# Getting Started

1 Implemented Algorithms

2 API Cheatsheet & Reference
   2.1 Installation .................................................. 7
   2.2 Model Save & Load ............................................ 8
   2.3 Fast Train with SUOD ....................................... 9
   2.4 Examples ....................................................... 9
   2.5 Benchmarks ................................................... 13
   2.6 API CheatSheet .............................................. 16
   2.7 API Reference ............................................... 20
   2.8 Known Issues & Warnings ................................ 148
   2.9 Outlier Detection 101 ...................................... 149
   2.10 Citations & Achievements ................................. 150
   2.11 Frequently Asked Questions .............................. 151
   2.12 About us ...................................................... 152

Bibliography 155

Python Module Index 159

Index 161
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.

PyOD includes more than 30 detection algorithms, from classical LOF (SIGMOD 2000) to the latest COPOD (ICDM 2020) and SUOD (MLSys 2021). Since 2017, PyOD [AZNL19] has been successfully used in numerous academic researches and commercial products [AZHC+21, AZNHL19]. It is also well acknowledged by the machine learning community with various dedicated posts/tutorials, including Analytics Vidhya, Towards Data Science, 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 classical ones from scikit-learn, latest deep learning methods, and emerging algorithms like COPOD.
- **Optimized performance with JIT and parallelization** when possible, using numba and joblib.
- **Fast training & prediction with SUOD** [AZHC+21].
- **Compatible with both Python 2 & 3**.

API Demo:

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

# get outlier scores
y_train_scores = clf.decision_scores_  # raw outlier scores on the train data
y_test_scores = clf.decision_function(X_test)  # predict raw outlier scores on test
```

Citing PyOD:

PyOD paper is published in Journal of Machine Learning Research (JMLR) (MLOSS track). If you use PyOD in a scientific publication, we would appreciate citations to the following paper:
Key Links and Resources:

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

IMPLEMENTED ALGORITHMS

PyOD toolkit consists of three major functional groups:

(i) Individual Detection Algorithms:

<table>
<thead>
<tr>
<th>Type</th>
<th>Abbr</th>
<th>Algorithm</th>
</tr>
</thead>
<tbody>
<tr>
<td>Probabilistic</td>
<td>ECOD</td>
<td>Unsupervised Outlier Detection Using Empirical Cumulative Distribution Functions</td>
</tr>
<tr>
<td>Probabilistic</td>
<td>COPOD</td>
<td>COPOD: Copula-Based Outlier Detection</td>
</tr>
<tr>
<td>Probabilistic</td>
<td>ABOD</td>
<td>Angle-Based Outlier Detection</td>
</tr>
<tr>
<td>Probabilistic</td>
<td>FastABOD</td>
<td>Fast Angle-Based Outlier Detection using approximation</td>
</tr>
<tr>
<td>Probabilistic</td>
<td>MAD</td>
<td>Median Absolute Deviation (MAD)</td>
</tr>
<tr>
<td>Probabilistic</td>
<td>SOS</td>
<td>Stochastic Outlier Selection</td>
</tr>
<tr>
<td>Linear Model</td>
<td>PCA</td>
<td>Principal Component Analysis (the sum of weighted projected distances to the eigenvector hyperplanes)</td>
</tr>
<tr>
<td>Linear Model</td>
<td>MCD</td>
<td>Minimum Covariance Determinant (use the mahalanobis distances as the outlier scores)</td>
</tr>
<tr>
<td>Linear Model</td>
<td>OCSVM</td>
<td>One-Class Support Vector Machines</td>
</tr>
<tr>
<td>Proximity-Based</td>
<td>LOF</td>
<td>Local Outlier Factor</td>
</tr>
<tr>
<td>Proximity-Based</td>
<td>COF</td>
<td>Connectivity-Based Outlier Factor (slower but reduce storage complexity)</td>
</tr>
<tr>
<td>Proximity-Based</td>
<td>Incr. COF</td>
<td>Memory Efficient Connectivity-Based Outlier Factor (slower but reduce storage complexity)</td>
</tr>
<tr>
<td>Proximity-Based</td>
<td>CBLOF</td>
<td>Clustering-Based Local Outlier Factor</td>
</tr>
<tr>
<td>Proximity-Based</td>
<td>LOCI</td>
<td>LOCI: Fast outlier detection using the local correlation integral</td>
</tr>
<tr>
<td>Proximity-Based</td>
<td>HBOS</td>
<td>Histogram-based Outlier Score</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>
</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>
</tr>
<tr>
<td>Proximity-Based</td>
<td>MedKNN</td>
<td>Median kNN (use the median distance to k nearest neighbors as the outlier score)</td>
</tr>
<tr>
<td>Proximity-Based</td>
<td>SOD</td>
<td>Subspace Outlier Detection</td>
</tr>
<tr>
<td>Proximity-Based</td>
<td>ROD</td>
<td>Rotation-based Outlier Detection</td>
</tr>
<tr>
<td>Outlier Ensembles</td>
<td>IForest</td>
<td>Isolation Forest</td>
</tr>
<tr>
<td>Outlier Ensembles</td>
<td>FB</td>
<td>Feature Bagging</td>
</tr>
<tr>
<td>Outlier Ensembles</td>
<td>LSCP</td>
<td>LSCP: Locally Selective Combination of Parallel Outlier Ensembles</td>
</tr>
<tr>
<td>Outlier Ensembles</td>
<td>XGBOD</td>
<td>Extreme Boosting Based Outlier Detection (Supervised)</td>
</tr>
<tr>
<td>Outlier Ensembles</td>
<td>LODA</td>
<td>Lightweight On-line Detector of Anomalies</td>
</tr>
<tr>
<td>Outlier Ensembles</td>
<td>SUOD</td>
<td>SUOD: Accelerating Large-scale Unsupervised Heterogeneous Outlier Detection (Acceleration)</td>
</tr>
<tr>
<td>Neural Networks</td>
<td>AutoEncoder</td>
<td>Fully connected AutoEncoder (use reconstruction error as the outlier score)</td>
</tr>
<tr>
<td>Neural Networks</td>
<td>VAE</td>
<td>Variational AutoEncoder (use reconstruction error as the outlier score)</td>
</tr>
<tr>
<td>Neural Networks</td>
<td>Beta-VAE</td>
<td>Variational AutoEncoder (all customized loss term by varying gamma and capacity)</td>
</tr>
<tr>
<td>Neural Networks</td>
<td>SO_GAAL</td>
<td>Single-Objective Generative Adversarial Active Learning</td>
</tr>
<tr>
<td>Neural Networks</td>
<td>MO_GAAL</td>
<td>Multiple-Objective Generative Adversarial Active Learning</td>
</tr>
<tr>
<td>Neural Networks</td>
<td>DeepSVDD</td>
<td>Deep One-Class Classification</td>
</tr>
</tbody>
</table>

(ii) Outlier Ensembles & Outlier Detector Combination Frameworks:
### Outlier Ensembles

<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>pyod.models.feature_bagging.FeatureBagging</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>
</tr>
<tr>
<td>Outlier Ensembles</td>
<td>XG-BOD</td>
<td>Extreme Boosting Based Outlier Detection (Supervised)</td>
<td>2018</td>
<td>pyod.models.xgbod.XGBOD</td>
</tr>
<tr>
<td>Outlier Ensembles</td>
<td>LODA</td>
<td>Lightweight On-line Detector of Anomalies</td>
<td>2016</td>
<td>pyod.models.loda.LODA</td>
</tr>
<tr>
<td>Outlier Ensembles</td>
<td>SUOD</td>
<td>SUOD: Accelerating Large-scale Unsupervised Heterogeneous Outlier Detection (Acceleration)</td>
<td>2021</td>
<td>pyod.models.suod.SUOD</td>
</tr>
</tbody>
</table>

#### Combination

<table>
<thead>
<tr>
<th>Type</th>
<th>Name</th>
<th>Function</th>
<th>Year</th>
<th>Ref</th>
</tr>
</thead>
<tbody>
<tr>
<td>Combination</td>
<td>Average</td>
<td>Simple combination by averaging the scores</td>
<td>2015</td>
<td>pyod.models.combination.average()</td>
</tr>
<tr>
<td>Combination</td>
<td>Weighted Average</td>
<td>Simple combination by averaging the scores with detector weights</td>
<td>2015</td>
<td>pyod.models.combination.average()</td>
</tr>
<tr>
<td>Combination</td>
<td>Maximization</td>
<td>Simple combination by taking the maximum scores</td>
<td>2015</td>
<td>pyod.models.combination.maximization()</td>
</tr>
<tr>
<td>Combination</td>
<td>AOM</td>
<td>Average of Maximum</td>
<td>2015</td>
<td>pyod.models.combination.aom()</td>
</tr>
<tr>
<td>Combination</td>
<td>MOA</td>
<td>Maximum of Average</td>
<td>2015</td>
<td>pyod.models.combination.moa()</td>
</tr>
<tr>
<td>Combination</td>
<td>Median</td>
<td>Simple combination by taking the median of the scores</td>
<td>2015</td>
<td>pyod.models.combination.median()</td>
</tr>
<tr>
<td>Combination</td>
<td>majority Vote</td>
<td>Simple combination by taking the majority vote of the labels (weights can be used)</td>
<td>2015</td>
<td>pyod.models.combination.majority_vote()</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>pyod.utils.data.generate_data()</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>pyod.utils.data.generate_data_clusters()</td>
<td>Synthesized data generation in clusters; more complex data patterns can be created with multiple clusters</td>
</tr>
<tr>
<td>Stat</td>
<td>pyod.utils.stat_models.wpearsonr()</td>
<td>Calculate the weighted Pearson correlation of two samples</td>
</tr>
<tr>
<td>Utility</td>
<td>pyod.utils.utility.get_label_n()</td>
<td>Turn raw outlier scores into binary labels by assign 1 to top n outlier scores</td>
</tr>
<tr>
<td>Utility</td>
<td>pyod.utils.utility.precision_n_scores()</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`.
API CHEATSHEET & REFERENCE

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

- `pyod.models.base.BaseDetector.fit()`: Fit detector. y is ignored in 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.predict_confidence()`: Predict the model’s sample-wise confidence (available in predict and predict_proba).

Key Attributes of a fitted model:

- `pyod.models.base.BaseDetector.decision_scores_`: The outliers scores of the training data. The higher, the more abnormal. Outliers tend to have higher scores.

- `pyod.models.base.BaseDetector.labels_`: The binary labels of the training data. 0 stands for inliers and 1 for outliers/anomalies.

2.1 Installation

It is recommended to use pip or conda 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
conda install -c conda-forge pyod
```

