

**Examples**

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

```python
def 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.
```

```python
>>> from pyod.utils.utility import invert_order
>>> scores = [0.1, 0.5, 0.3, 0.2, 0.7]
>>> invert_order(scores)
array([0.7, 0.5, 0.3, 0.2, 0.1])
```
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

```python
>>> 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 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

2.5. API Reference 89
• 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

2.5.3 Module contents

2.6 Known Issues & Warnings

This is the central place to track known issues.

2.6.1 Installation

There are some known dependency issues/notes. Refer installation for more information.

2.6.2 Neural Networks

SO_GAAL and MO_GAAL may only work under Python 3.5+.

2.6.3 Differences between PyOD and scikit-learn

Although PyOD is built on top of scikit-learn and inspired by its API design, some differences should be noted:

• All models in PyOD follow the tradition that the outlying objects come with higher scores while the normal
  objects have lower scores. scikit-learn has an inverted design–lower scores stand for outlying objects.

• PyOD uses “0” to represent inliers and “1” to represent outliers. Differently, scikit-learn returns “-1” for ana-
  malies/outliers and “1” for inliers.

• Although Isolation Forests, One-class SVM, and Local Outlier Factor are implemented in both PyOD and scikit-
  learn, users are not advised to mix the use of them, e.g., calling one model from PyOD and another model
  from scikit-learn. It is recommended to only use one library for consistency (for three models, the PyOD
  implementation is indeed a set of wrapper functions of scikit-learn).

• PyOD models may not work with scikit-learn’s check_estimator function. Similarly, scikit-learn models would
  not work with PyOD’s check_estimator function.

2.7 Outlier Detection 101

Outlier detection broadly refers to the task of identifying observations which may be considered anomalous given the
distribution of a sample. Any observation belonging to the distribution is referred to as an inlier and any outlying point
is referred to as an outlier.
In the context of machine learning, there are three common approaches for this task:

1. **Unsupervised Outlier Detection**
   - Training data (unlabelled) contains both normal and anomalous observations.
   - The model identifies outliers during the fitting process.
   - This approach is taken when outliers are defined as points that exist in low-density regions in the data.
   - Any new observations that do not belong to high-density regions are considered outliers.

2. **Semi-supervised Novelty Detection**
   - Training data consists only of observations describing normal behavior.
   - The model is fit on training data and then used to evaluate new observations.
   - This approach is taken when outliers are defined as points differing from the distribution of the training data.
   - Any new observations differing from the training data within a threshold, even if they form a high-density region, are considered outliers.

3. **Supervised Outlier Classification**
   - The ground truth label (inlier vs outlier) for every observation is known.
   - The model is fit on imbalanced training data and then used to classify new observations.
   - This approach is taken when ground truth is available and it is assumed that outliers will follow the same distribution as in the training set.
   - Any new observations are classified using the model.

The algorithms found in **PyOD** focus on the first two approaches which differ in terms of how the training data is defined and how the model’s outputs are interpreted. If interested in learning more, please refer to our **Anomaly Detection Resources** page for relevant related books, papers, videos, and toolboxes.

### 2.8 Citations & Achievements

#### 2.8.1 Citing PyOD

PyOD paper is accepted at JMLR (machine learning open-source software track). If you use PyOD in a scientific publication, we would appreciate citations to the following paper:

```latex
@article{zhao2019pyod,
  author = {Zhao, Yue and Nasrullah, Zain and Li, Zheng},
  title = {PyOD: A Python Toolbox for Scalable Outlier Detection},
  journal = {Journal of Machine Learning Research},
  year = {2019},
  volume = {20},
  number = {96},
  pages = {1-7},
  url = {http://jmlr.org/papers/v20/19-011.html}
}
```

or:

---

2.8. Citations & Achievements 91
2.8.2 Scientific Work Using or Referencing PyOD

We are appreciated that PyOD has been increasing referred and cited in scientific works. An incomplete list is provided below:

2019


Zhao, Y., Hryniewicki, M.K., Nasrullah, Z., and Li, Z., 2019. LSCP: Locally Selective Combination in Parallel Outlier Ensembles. *SIAM International Conference on Data Mining (SDM)*, SIAM.