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

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

**Required Dependencies:**

- Python 2.7, 3.5, 3.6, or 3.7
• combo>=0.0.8
• joblib
• numpy>=1.13
• numba>=0.35
• scipy>=0.20.0
• scikit_learn>=0.19.1
• statsmodels

Optional Dependencies (see details below):
• combo (optional, required for models/combination.py and FeatureBagging)
• keras (optional, required for AutoEncoder, and other deep learning models)
• matplotlib (optional, required for running examples)
• pandas (optional, required for running benchmark)
• suod (optional, required for running SUOD model)
• tensorflow (optional, required for AutoEncoder, and other deep learning models)
• xgboost (optional, required for XGBOD)

**Warning:** PyOD has multiple neural network based models, e.g., AutoEncoders, which are implemented in both PyTorch and Tensorflow. However, PyOD does NOT install DL libraries 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:** 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 Model Save & Load

PyOD takes a similar approach of sklearn regarding model persistence. See model persistence for clarification.

In short, we recommend to use joblib or pickle for saving and loading PyOD models. See “examples/save_load_model_example.py” for an example. In short, it is simple as below:

```python
from joblib import dump, load

# save the model
dump(clf, 'clf.joblib')

# load the model
clf = load('clf.joblib')
```

It is known that there are challenges in saving neural network models. Check #328 and #88 for temporary workaround.
2.3 Fast Train with SUOD

**Fast training and prediction:** it is possible to train and predict with a large number of detection models in PyOD by leveraging SUOD framework. See SUOD Paper and SUOD example.

```python
from pyod.models.suod import SUOD

detector_list = [LOF(n_neighbors=15), LOF(n_neighbors=20),
                 LOF(n_neighbors=25), LOF(n_neighbors=35),
                 COPOD(), IForest(n_estimators=100),
                 IForest(n_estimators=200)]

clf = SUOD(base_estimators=detector_list, n_jobs=2, combination='average',
           verbose=False)
```

2.4 Examples

2.4.1 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

Towards Data Science: Anomaly Detection for Dummies

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

awesome-machine-learning: General-Purpose Machine Learning

2.4.2 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

# it is possible to get the prediction confidence as well
y_test_pred, y_test_pred_confidence = clf.predict(X_test, return_ ˓→confidence=True)  # outlier labels (0 or 1) and confidence in the range ˓→of [0,1]
```

4. Evaluate the prediction using ROC and Precision @ Rank n `pyod.utils.data.evaluate_print()`.

```python
from pyod.utils.data import evaluate_print

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

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

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

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

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

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

### 2.4.3 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.
2.4. Examples
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]
n_clf = len(k_list) # Number of classifiers being trained
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:

```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
```

5. Finally, all four combination methods are evaluated by ROC and Precision @ Rank n:
Combining 20 kNN detectors
Combination by Average ROC: 0.9194, precision @ rank n: 0.4531
Combination by Maximization ROC: 0.9198, precision @ rank n: 0.4688
Combination by AOM ROC: 0.9257, precision @ rank n: 0.4844
Combination by MOA ROC: 0.9263, precision @ rank n: 0.4688

References

2.5 Benchmarks

2.5.1 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>
## 2.5.2 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>OBB</th>
<th>HBOS 1Forest</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>0.768</td>
<td>0.783</td>
<td>0.778</td>
<td>0.821</td>
<td>0.786</td>
<td>0.778</td>
<td>0.779</td>
<td>0.781</td>
<td>0.7815</td>
</tr>
<tr>
<td>cardio</td>
<td>1831</td>
<td>21</td>
<td>9.6122</td>
<td>0.569</td>
<td>0.927</td>
<td>0.586</td>
<td>0.835</td>
<td>0.723</td>
<td>0.573</td>
<td>0.813</td>
<td>0.934</td>
<td>0.9504</td>
</tr>
<tr>
<td>glass</td>
<td>214</td>
<td>9</td>
<td>4.2056</td>
<td>0.795</td>
<td>0.850</td>
<td>0.872</td>
<td>0.738</td>
<td>0.850</td>
<td>0.864</td>
<td>0.790</td>
<td>0.632</td>
<td>0.6747</td>
</tr>
<tr>
<td>ionosphere</td>
<td>351</td>
<td>33</td>
<td>35.8974</td>
<td>0.924</td>
<td>0.813</td>
<td>0.873</td>
<td>0.561</td>
<td>0.949</td>
<td>0.875</td>
<td>0.955</td>
<td>0.8419</td>
<td>0.7962</td>
</tr>
<tr>
<td>letter</td>
<td>1600</td>
<td>32</td>
<td>6.2500</td>
<td>0.878</td>
<td>0.507</td>
<td>0.866</td>
<td>0.592</td>
<td>0.778</td>
<td>0.876</td>
<td>0.859</td>
<td>0.8074</td>
<td>0.6118</td>
</tr>
<tr>
<td>lympho</td>
<td>148</td>
<td>18</td>
<td>4.0541</td>
<td>0.911</td>
<td>0.972</td>
<td>0.975</td>
<td>0.995</td>
<td>0.974</td>
<td>0.977</td>
<td>0.900</td>
<td>0.9759</td>
<td>0.9847</td>
</tr>
<tr>
<td>mnist</td>
<td>7603</td>
<td>100</td>
<td>9.2069</td>
<td>0.781</td>
<td>0.800</td>
<td>0.720</td>
<td>0.574</td>
<td>0.815</td>
<td>0.848</td>
<td>0.716</td>
<td>0.8665</td>
<td>0.8527</td>
</tr>
<tr>
<td>musk</td>
<td>3062</td>
<td>166</td>
<td>3.1679</td>
<td>0.184</td>
<td>0.987</td>
<td>0.526</td>
<td>1.000</td>
<td>0.999</td>
<td>0.798</td>
<td>0.528</td>
<td>0.9998</td>
<td>1.0000</td>
</tr>
<tr>
<td>optdigits</td>
<td>5216</td>
<td>64</td>
<td>2.8758</td>
<td>0.466</td>
<td>0.508</td>
<td>0.443</td>
<td>0.873</td>
<td>0.725</td>
<td>0.370</td>
<td>0.450</td>
<td>0.3979</td>
<td>0.4997</td>
</tr>
<tr>
<td>pendigits</td>
<td>6870</td>
<td>16</td>
<td>2.2707</td>
<td>0.687</td>
<td>0.948</td>
<td>0.459</td>
<td>0.923</td>
<td>0.943</td>
<td>0.748</td>
<td>0.469</td>
<td>0.8344</td>
<td>0.9303</td>
</tr>
<tr>
<td>pima</td>
<td>768</td>
<td>8</td>
<td>34.8958</td>
<td>0.679</td>
<td>0.734</td>
<td>0.623</td>
<td>0.700</td>
<td>0.680</td>
<td>0.707</td>
<td>0.627</td>
<td>0.675</td>
<td>0.6215</td>
</tr>
<tr>
<td>satellite</td>
<td>6435</td>
<td>36</td>
<td>31.6395</td>
<td>0.571</td>
<td>0.669</td>
<td>0.557</td>
<td>0.758</td>
<td>0.702</td>
<td>0.683</td>
<td>0.557</td>
<td>0.803</td>
<td>0.6622</td>
</tr>
<tr>
<td>satimage 2</td>
<td>5803</td>
<td>36</td>
<td>1.2235</td>
<td>0.819</td>
<td>0.991</td>
<td>0.457</td>
<td>0.980</td>
<td>0.994</td>
<td>0.953</td>
<td>0.457</td>
<td>0.995</td>
<td>0.9978</td>
</tr>
<tr>
<td>shuttle</td>
<td>49097</td>
<td>9</td>
<td>7.1511</td>
<td>0.623</td>
<td>0.627</td>
<td>0.472</td>
<td>0.985</td>
<td>0.997</td>
<td>0.653</td>
<td>0.526</td>
<td>0.9903</td>
<td>0.9917</td>
</tr>
<tr>
<td>vertebral</td>
<td>240</td>
<td>6</td>
<td>12.5000</td>
<td>0.426</td>
<td>0.348</td>
<td>0.416</td>
<td>0.326</td>
<td>0.390</td>
<td>0.381</td>
<td>0.408</td>
<td>0.3905</td>
<td>0.4431</td>
</tr>
<tr>
<td>vowels</td>
<td>1456</td>
<td>12</td>
<td>3.4341</td>
<td>0.960</td>
<td>0.585</td>
<td>0.942</td>
<td>0.672</td>
<td>0.758</td>
<td>0.968</td>
<td>0.941</td>
<td>0.8075</td>
<td>0.7802</td>
</tr>
<tr>
<td>wbc</td>
<td>378</td>
<td>30</td>
<td>5.5556</td>
<td>0.904</td>
<td>0.922</td>
<td>0.932</td>
<td>0.951</td>
<td>0.931</td>
<td>0.936</td>
<td>0.934</td>
<td>0.9215</td>
<td>0.9319</td>
</tr>
</tbody>
</table>