2018


Zhao, Y. and Hryniewicki, M.K., 2018, July. XGBOD: improving supervised outlier detection with unsupervised representation learning. In *2018 International Joint Conference on Neural Networks (IJCNN) (pp. 1-8)*. IEEE.

2.8.3 Featured Posts & Achievements

PyOD has been well acknowledged by the machine learning community with a few featured posts and tutorials.

**Analytics Vidhya**: An Awesome Tutorial to Learn Outlier Detection in Python using PyOD Library

**KDnuggets**: Intuitive Visualization of Outlier Detection Methods

**Computer Vision News (March 2019)**: Python Open Source Toolbox for Outlier Detection
FLOYDHUB: Introduction to Anomaly Detection in Python

awesome-machine-learning: General-Purpose Machine Learning

Workshop/Showcase using PyOD:

- Detecting the Unexpected: An Introduction to Anomaly Detection Methods, *KISS Technosignatures Workshop* by Dr. Kiri Wagstaff @ Jet Propulsion Laboratory, California Institute of Technology. [Workshop Video] [PDF]

GitHub Python Trending:

- 2019: Apr 5th-6th, Feb 10th-11th, Jan 23th-24th, Jan 10th-14th
- 2018: Jun 15, Dec 8th-9th

Miscellaneous:

- PythonAwesome
- awesome-python
- PapersWithCode

## 2.9 Frequently Asked Questions

### 2.9.1 What is the Next?

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

- GPU support
- Installation efficiency improvement, such as using docker
- Add contact channel with Gitter
- Support additional languages, see Manage Translations
- Fix the bug that numba enabled function may be excluded from code coverage
- Decide which Python interpreter should readthedocs use. 3.X invokes Python 3.7 which has no TF supported for now.

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

### 2.9.2 How to Contribute

You are welcome to contribute to this exciting project:

- 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 to development branch and follow the pull request template PR template
- Automatic tests will be triggered. Make sure all tests are passed. Please make sure all added modules are accompanied with proper test functions.
To make sure the code has the same style and standard, please refer to abod.py, hbos.py, or feature_bagging.py for example.

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

2.9.3 Inclusion Criteria

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

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

2.10 About us

2.10.1 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)
Yahya Almardeny (joined in 2019): LinkedIn (Yahya Almardeny)

References


p

pyod, 90
pyod.models, 82
pyod.models.abod, 18
pyod.models.auto_encoder, 21
pyod.models.base, 16
pyod.models.cblof, 25
pyod.models.cof, 28
pyod.models.combination, 31
pyod.models.feature_bagging, 32
pyod.models.hbos, 36
pyod.models.iforest, 39
pyod.models.knn, 42
pyod.models.loci, 50
pyod.models.lof, 46
pyod.models.lscp, 53
pyod.models.mcd, 56
pyod.models.mo_gaal, 59
pyod.models.ocsvm, 62
pyod.models.pca, 65
pyod.models.so_gaal, 73
pyod.models.sod, 70
pyod.models.sos, 75
pyod.models.xgbod, 79
pyod.utils.data, 82
pyod.utils.example, 85
pyod.utils.stat_models, 86
pyod.utils.utility, 87
Index

A
ABOD (class in pyod.models.abod), 18
aom() (in module pyod.models.combination), 31
argmaxn() (in module pyod.utils.utility), 87