14 Chapter 2. API Cheatsheet & Reference
### 2.5.3 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>OCSVM</th>
<th>MCD</th>
<th>LOF</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>0.4464</td>
<td>0.4334</td>
</tr>
<tr>
<td>cardio</td>
<td>1831</td>
<td>21</td>
<td>9.6122</td>
<td>0.2374</td>
<td>0.3808</td>
<td>0.4539</td>
<td>0.4230</td>
<td>0.3326</td>
<td>0.5111</td>
</tr>
<tr>
<td>glass</td>
<td>214</td>
<td>9</td>
<td>4.2065</td>
<td>0.1702</td>
<td>0.0726</td>
<td>0.0726</td>
<td>0.0726</td>
<td>0.0726</td>
<td>0.0726</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>0.3295</td>
<td>0.6369</td>
<td>0.8602</td>
<td>0.7063</td>
</tr>
<tr>
<td>letter</td>
<td>1600</td>
<td>32</td>
<td>6.2500</td>
<td>0.3801</td>
<td>0.0749</td>
<td>0.0749</td>
<td>0.0749</td>
<td>0.0749</td>
<td>0.0749</td>
</tr>
<tr>
<td>lympho</td>
<td>148</td>
<td>18</td>
<td>4.0541</td>
<td>0.4485</td>
<td>0.7517</td>
<td>0.7517</td>
<td>0.7517</td>
<td>0.7517</td>
<td>0.7517</td>
</tr>
<tr>
<td>mnist</td>
<td>7603</td>
<td>100</td>
<td>9.2069</td>
<td>0.3558</td>
<td>0.3348</td>
<td>0.3299</td>
<td>0.3462</td>
<td>0.3962</td>
<td>0.3846</td>
</tr>
<tr>
<td>musk</td>
<td>3062</td>
<td>166</td>
<td>3.1679</td>
<td>0.6507</td>
<td>0.7766</td>
<td>0.2235</td>
<td>0.9783</td>
<td>0.2735</td>
<td>0.1695</td>
</tr>
<tr>
<td>optdigits</td>
<td>5216</td>
<td>64</td>
<td>2.8758</td>
<td>0.0000</td>
<td>0.0000</td>
<td>0.0244</td>
<td>0.2194</td>
<td>0.0301</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>0.2766</td>
<td>0.0658</td>
<td>0.2979</td>
<td>0.3422</td>
<td>0.0984</td>
</tr>
<tr>
<td>pima</td>
<td>768</td>
<td>8</td>
<td>34.8958</td>
<td>0.5193</td>
<td>0.5413</td>
<td>0.4480</td>
<td>0.5424</td>
<td>0.5111</td>
<td>0.5415</td>
</tr>
<tr>
<td>satimage2</td>
<td>5803</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>0.6576</td>
<td>0.4999</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>
</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>0.0345</td>
<td>0.0238</td>
</tr>
<tr>
<td>vowels</td>
<td>1456</td>
<td>12</td>
<td>3.4341</td>
<td>0.5710</td>
<td>0.0831</td>
<td>0.3224</td>
<td>0.1297</td>
<td>0.1875</td>
<td>0.5095</td>
</tr>
<tr>
<td>wbc</td>
<td>378</td>
<td>30</td>
<td>5.5556</td>
<td>0.3064</td>
<td>0.5055</td>
<td>0.5188</td>
<td>0.5817</td>
<td>0.5085</td>
<td>0.4952</td>
</tr>
</tbody>
</table>
### 2.5.4 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>ABOD</th>
<th>CBLOF</th>
<th>FFB</th>
<th>HBOSIForest</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>0.3667</td>
<td>0.2123</td>
<td>0.5651</td>
<td>0.1383</td>
<td>0.2659</td>
<td>0.1075</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>0.1255</td>
<td>0.7741</td>
<td>0.0053</td>
<td>0.2017</td>
<td>0.0173</td>
<td>0.0025</td>
<td>0.0325</td>
<td>0.0010</td>
</tr>
<tr>
<td>glass</td>
<td>214</td>
<td>9</td>
<td>4.2056</td>
<td>0.0352</td>
<td>0.0359</td>
<td>0.0359</td>
<td>0.0317</td>
<td>0.0870</td>
<td>0.0093</td>
<td>0.0025</td>
<td>0.0014</td>
<td>0.0010</td>
</tr>
<tr>
<td>ionosphere</td>
<td>351</td>
<td>33</td>
<td>35.8974</td>
<td>0.00645</td>
<td>0.0459</td>
<td>0.0728</td>
<td>0.0080</td>
<td>0.2086</td>
<td>0.0300</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>0.2617</td>
<td>0.1882</td>
<td>0.1942</td>
<td>0.0888</td>
<td>0.0041</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>0.0030</td>
<td>0.7017</td>
<td>0.0110</td>
<td>0.0027</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>0.2479</td>
<td>0.1139</td>
<td>0.7427</td>
<td>0.0048</td>
<td>0.1934</td>
<td>0.0374</td>
<td>0.0020</td>
<td>0.0015</td>
<td>0.0010</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>13.86</td>
<td>0.0585</td>
<td>0.2017</td>
<td>0.0173</td>
<td>0.0025</td>
<td>0.0014</td>
<td>0.0010</td>
</tr>
<tr>
<td>optdigits</td>
<td>5216</td>
<td>64</td>
<td>2.8758</td>
<td>2.7279</td>
<td>0.4979</td>
<td>14.399</td>
<td>0.0303</td>
<td>0.7788</td>
<td>2.1205</td>
<td>1.8599</td>
<td>1.5618</td>
<td>0.0519</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>0.5879</td>
<td>2.2209</td>
<td>0.9666</td>
<td>0.0062</td>
<td></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>0.9098</td>
<td>0.0010</td>
<td>0.1923</td>
<td>0.0590</td>
<td>0.0010</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>0.0101</td>
<td>0.6449</td>
<td>1.2578</td>
<td>0.9868</td>
<td>2.6916</td>
<td>1.3697</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>59.213</td>
<td>10.0953</td>
<td>3.3906</td>
<td>9.4958</td>
<td>11.1500</td>
<td>0.1214</td>
<td>94.4683</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>0.0040</td>
<td>0.1786</td>
<td>0.0161</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>0.3125</td>
<td>0.0040</td>
<td>0.2758</td>
<td>0.1125</td>
<td>0.0367</td>
<td>0.9745</td>
<td>0.0469</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>0.0060</td>
<td>0.2030</td>
<td>0.0287</td>
<td>0.0078</td>
<td>0.0864</td>
<td>0.0062</td>
</tr>
</tbody>
</table>

### 2.6 API CheatSheet

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

- **pyod.models.base.BaseDetector.fit()**: Fit detector. y is ignored in 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.predict_confidence()**: Predict the model’s sample-wise confidence (available in predict and predict_proba).
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.

**Fast training and prediction**: it is possible to train and predict with a large number of detection models in PyOD by leveraging SUOD framework [BZHC+21]. See SUOD Paper and repository.

See base class definition below:

### 2.6.1 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

**abstract decision_function(X)**
- Predict raw anomaly scores of X using the fitted detector.
  - The anomaly score of an input sample is computed based on the fitted detector. For consistency, outliers are assigned with higher anomaly scores.
    - **Parameters** `X (numpy array of shape (n_samples, n_features))` – The input samples. Sparse matrices are accepted only if they are supported by the base estimator.
    - **Returns** `anomaly_scores` – The anomaly score of the input samples.
    - **Return type** `numpy array of shape (n_samples,)`

**abstract fit(X, y=None)**
- Fit detector. `y` is ignored in unsupervised methods.
  - **Parameters**

---

2.6. API CheatSheet
• **X** *(numpy array of shape \((n_{\text{samples}}, n_{\text{features}})\)) – The input samples.*

• **y** *(Ignored) – Not used, present for API consistency by convention.*

**Returns**

* `self` – Fitted estimator.

**Return type**

* `object`

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

DEPRECATED

Fit detector first and then predict whether a particular sample is an outlier or not. `y` is ignored in unsupervised models.