AutoEncoder (class in pyod.models.auto_encoder), 21
average() (in module pyod.models.combination), 31

B
BaseDetector (class in pyod.models.base), 16
bin_edges_ (pyod.models.hbos.HBOS attribute), 36

C
CBLOF (class in pyod.models.cblof), 25
check_consistent_shape() (in module pyod.utils.data), 82
check_detector() (in module pyod.utils.utility), 87
check_parameter() (in module pyod.utils.utility), 87
clf_ (pyod.models.xgbod.XGBOD attribute), 80
cluster_centers_ (pyod.models.cblof.CBLOF attribute), 26
cluster_labels_ (pyod.models.cblof.CBLOF attribute), 26
cluster_sizes_ (pyod.models.cblof.CBLOF attribute), 26
clustering_estimator_ (pyod.models.cblof.CBLOF attribute), 26
coeff_ (pyod.models.ocsvm.OCSVM attribute), 63
COF (class in pyod.models.cof), 28
components_ (pyod.models.pca.PCA attribute), 67
compression_rate_ (pyod.models.auto_encoder.AutoEncoder attribute), 22
covariance_ (pyod.models.mcd.MCD attribute), 57

d
data_visualize() (in module pyod.utils.example), 85
decision_function() (pyod.models.abod.ABOD method), 19
decision_function() (pyod.models.auto_encoder.AutoEncoder method), 23
decision_function() (pyod.models.base.BaseDetector method), 16
decision_function() (pyod.models.cblof.CBLOF method), 26
decision_function() (pyod.models.cof.COF method), 29
decision_function() (pyod.models.feature_bagging.FeatureBagging method), 34
decision_function() (pyod.models.hbos.HBOS method), 37
decision_function() (pyod.models.ift.TestIForest method), 40
decision_function() (pyod.models.ift.TestLOCI method), 51
decision_function() (pyod.models.ift.TestLOF method), 48
decision_function() (pyod.models.lscp.LSCP method), 54
decision_function() (pyod.models.mcd.MCD method), 57
decision_function() (pyod.models.mo_gaal.MO_GAAL method), 60
decision_function() (pyod.models.ocsvm.OCSVM method), 63
decision_function() (pyod.models.pca.PCA method), 68
decision_function() (pyod.models.so_gaal.SOGAAL method), 73
decision_function() (pyod.models.sod.SOD method), 71
Index
pyod Documentation, Release 0.7.3

K
KNN (class in pyod.models.knn), 42

L
labels_ (pyod.models.abod.ABOD attribute), 19
labels_ (pyod.models.auto_encoder.AutoEncoder attribute), 23
labels_ (pyod.models.base.BaseDetector attribute), 16
labels_ (pyod.models.cblof.CBLOF attribute), 26
labels_ (pyod.models.cof.COF attribute), 29
labels_ (pyod.models.feature_bagging.FeatureBagging attribute), 34
labels_ (pyod.models.hbos.HBOS attribute), 37
labels_ (pyod.models.iforest.IForest attribute), 40
labels_ (pyod.models.knn.KNN attribute), 44
labels_ (pyod.models.lof.LOF attribute), 47
labels_ (pyod.models.lscp.LSCP attribute), 53
labels_ (pyod.models.mcd.MCD attribute), 57
labels_ (pyod.models.mo_gaal.MO_GAAL attribute), 60
labels_ (pyod.models.ocsvm.OCSVM attribute), 63
labels_ (pyod.models.pca.PCA attribute), 68
labels_ (pyod.models.so_gaal.SO_GAAL attribute), 73
labels_ (pyod.models.sos.SOS attribute), 71
labels_ (pyod.models.sod.SOD attribute), 76
labels_ (pyod.models.xgbod.XGBOD attribute), 80
large_cluster_labels_ (pyod.models.cblof.CBLOF attribute), 26
location_ (pyod.models.mcd.MCD attribute), 57
LOCI (class in pyod.models.loci), 50
LOF (class in pyod.models.lof), 46
LSCP (class in pyod.models.lscp), 53

M
max_samples_ (pyod.models.iforest.IForest attribute), 40