- **X** *(numpy array of shape \((n_{\text{samples}}, n_{\text{features}})\)) – The input samples.*
- **y** *(Ignored) – Not used, present for API consistency by convention.*

**outlier_labels** *(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.*

Deprecation since version 0.6.9: `fit_predict` will be removed in pyod 0.8.0; 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 \((n_{\text{samples}}, n_{\text{features}})\)) – The input samples.*
- **y** *(Ignored) – Not used, present for API consistency by convention.*
- **scoring** *(str, optional (default='roc_auc_score')) – Evaluation metric:*
  - `'roc_auc_score'` – ROC score
  - `'pre_n_score'` – Precision @ rank n score

**score** *(float)*

Deprecation since version 0.6.9: `fit_predict_score` will be removed in pyod 0.8.0; 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** *(bool, optional (default=True)) – 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, return_confidence=False)*

Predict if a particular sample is an outlier or not.

**Parameters**

- **X** *(numpy array of shape \((n_{\text{samples}}, n_{\text{features}})\)) – The input samples.*
• **return_confidence** *(boolean, optional(default=False)) – If True, also return the confidence of prediction.*

**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.*

• **confidence** *(numpy array of shape (n_samples,)) – Only if return_confidence is set to True.*

**predict_confidence(X)**

Predict the model’s confidence in making the same prediction under slightly different training sets. See [BPVD20].

**Parameters**

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

**Returns**

- **confidence** – For each observation, tells how consistently the model would make the same prediction if the training set was perturbed. Return a probability, ranging in [0,1].

**Return type**

- numpy array of shape (n_samples,)

**predict_proba(X, method='linear', return_confidence=False)**

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’.*

- **return_confidence** *(boolean, optional(default=False)) – If True, also return the confidence of prediction.*

**Returns**

- **outlier_probability** – 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]. Note it depends on the number of classes, which is by default 2 classes ([proba of normal, proba of outliers]).

**Return type**

- numpy array of shape (n_samples, n_classes)

**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
2.7 API Reference

2.7.1 All Models

**pyod.models.abod module**

Angle-based Outlier Detector (ABOD)

```python
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.
\textbf{Returns} anomaly\_scores – The anomaly score of the input samples.

\textbf{Return type} numpy array of shape (n\_samples,)

\textbf{fit}(X, y=\text{None})

Fit detector. \(y\) is ignored in unsupervised methods.

\textbf{Parameters}

- \textbf{X} (\text{numpy array of shape (n\_samples, n\_features)}) – The input samples.
- \textbf{y} (\text{Ignored}) – Not used, present for API consistency by convention.

\textbf{Returns} self – Fitted estimator.

\textbf{Return type} object

\textbf{fit\_predict}(X, y=\text{None})

DEPRECATED

Fit detector first and then predict \textbf{whether a particular sample} is an outlier or not. \(y\) is ignored in unsupervised models.

\textbf{X} [\text{numpy array of shape (n\_samples, n\_features)}] The input samples.

\textbf{y} [\text{Ignored}] Not used, present for API consistency by convention.

\textbf{outlier\_labels} [\text{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: \text{fit\_predict} will be removed in pyod 0.8.0.; it will be replaced by calling \text{fit} function first and then accessing \text{labels\_} attribute for consistency.

\textbf{fit\_predict\_score}(X, y, \text{scoring}='roc\_auc\_score')

DEPRECATED

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

\textbf{X} [\text{numpy array of shape (n\_samples, n\_features)}] The input samples.

\textbf{y} [\text{Ignored}] Not used, present for API consistency by convention.

\textbf{scoring} [\text{str, optional (default='roc\_auc\_score')}\text{]} Evaluation metric:

- ‘roc\_auc\_score’: ROC score
- ‘prc\_n\_score’: Precision @ rank n score

\textbf{score} : float

Deprecated since version 0.6.9: \text{fit\_predict\_score} will be removed in pyod 0.8.0.; it will be replaced by calling \text{fit} function first and then accessing \text{labels\_} attribute for consistency. Scoring could be done by calling an evaluation method, e.g., AUC ROC.

\textbf{get\_params} \textbf{(deep=True)}

Get parameters for this estimator.


\textbf{Parameters} \textbf{deep} (\text{bool, optional (default=True)}) – If True, will return the parameters for this estimator and contained subobjects that are estimators.

\textbf{Returns} \textbf{params} – Parameter names mapped to their values.
**Return type** mapping of string to any

**predict**($X$, $return\_confidence=False$)

Predict if a particular sample is an outlier or not.

**Parameters**

- $X$ (*numpy array of shape (n_samples, n_features)*) – The input samples.
- $return\_confidence$ (*boolean, optional (default=False)*) – If True, also return the confidence of prediction.

**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.
- $confidence$ (*numpy array of shape (n_samples,)*) – Only if $return\_confidence$ is set to True.

**predict\_confidence**($X$)

Predict the model’s confidence in making the same prediction under slightly different training sets. See [BPVD20].

**Parameters**

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

**Returns**

- $confidence$ – For each observation, tells how consistently the model would make the same prediction if the training set was perturbed. Return a probability, ranging in [0,1].

**Return type** *numpy array of shape (n_samples,)*

**predict\_proba**($X$, method='linear', $return\_confidence=False$)

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’.
- $return\_confidence$ (*boolean, optional (default=False)*) – If True, also return the confidence of prediction.

**Returns**

- $outlier\_probability$ – 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]. Note it depends on the number of classes, which is by default 2 classes ([proba of normal, proba of outliers]).

**Return type** *numpy array of shape (n_samples, n_classes)*

**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(hidden_neurons=None, hidden_activation='relu',
                                           output_activation='sigmoid', loss=mean_squared_error,
                                           optimizer='adam', epochs=100, batch_size=32, dropout_rate=0.2,
                                           l2_regularizer=0.1, validation_size=0.1, preprocessing=True,
                                           verbose=1, random_state=None, contamination=0.1)
```

Bases: `pyod.models.base.BaseDetector`

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

**Parameters**

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

- **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 \( n \times \text{contamination} \) 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 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 ignored in unsupervised methods.
Parameters

- **X** (*numpy array of shape (n_samples, n_features]*) – The input samples.
- **y** (*Ignored*) – Not used, present for API consistency by convention.

**Returns**
- **self** – Fitted estimator.

**Return type**
- **object**

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

DEPRECATED

**Fit detector first and then predict whether a particular sample** is an outlier or not. **y** is ignored in unsupervised models.

**X** [*numpy array of shape (n_samples, n_features)]* The input samples.

**y** [*Ignored*] Not used, present for API consistency by convention.

**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 0.8.0.; 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** [*Ignored*] Not used, present for API consistency by convention.

**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 0.8.0.; 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** (*bool, optional (default=True]*) – 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, return_confidence=False**) Predict if a particular sample is an outlier or not.

**Parameters**
- \(X\) (numpy array of shape \((n_{\text{samples}}, n_{\text{features}})\)) – The input samples.

- `return_confidence` (boolean, optional(default=False)) – If True, also return the confidence of prediction.

**Returns**

- `outlier_labels` (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.

- `confidence` (numpy array of shape \((n_{\text{samples}},)\)) – Only if `return_confidence` is set to True.

**predict_confidence**\((X)\)  
Predict the model’s confidence in making the same prediction under slightly different training sets. See [BPVD20].

**Parameters**

- \(X\) (numpy array of shape \((n_{\text{samples}}, n_{\text{features}})\)) – The input samples.

**Returns**

- `confidence` – For each observation, tells how consistently the model would make the same prediction if the training set was perturbed. Return a probability, ranging in [0,1].

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

**predict_proba**\((X, \text{method=}'\text{linear}', \text{return\_confidence}=\text{False})\)  
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 \((n_{\text{samples}}, n_{\text{features}})\)) – The input samples.

- `method` (str, optional (default=’linear’)) – probability conversion method. It must be one of ‘linear’ or ‘unify’.

- `return_confidence` (boolean, optional(default=False)) – If True, also return the confidence of prediction.

**Returns**

- `outlier_probability` – 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]. Note it depends on the number of classes, which is by default 2 classes ([proba of normal, proba of outliers]).

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

**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 `\langle component\rangle__\langle parameter\rangle` so that it's possible to update each component of a nested object.


**Returns**

- `self`

**Return type**  
object
pyod.models.auto_encoder_torch module

Using AutoEncoder with Outlier Detection (PyTorch)

class pyod.models.auto_encoder_torch.AutoEncoder(hidden_neurons=None, hidden_activation='relu',
    batch_norm=True, learning_rate=0.001,
    epochs=100, batch_size=32, dropout_rate=0.2,
    weight_decay=1e-05, preprocessing=True,
    loss_fn=None, contamination=0.1, device=None)

Bases: pyod.models.base.BaseDetector

Auto Encoder (AE) is a type of neural networks for learning useful data representations in an unsupervised manner. 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.

Notes

This is the PyTorch version of AutoEncoder. See auto_encoder.py for the TensorFlow version.

The documentation is not finished!

Parameters

- **hidden_neurons** (list, optional (default=[64, 32])) – The number of neurons per hidden layers. So the network has the structure as [n_features, 64, 32, 64, n_features]
- **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/
- **batch_norm** (boolean, optional (default=True)) – Whether to apply Batch Normalization. See https://pytorch.org/docs/stable/generated/torch.nn.BatchNorm1d.html
- **loss** (str or obj, optional (default=torch.nn.MSELoss)) – String (name of objective function) or objective function. NOT SUPPORT FOR CHANGE YET.
- **optimizer** (str, optional (default='adam')) – String (name of optimizer) or optimizer instance. NOT SUPPORT FOR CHANGE YET.
- **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.
For verbose >= 1, model summary may be printed.

- **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_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 ignored in unsupervised methods.
Parameters

- \(X\) (numpy array of shape \((n_{\text{samples}}, n_{\text{features}})\)) – The input samples.
- \(y\) (Ignored) – Not used, present for API consistency by convention.

Returns self – Fitted estimator.

Return type object

fit_predict\((X, y=None)\)
DEPRECATED

Fit detector first and then predict whether a particular sample is an outlier or not. \(y\) is ignored in unsupervised models.

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

\(y\) [Ignored] Not used, present for API consistency by convention.

outlier_labels [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.

Deprecated since version 0.6.9: fit_predict will be removed in pyod 0.8.0.; it will be replaced by calling fit function first and then accessing labels_ attribute for consistency.

fit_predict_score\((X, y, \text{scoring}='roc_auc_score')\)
DEPRECATED

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

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

\(y\) [Ignored] Not used, present for API consistency by convention.

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 0.8.0.; 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\((\text{deep=}\text{True})\)

Get parameters for this estimator.


Parameters deep (bool, optional (default=True)) – 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\((X, \text{return\_confidence=False})\)

Predict if a particular sample is an outlier or not.

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

- **return_confidence** *(boolean, optional (default=False)) – If True, also return the confidence of prediction.*

**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.*

- **confidence** *(numpy array of shape (n_samples,)) – Only if return_confidence is set to True.*

**predict_confidence(X)**

Predict the model’s confidence in making the same prediction under slightly different training sets. See [BPVD20].

**Parameters**

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

**Returns**

- **confidence** – For each observation, tells how consistently the model would make the same prediction if the training set was perturbed. Return a probability, ranging in [0,1].

**Return type**

- numpy array of shape (n_samples,)

**predict_proba(X, method='linear', return_confidence=False)**

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’.*

- **return_confidence** *(boolean, optional (default=False)) – If True, also return the confidence of prediction.*

**Returns**

- **outlier_probability** – 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]. Note it depends on the number of classes, which is by default 2 classes ([proba of normal, proba of outliers]).

**Return type**

- numpy array of shape (n_samples, n_classes)

**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**
class pyod.models.auto_encoder_torch.PyODDataset:
    X, y=None, mean=None, std=None)

PyOD Dataset class for PyTorch Dataloader

functions: Dict[str, Callable] = {'concat': functools.partial(<function
    Dataset.register_datapipe_as_function.<locals>.class_function>,
    'torch.utils.data.datapipes.map.combining.ConcaterMapDataPipe', False),
    'map':
    functools.partial(<function
    Dataset.register_datapipe_as_function.<locals>.class_function>,
    'torch.utils.data.datapipes.map.callable.MapperMapDataPipe', False)}

classmethod register_datapipe_as_function:
    function_name, cls_to_register, enable_df_api_tracing=False)

classmethod register_function:
    function_name, function

class pyod.models.auto_encoder_torch.inner_autoencoder:
    n_features=128, hidden_neurons=[128, 64],
    dropout_rate=0.2, batch_norm=True,
    hidden_activation='relu')

T_destination
    alias of TypeVar('T_destination', bound=Mapping[str, torch.Tensor])

add_module:
    name: str, module: Optional[torch.nn.modules.module.Module] -> None

    Adds a child module to the current module.

    The module can be accessed as an attribute using the given name.

    Parameters

        • name (string) – name of the child module. The child module can be accessed from this
          module using the given name

        • module (Module) – child module to be added to the module.

apply:
    fn: Callable[[torch.nn.modules.module.Module], None] -> torch.nn.modules.module.T

    Applies fn recursively to every submodule (as returned by .children() as well as self. Typical use
    includes initializing the parameters of a model (see also nn-init-doc).

    Parameters fn (Module -> None) – function to be applied to each submodule

    Returns self

    Return type Module

Example:

    >>> @torch.no_grad()
    >>> def init_weights(m):
    >>>     print(m)
    >>>     if type(m) == nn.Linear:
    >>>         m.weight.fill_().1.0
    >>>     print(m.weight)
    >>> net = nn.Sequential(nn.Linear(2, 2), nn.Linear(2, 2))
    >>> net.apply(init_weights)
    Linear(in_features=2, out_features=2, bias=True)
    Parameter containing:
    tensor([[1., 1.],
            [1., 1]])
Linear(in_features=2, out_features=2, bias=True)
Parameter containing:
tensor([[ 1., 1.],
[ 1., 1.]])
Sequential(
    (0): Linear(in_features=2, out_features=2, bias=True)
    (1): Linear(in_features=2, out_features=2, bias=True)
)
Sequential(
    (0): Linear(in_features=2, out_features=2, bias=True)
    (1): Linear(in_features=2, out_features=2, bias=True)
)

**bfloat16** → torch.nn.modules.module.T
Casts all floating point parameters and buffers to **bfloat16** datatype.

**Note:** This method modifies the module in-place.

**Returns**  self

**Return type**  Module

buffers**(recurse: bool = True)** → Iterator[torch.Tensor]
Returns an iterator over module buffers.

**Parameters**  recurse *(bool)* – if True, then yields buffers of this module and all submodules. Otherwise, yields only buffers that are direct members of this module.

**Yields**  torch.Tensor – module buffer

Example:

```python
>>> for buf in model.buffers():
    print(type(buf), buf.size())
<class 'torch.Tensor'> (20L,)
<class 'torch.Tensor'> (20L, 1L, 5L, 5L)
```

children() → Iterator[torch.nn.modules.module.Module]
Returns an iterator over immediate children modules.

**Yields**  Module – a child module

cpu() → torch.nn.modules.module.T
Moves all model parameters and buffers to the CPU.

**Note:** This method modifies the module in-place.

**Returns**  self

**Return type**  Module

cuda**(device: Optional[Union[int, torch.device]] = None)** → torch.nn.modules.module.T
Moves all model parameters and buffers to the GPU.
This also makes associated parameters and buffers different objects. So it should be called before constructing optimizer if the module will live on GPU while being optimized.

Note: This method modifies the module in-place.

**Parameters**

- **device** (*int, optional*) – if specified, all parameters will be copied to that device

**Returns**

- **self**

**Return type**

- **Module**

### double() → torch.nn.modules.module.T

Casts all floating point parameters and buffers to `double` datatype.

Note: This method modifies the module in-place.

**Returns**

- **self**

**Return type**

- **Module**

### dump_patches: bool = False

This allows better BC support for `load_state_dict()`. In `state_dict()`, the version number will be saved as in the attribute `_metadata` of the returned state dict, and thus pickled. `_metadata` is a dictionary with keys that follow the naming convention of state dict. See `_load_from_state_dict` on how to use this information in loading.

If new parameters/buffers are added/removed from a module, this number shall be bumped, and the module’s `_load_from_state_dict` method can compare the version number and do appropriate changes if the state dict is from before the change.

### eval() → torch.nn.modules.module.T

Sets the module in evaluation mode.

This has any effect only on certain modules. See documentations of particular modules for details of their behaviors in training/evaluation mode, if they are affected, e.g. Dropout, BatchNorm, etc.

This is equivalent with `self.train(False)`.

See locally-disable-grad-doc for a comparison between `eval()` and several similar mechanisms that may be confused with it.

**Returns**

- **self**

**Return type**

- **Module**

### extra_repr() → str

Set the extra representation of the module

To print customized extra information, you should re-implement this method in your own modules. Both single-line and multi-line strings are acceptable.

### float() → torch.nn.modules.module.T

Casts all floating point parameters and buffers to `float` datatype.

Note: This method modifies the module in-place.
Returns  

Return type  Module

forward(x)

Defines the computation performed at every call.
Should be overridden by all subclasses.

Note: Although the recipe for forward pass needs to be defined within this function, one should call the Module instance afterwards instead of this since the former takes care of running the registered hooks while the latter silently ignores them.

get_buffer(target: str) → torch.Tensor

Returns the buffer given by target if it exists, otherwise throws an error.
See the docstring for get_submodule for a more detailed explanation of this method’s functionality as well as how to correctly specify target.

Parameters target – The fully-qualified string name of the buffer to look for. (See get_submodule for how to specify a fully-qualified string.)

Returns  The buffer referenced by target

Return type  torch.Tensor

Raises  AttributeError – If the target string references an invalid path or resolves to something that is not a buffer

get_extra_state() → Any

Returns any extra state to include in the module’s state_dict. Implement this and a corresponding set_extra_state() for your module if you need to store extra state. This function is called when building the module’s state_dict().

Note that extra state should be pickleable to ensure working serialization of the state_dict. We only provide provide backwards compatibility guarantees for serializing Tensors; other objects may break backwards compatibility if their serialized pickled form changes.