maximization_ (in module pyod.models.combination), 32
MCD (class in pyod.models.mcd), 56
mean_ (pyod.models.pca.PCA attribute), 67
MO_GAAL (class in pyod.models.mo_gaal), 59
moa_ (in module pyod.models.combination), 32
model_ (pyod.models.auto_encoder.AutoEncoder attribute), 22

N
n_clusters_ (pyod.models.cblof.CBLOF attribute), 26
n_components_ (pyod.models.pca.PCA attribute), 67
n_detector_ (pyod.models.xgbod.XGBOD attribute), 80
n_neighbors_ (pyod.models.cof.COF attribute), 29
n_neighbors_ (pyod.models.lof.LOF attribute), 47
noise_variance_ (pyod.models.pca.PCA attribute), 67, 69

O
OCSVM (class in pyod.models.ocsvm), 62

P
pairwise_distances_no_broadcast() (in module pyod.utils.stat_models), 86
PCA (class in pyod.models.pca), 65
pearsonr_mat() (in module pyod.utils.stat_models), 86
precision_ (pyod.models.mcd.MCD attribute), 57
precision_n_scores() (in module pyod.utils.utility), 89
predict() (pyod.models.abod.ABOD method), 20
predict() (pyod.models.auto_encoder.AutoEncoder method), 24
predict() (pyod.models.base.BaseDetector method), 17
predict() (pyod.models.cblof.CBLOF method), 28
predict() (pyod.models.cof.COF method), 30
predict() (pyod.models.feature_bagging.FeatureBagging method), 35
predict() (pyod.models.hbos.HBOS method), 38
predict() (pyod.models.iforest.IForest method), 42
predict() (pyod.models.knn.KNN method), 45
predict() (pyod.models.loci.LOCI method), 52
predict() (pyod.models.lof.LOF method), 49
predict() (pyod.models.lscp.LSCP method), 55
predict() (pyod.models.mcd.MCD method), 59
predict() (pyod.models.mo_gaal.MO_GAAL method), 61
predict() (pyod.models.ocsvm.OCSVM method), 65
predict() (pyod.models.pca.PCA method), 69
predict() (pyod.models.so_gaal.SO_GAAL method), 75
predict() (pyod.models.sod.SOD method), 72
predict() (pyod.models.sos.SOS method), 78
predict() (pyod.models.xgbod.XGBOD method), 81
predict_proba() (pyod.models.abod.ABOD method), 20
predict_proba() (pyod.models.auto_encoder.AutoEncoder method), 24
predict_proba() (pyod.models.base.BaseDetector method), 18
predict_proba() (pyod.models.cblof.CBLOF method), 28
predict_proba() (pyod.models.cof.COF method), 30
predict_proba() (pyod.models.feature_bagging.FeatureBagging method), 35
threshold_ (pyod.models.auto_encoder.AutoEncoder attribute), 22
threshold_ (pyod.models.base.BaseDetector attribute), 16
threshold_ (pyod.models.cblof.CBLOF attribute), 26
threshold_ (pyod.models.cof.COF attribute), 29
threshold_ (pyod.models.feature_bagging.FeatureBagging attribute), 34
threshold_ (pyod.models.hbos.HBOS attribute), 37
threshold_ (pyod.models.ifs.IForest attribute), 40
threshold_ (pyod.models.knn.KNN attribute), 44
threshold_ (pyod.models.loci.LOCI attribute), 50
threshold_ (pyod.models.iforest.IForest attribute), 40
threshold_ (pyod.models.loci.LOCI attribute), 50
threshold_ (pyod.models.mcd.MCD attribute), 57
threshold_ (pyod.models.mo_gaal.MO_GAAL attribute), 60
threshold_ (pyod.models.ocsvm.OCSVM attribute), 63
threshold_ (pyod.models.pca.PCA attribute), 68
threshold_ (pyod.models.so_gaal.SO_GAAL attribute), 73
threshold_ (pyod.models.sod.SOD attribute), 71
threshold_ (pyod.models.sos.SOS attribute), 76

V
visualize() (in module pyod.utils.example), 85

W
wpearsonr() (in module pyod.utils.stat_models), 86

X
XGBOD (class in pyod.models.xgbod), 79