Returns  Any extra state to store in the module’s state_dict

Return type  object

get_parameter(target: str) → torch.nn.parameter.Parameter

Returns the parameter given by target if it exists, otherwise throws an error.
See the docstring for get_submodule for a more detailed explanation of this method’s functionality as well as how to correctly specify target.

Parameters target – The fully-qualified string name of the Parameter to look for. (See get_submodule for how to specify a fully-qualified string.)

Returns  The Parameter referenced by target

Return type  torch.nn.Parameter

Raises  AttributeError – If the target string references an invalid path or resolves to something that is not an nn.Parameter

get_submodule(target: str) → torch.nn.modules.module.Module

Returns the submodule given by target if it exists, otherwise throws an error.
For example, let’s say you have an nn.Module A that looks like this:
(The diagram shows an `nn.Module` A has a nested submodule `net_b`, which itself has two submodules `net_c` and `linear`. `net_c` then has a submodule `conv`.

To check whether or not we have the `linear` submodule, we would call `get_submodule("net_b.linear")`. To check whether we have the `conv` submodule, we would call `get_submodule("net_b.net_c.conv")`.

The runtime of `get_submodule` is bounded by the degree of module nesting in `target`. A query against `named_modules` achieves the same result, but it is O(N) in the number of transitive modules. So, for a simple check to see if some submodule exists, `get_submodule` should always be used.

**Parameters**
- `target` – The fully-qualified string name of the submodule to look for. (See above example for how to specify a fully-qualified string.)

**Returns**
- The submodule referenced by `target`

**Return type**
- `torch.nn.Module`

**Raises**
- `AttributeError` – If the target string references an invalid path or resolves to something that is not an `nn.Module`

`half()` → `torch.nn.modules.module.T`
Casts all floating point parameters and buffers to half datatype.

**Note:**
This method modifies the module in-place.

**Returns**
- `self`

**Return type**
- `Module`

`load_state_dict(state_dict: OrderedDict[str, Tensor], strict: bool = True)`
Copies parameters and buffers from `state_dict` into this module and its descendants. If `strict` is True, then the keys of `state_dict` must exactly match the keys returned by this module’s `state_dict()` function.

**Parameters**
- `state_dict` – a dict containing parameters and persistent buffers.
- `strict` (bool, optional) – whether to strictly enforce that the keys in `state_dict` match the keys returned by this module’s `state_dict()` function. Default: True

**Returns**
- `missing_keys` is a list of str containing the missing keys
- `unexpected_keys` is a list of str containing the unexpected keys

**Return type**
- `NamedTuple` with `missing_keys` and `unexpected_keys` fields

**Note:**
If a parameter or buffer is registered as `None` and its corresponding key exists in `state_dict`, `load_state_dict()` will raise a `RuntimeError`.

`modules()` → `Iterator[torch.nn.modules.module.Module]`
Returns an iterator over all modules in the network.

**Yields**
- `Module` – a module in the network
**Note:** Duplicate modules are returned only once. In the following example, \( l \) will be returned only once.

Example:

```python
>>> l = nn.Linear(2, 2)
>>> net = nn.Sequential(l, l)
>>> for idx, m in enumerate(net.modules()):
    print(idx, '->', m)

0 -> Sequential(  
(0): Linear(in_features=2, out_features=2, bias=True)  
(1): Linear(in_features=2, out_features=2, bias=True)  
)
1 -> Linear(in_features=2, out_features=2, bias=True)
```

**named_buffers**

\( \text{named_buffers}(\text{prefix: str} = ",", \text{recurse: bool} = \text{True}) \rightarrow \text{Iterator}[(\text{str, torch.Tensor})] \)

Returns an iterator over module buffers, yielding both the name of the buffer as well as the buffer itself.

**Parameters**

- **prefix** (str) – prefix to prepend to all buffer names.
- **recurse** (bool) – if True, then yields buffers of this module and all submodules. Otherwise, yields only buffers that are direct members of this module.

**Yields**

\((\text{string, torch.Tensor})\) – Tuple containing the name and buffer

Example:

```python
>>> for name, buf in self.named_buffers():
    if name in ['running_var']:
        print(buf.size())
```

**named_children**

\( \rightarrow \text{Iterator}[(\text{str, torch.nn.modules.module.Module})] \)

Returns an iterator over immediate children modules, yielding both the name of the module as well as the module itself.

**Yields**

\((\text{string, Module})\) – Tuple containing a name and child module

Example:

```python
>>> for name, module in model.named_children():
    if name in ['conv4', 'conv5']:
        print(module)
```

**named_modules**

\( \text{named_modules}(\text{memo: Optional[Set[torch.nn.modules.module.Module]]} = \text{None}, \text{prefix: str} = ",", \text{remove_duplicate: bool} = \text{True}) \)

Returns an iterator over all modules in the network, yielding both the name of the module as well as the module itself.

**Parameters**

- **memo** – a memo to store the set of modules already added to the result
- **prefix** – a prefix that will be added to the name of the module
- **remove_duplicate** – whether to remove the duplicated module instances in the result
- **not (or)** –
Yields (string, Module) – Tuple of name and module

Note: Duplicate modules are returned only once. In the following example, `l` will be returned only once.

Example:

```python
>>> l = nn.Linear(2, 2)
>>> net = nn.Sequential(l, l)
>>> for idx, m in enumerate(net.named_modules()):
    print(idx, '->', m)
0 -> ('', Sequential((0): Linear(in_features=2, out_features=2, bias=True)
  (1): Linear(in_features=2, out_features=2, bias=True))
1 -> ('0', Linear(in_features=2, out_features=2, bias=True))
```

named_parameters(prefix: str = '', recurse: bool = True) → Iterator[Tuple[str, torch.nn.parameter.Parameter]]

Returns an iterator over module parameters, yielding both the name of the parameter as well as the parameter itself.

Parameters

- **prefix** (str) – prefix to prepend to all parameter names.
- **recurse** (bool) – if True, then yields parameters of this module and all submodules. Otherwise, yields only parameters that are direct members of this module.

Yields (string, Parameter) – Tuple containing the name and parameter

Example:

```python
>>> for name, param in self.named_parameters():
    if name in ['bias']:
        print(param.size())
```

parameters(recurse: bool = True) → Iterator[torch.nn.parameter.Parameter]

Returns an iterator over module parameters. This is typically passed to an optimizer.

Parameters **recurse** (bool) – if True, then yields parameters of this module and all submodules. Otherwise, yields only parameters that are direct members of this module.

Yields Parameter – module parameter

Example:

```python
>>> for param in model.parameters():
    print(type(param), param.size())
<class 'torch.Tensor'> (20L,)
<class 'torch.Tensor'> (20L, 1L, 5L, 5L)
```


Registers a backward hook on the module.
This function is deprecated in favor of `register_full_backward_hook()` and the behavior of this function will change in future versions.

**Returns** a handle that can be used to remove the added hook by calling `handle.remove()`

**Return type** `torch.utils.hooks.RemovableHandle`

### register_buffer

```
register_buffer(name: str, tensor: Optional[torch.Tensor], persistent: bool = True) → None
```

Adds a buffer to the module.

This is typically used to register a buffer that should not to be considered a model parameter. For example, BatchNorm’s `running_mean` is not a parameter, but is part of the module’s state. Buffers, by default, are persistent and will be saved alongside parameters. This behavior can be changed by setting `persistent` to `False`. The only difference between a persistent buffer and a non-persistent buffer is that the latter will not be a part of this module’s `state_dict`.

Buffers can be accessed as attributes using given names.

**Parameters**

- **name** *(string)* – name of the buffer. The buffer can be accessed from this module using the given name
- **tensor** *(Tensor or None)* – buffer to be registered. If `None`, then operations that run on buffers, such as `cuda`, are ignored. If `None`, the buffer is not included in the module’s `state_dict`.
- **persistent** *(bool)* – whether the buffer is part of this module’s `state_dict`.

**Example:**

```python
>>> self.register_buffer('running_mean', torch.zeros(num_features))
```

### register_forward_hook

```
register_forward_hook(hook: Callable[ [...], None]) → torch.utils.hooks.RemovableHandle
```

Registers a forward hook on the module.

The hook will be called every time after `forward()` has computed an output. It should have the following signature:

```python
hook(module, input, output) -> None or modified output
```

The input contains only the positional arguments given to the module. Keyword arguments won’t be passed to the hooks and only to the `forward`. The hook can modify the output. It can modify the input inplace but it will not have effect on forward since this is called after `forward()` is called.

**Returns** a handle that can be used to remove the added hook by calling `handle.remove()`

**Return type** `torch.utils.hooks.RemovableHandle`

### register_forward_pre_hook

```
register_forward_pre_hook(hook: Callable[ [...], None]) → torch.utils.hooks.RemovableHandle
```

Registers a forward pre-hook on the module.

The hook will be called every time before `forward()` is invoked. It should have the following signature:

```python
hook(module, input) -> None or modified input
```

The input contains only the positional arguments given to the module. Keyword arguments won’t be passed to the hooks and only to the `forward`. The hook can modify the input. User can either return a tuple or a single modified value in the hook. We will wrap the value into a tuple if a single value is returned(unless that value is already a tuple).

**Returns** a handle that can be used to remove the added hook by calling `handle.remove()`
Return type  torch.utils.hooks.RemovableHandle

register_full_backward_hook(hook: Callable[[torch.nn.modules.module.Module,
    Union[Tuple[torch.Tensor, ...], torch.Tensor],
    Union[Tuple[torch.Tensor, ...], torch.Tensor]],
    Union[None, torch.Tensor]])  →  
torch.utils.hooks.RemovableHandle

 Registers a backward hook on the module.

 The hook will be called every time the gradients with respect to module inputs are computed. The hook
 should have the following signature:

```python
hook(module, grad_input, grad_output) -> tuple(Tensor) or None
```

The `grad_input` and `grad_output` are tuples that contain the gradients with respect to the inputs and
outputs respectively. The hook should not modify its arguments, but it can optionally return a new gra-
dient with respect to the input that will be used in place of `grad_input` in subsequent computations.
`grad_input` will only correspond to the inputs given as positional arguments and all kwarg arguments are
ignored. Entries in `grad_input` and `grad_output` will be `None` for all non-Tensor arguments.

For technical reasons, when this hook is applied to a Module, its forward function will receive a view of
each Tensor passed to the Module. Similarly the caller will receive a view of each Tensor returned by the
Module’s forward function.

**Warning:** Modifying inputs or outputs inplace is not allowed when using backward hooks and will
raise an error.

Returns  a handle that can be used to remove the added hook by calling `handle.remove()`

Return type  torch.utils.hooks.RemovableHandle

register_parameter(name: str, param: Optional[torch.nn.parameter.Parameter])  →  None

 Adds a parameter to the module.

 The parameter can be accessed as an attribute using given name.

Parameters

- **name** (string) – name of the parameter. The parameter can be accessed from this module
  using the given name
- **param** (Parameter or None) – parameter to be added to the module. If None, then
  operations that run on parameters, such as `cuda`, are ignored. If None, the parameter is
  not included in the module’s `state_dict`.

requires_grad_(requires_grad: bool = True)  →  torch.nn.modules.module.T

Change if autograd should record operations on parameters in this module.

This method sets the parameters’ `requires_grad` attributes in-place.

This method is helpful for freezing part of the module for finetuning or training parts of a model individually
(e.g., GAN training).

See locally-disable-grad-doc for a comparison between `.requires_grad()` and several similar mechanisms
that may be confused with it.

Parameters **requires_grad** (bool) – whether autograd should record operations on param-
eters in this module. Default: True.

Returns  self
Return type: Module

`set_extra_state(state: Any)`
This function is called from `load_state_dict()` to handle any extra state found within the `state_dict`. Implement this function and a corresponding `get_extra_state()` for your module if you need to store extra state within its `state_dict`.

Parameters:
- `state (dict)` – Extra state from the `state_dict`

`share_memory()` → torch.nn.modules.module.T
See `torch.Tensor.share_memory()`

`state_dict(destination=None, prefix='', keep_vars=False)`
Returns a dictionary containing a whole state of the module.
Both parameters and persistent buffers (e.g. running averages) are included. Keys are corresponding parameter and buffer names. Parameters and buffers set to `None` are not included.

Returns: a dictionary containing a whole state of the module

Return type: dict

Example:
```python
>>> module.state_dict().keys()
['bias', 'weight']
```

`to(*args, **kwargs)`
Moves and/or casts the parameters and buffers.
This can be called as
```python
to(device=None, dtype=None, non_blocking=False)
to(dtype, non_blocking=False)
to(tensor, non_blocking=False)
to(memory_format=torch.channels_last)
```
Its signature is similar to `torch.Tensor.to()`, but only accepts floating point or complex dtypes. In addition, this method will only cast the floating point or complex parameters and buffers to `dtype` (if given). The integral parameters and buffers will be moved `device`, if that is given, but with dtypes unchanged. When `non_blocking` is set, it tries to convert/move asynchronously with respect to the host if possible, e.g., moving CPU Tensors with pinned memory to CUDA devices.

See below for examples.

**Note:** This method modifies the module in-place.

**Parameters**
- `device (torch.device)` – the desired device of the parameters and buffers in this module
- `dtype (torch.dtype)` – the desired floating point or complex dtype of the parameters and buffers in this module
- `tensor (torch.Tensor)` – Tensor whose dtype and device are the desired dtype and device for all parameters and buffers in this module
- `memory_format (torch.memory_format)` – the desired memory format for 4D parameters and buffers in this module (keyword only argument)
**Returns**

self

**Return type** Module

Examples:

```python
>>> linear = nn.Linear(2, 2)
>>> linear.weight
Parameter containing:
tensor([[ 0.1913, -0.3420],
        [-0.5113, -0.2325]])
>>> linear.to(torch.double)
Linear(in_features=2, out_features=2, bias=True)
>>> linear.weight
Parameter containing:
tensor([[ 0.1913, -0.3420],
        [-0.5113, -0.2325]], dtype=torch.float64)
>>> gpul = torch.device("cuda:1")
>>> linear.to(gpul, dtype=torch.half, non_blocking=True)
Linear(in_features=2, out_features=2, bias=True)
>>> linear.weight
Parameter containing:
tensor([[ 0.1914, -0.3420],
        [-0.5112, -0.2324]], dtype=torch.float16, device='cuda:1')
>>> cpu = torch.device("cpu")
>>> linear.to(cpu)
Linear(in_features=2, out_features=2, bias=True)
>>> linear.weight
Parameter containing:
tensor([[ 0.1914, -0.3420],
        [-0.5112, -0.2324]], dtype=torch.float16)

>>> linear = nn.Linear(2, 2, bias=None).to(torch.cdouble)
>>> linear.weight
Parameter containing:
tensor([[ 0.3741+0.j, 0.2382+0.j],
        [ 0.5593+0.j, -0.4443+0.j]], dtype=torch.complex128)
>>> linear(torch.ones(3, 2, dtype=torch.cdouble))
tensor([[0.6122+0.j, 0.1150+0.j],
        [0.6122+0.j, 0.1150+0.j],
        [0.6122+0.j, 0.1150+0.j]], dtype=torch.complex128)
```

**to_empty**(*, device: Union[str, torch.device]) → torch.nn.modules.module.T

Moves the parameters and buffers to the specified device without copying storage.

**Parameters**

device (torch.device) – The desired device of the parameters and buffers in this module.

**Returns**

self

**Return type** Module

**train**(mode: bool = True) → torch.nn.modules.module.T

Sets the module in training mode.

This has any effect only on certain modules. See documentations of particular modules for details of their behaviors in training/evaluation mode, if they are affected, e.g. Dropout, BatchNorm, etc.
**Parameters**

*mode* (``bool``) – whether to set training mode (``True``) or evaluation mode (``False``).

Default: ``True``.

**Returns**

``self``

**Return type**

``Module``

---

**training**:

``bool``

**type** (``dst_type: Union[torch.dtype, str]``) → ``torch.nn.modules.module.T``

Casts all parameters and buffers to ``dst_type``.

**Note:** This method modifies the module in-place.

**Parameters**

``dst_type`` (``type or string``) – the desired type

**Returns**

``self``

**Return type**

``Module``

---

**xpu** (``device: Optional[Union[int, torch.device]] = None``) → ``torch.nn.modules.module.T``

Moves all model parameters and buffers to the XPU.

This also makes associated parameters and buffers different objects. So it should be called before constructing optimizer if the module will live on XPU while being optimized.

**Note:** This method modifies the module in-place.

**Parameters**

``device`` (``int, optional``) – if specified, all parameters will be copied to that device

**Returns**

``self``

**Return type**

``Module``

---

**zero_grad** (``set_to_none: bool = False``) → ``None``

Sets gradients of all model parameters to zero. See similar function under ``torch.optim.Optimizer`` for more context.

**Parameters**

``set_to_none`` (``bool``) – instead of setting to zero, set the grads to None. See ``torch.optim.Optimizer.zero_grad()`` for details.

---

### pyod.models.cblof module

Clustering Based Local Outlier Factor (CBLOF)

**class** `pyod.models.cblof.CBLOF``(n_clusters=8, contamination=0.1, clustering_estimator=None, alpha=0.9, beta=5, use_weights=False, check_estimator=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 |C1|, |C2|, ..., |Cn|, beta = |Ck|/|Ck-1|*

- **use_weights** *(bool, optional (default=False)) – If set to True, the size of clusters are used as weights in outlier score calculation.*

- **check_estimator** *(bool, optional (default=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.*

**clustering_estimator_**
Base estimator for clustering.

**Type** Estimator, sklearn instance

**cluster_labels_**
Cluster assignment for the training samples.

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

**n_clusters_**
Actual number of clusters (possibly different from n_clusters).
Type int

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

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,)

cluster_centers_
The center of each cluster.
Type numpy array of shape (n_clusters_, n_features)

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

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

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 ignored in unsupervised methods.

Parameters

- X (numpy array of shape (n_samples, n_features)) – The input samples.
- y (Ignored) – Not used, present for API consistency by convention.

Returns

- self – Fitted estimator.

Return type

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

Fit detector first and then predict whether a particular sample is an outlier or not. \( y \) is ignored in unsupervised models.

- \( X \) [npy array of shape (n_samples, n_features)] The input samples.
- \( y \) [Ignored] Not used, present for API consistency by convention.

outlier_labels [npy 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 0.8.0.; it will be replaced by calling fit function first and then accessing labels_ attribute for consistency.

fit_predict_score \( (X, y, \text{scoring='roc_auc_score'}) \)
DEPRECATED

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

- \( X \) [npy array of shape (n_samples, n_features)] The input samples.
- \( y \) [Ignored] Not used, present for API consistency by convention.
- \( \text{scoring} \) [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 0.8.0.; 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 \( (\text{deep=True}) \)

Get parameters for this estimator.


Parameters deep (bool, optional (default=True)) – 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 \( (X, \text{return_confidence=False}) \)

Predict if a particular sample is an outlier or not.

Parameters

- \( X \) (npy array of shape (n_samples, n_features)) – The input samples.
- \( \text{return_confidence} \) (bool, optional (default=False)) – If True, also return the confidence of prediction.

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.

• **confidence** (*numpy array of shape (n_samples,)*) – Only if return_confidence is set to True.

**predict_confidence**(*X*)

Predict the model’s confidence in making the same prediction under slightly different training sets. See [BPVD20].

**Parameters**

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

**Returns**

* confidence – For each observation, tells how consistently the model would make the same prediction if the training set was perturbed. Return a probability, ranging in [0,1].

**Return type**

*numpy array of shape (n_samples,)*

**predict_proba**(*X, method=’linear’, return_confidence=False*)

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’.

* return_confidence (*boolean, optional (default=False)*) – If True, also return the confidence of prediction.

**Returns**

* outlier_probability – 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]. Note it depends on the number of classes, which is by default 2 classes ([proba of normal, proba of outliers]).

**Return type**

*numpy array of shape (n_samples, n_classes)*

**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, method='fast')`

*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.

Two version of COF are supported:

- Fast COF: computes the entire pairwise distance matrix at the cost of a O(n^2) memory requirement.
- Memory efficient COF: calculates pairwise distances incrementally. Use this implementation when it is not feasible to fit the n-by-n distance in memory. This leads to a linear overhead because many distances will have to be recalculated.

**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.*

- **method** *(string, optional (default='fast')) – Valid values for method are:
  - 'fast' Fast COF, computes the full pairwise distance matrix up front.
  - 'memory' Memory-efficient COF, computes pairwise distances only when needed at the cost of computational speed.*

**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 ignored in unsupervised methods.

**Parameters**

- **X** (*numpy array of shape (n_samples, n_features]*) – The input samples.
- **y** (*Ignored*) – Not used, present for API consistency by convention.

**Returns** *self* – Fitted estimator.

**Return type** *object*

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

DEPRECATED

Fit detector first and then predict whether a particular sample is an outlier or not. y is ignored in unsupervised models.

**Parameters**

- **X** (*numpy array of shape (n_samples, n_features]*) – The input samples.
- **y** (*Ignored*) – Not used, present for API consistency by convention.

**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.

Deprecated since version 0.6.9: **fit_predict** will be removed in pyod 0.8.0.; 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** (*Ignored*) – Not used, present for API consistency by convention.
- **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 0.8.0.; 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** (bool, optional (default=True)) – 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*, *return_confidence=False*)

Predict if a particular sample is an outlier or not.

**Parameters**

- **X** (*numpy array of shape (n_samples, n_features]*) – The input samples.
- **return_confidence** (*boolean, optional (default=False]*) – If True, also return the confidence of prediction.

**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.
- **confidence** (*numpy array of shape (n_samples,]*) – Only if return_confidence is set to True.

**predict_confidence**(*X*)

Predict the model’s confidence in making the same prediction under slightly different training sets. See [BPVD20].

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

**Returns** **confidence** – For each observation, tells how consistently the model would make the same prediction if the training set was perturbed. Return a probability, ranging in [0,1].

**Return type** numpy array of shape (n_samples,)

**predict_proba**(*X*, *method=’linear’, *return_confidence=False*)

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’.
- **return_confidence** (*boolean, optional (default=False]*) – If True, also return the confidence of prediction.

**Returns** **outlier_probability** – 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]. Note it depends on the number of classes, which is by default 2 classes ([proba of normal, proba of outliers]).

**Return type** numpy array of shape (n_samples, n_classes)

**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.
pyod Documentation, Release 0.9.7


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_weights=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_weights (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.majority_vote(scores, weights=None)

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

Parameters scores (numpy array of shape (n_samples, n_estimators)) – Score matrix from multiple estimators on the same samples.
• weights [numpy array of shape (1, n_estimators)] If specified, using weighted majority weight.
Returns **combined_scores** – The combined scores.

Return type  numpy array of shape (n_samples, )

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

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

Returns **combined_scores** – The combined scores.

Return type  numpy array of shape (n_samples, )

**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,)
Copula Based Outlier Detector (COPOD)

```python
class pyod.models.copod.COPOD(contamination=0.1, n_jobs=1)
    Bases: pyod.models.base.BaseDetector
```

COPOD class for Copula Based Outlier Detector. COPOD is a parameter-free, highly interpretable outlier detection algorithm based on empirical copula models. See [BLZB+20] 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_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.*

**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. 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,)

**explain_outlier(ind, columns=None, cutoffs=None, feature_names=None, file_name=None, file_type=None)**

Plot dimensional outlier graph for a given data point within the dataset.

Parameters:

- `ind` *(int)* – The index of the data point one wishes to obtain a dimensional outlier graph for.
- `columns` *(list)* – Specify a list of features/dimensions for plotting. If not specified, use all features.
• **cutoffs** *(list of floats in (0., 1), optional (default=[0.95, 0.99]))* – The significance cutoff bands of the dimensional outlier graph.

• **feature_names** *(list of strings)* – The display names of all columns of the dataset, to show on the x-axis of the plot.

• **file_name** *(string)* – The name to save the figure

• **file_type** *(string)* – The file type to save the figure

**Returns** *Plot* – The dimensional outlier graph for data point with index ind.

**Return type** *matplotlib plot*

`fit(X, y=None)`

Fit detector. y is ignored in unsupervised methods.

- **param X:** The input samples.
- **type X:** numpy array of shape (n_samples, n_features)
- **param y:** Not used, present for API consistency by convention.
- **type y:** Ignored

**Returns** *self* – Fitted estimator.

**Return type** *object*

`fit_predict(X, y=None)`

**DEPRECATED**

Fit detector first and then predict whether a particular sample is an outlier or not. y is ignored in unsupervised models.

- **X** [numpy array of shape (n_samples, n_features)] The input samples.
- **y** [Ignored] Not used, present for API consistency by convention.

**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 0.8.0.; 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** [Ignored] Not used, present for API consistency by convention.
- **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 0.8.0.; 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** *(bool, optional (default=True)) –* 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, *return_confidence=False*)

Predict if a particular sample is an outlier or not.

Parameters

- **X** *(numpy array of shape (n_samples, n_features)) –* The input samples.
- **return_confidence** *(boolean, optional (default=False)) –* If True, also return the confidence of prediction.

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.
- **confidence** *(numpy array of shape (n_samples,)) –* Only if return_confidence is set to True.

**predict_confidence**(X)

Predict the model’s confidence in making the same prediction under slightly different training sets. See [BPVD20].

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

Returns **confidence** – For each observation, tells how consistently the model would make the same prediction if the training set was perturbed. Return a probability, ranging in [0,1].

Return type  numpy array of shape (n_samples,)

**predict_proba**(X, *method='linear', return_confidence=False*)

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’.
- **return_confidence** *(boolean, optional (default=False)) –* If True, also return the confidence of prediction.

Returns **outlier_probability** – 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]. Note it depends on the number of classes, which is by default 2 classes ([proba of normal, proba of outliers]).

Return type  numpy array of shape (n_samples, n_classes)
**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.copod.ecdf(X)

Calculated the empirical CDF of a given dataset.

**Parameters**

- **X**: The training dataset.

**Returns** ecdf(X) – Empirical CDF of X

**Return type** float

**pyod.models.deep_svdd module**

Deep One-Class Classification for outlier detection

class pyod.models.deep_svdd.DeepSVDD(c=None, use_ae=False, hidden_neurons=None, hidden_activation='relu', output_activation='sigmoid', optimizer='adam', epochs=100, batch_size=32, dropout_rate=0.2, l2_regularizer=0.1, validation_size=0.1, preprocessing=True, verbose=1, random_state=None, contamination=0.1)

Bases: pyod.models.base.BaseDetector

Deep One-Class Classifier with AutoEncoder (AE) is a type of neural networks for learning useful data representations in an unsupervised way. DeepSVDD trains a neural network while minimizing the volume of a hypersphere that encloses the network representations of the data, forcing the network to extract the common factors of variation. Similar to PCA, DeepSVDD could be used to detect outlying objects in the data by calculating the distance from center. See [BRVG+18] for details.

**Parameters**

- **c** *(float, optional (default='forward_nn_pass'))* – Deep SVDD center, the default will be calculated based on network initialization first forward pass. To get repeated results set random_state if c is set to None.
- **use_ae** *(bool, optional (default=False))* – The AutoEncoder type of DeepSVDD it reverse neurons from hidden_neurons if set to True.
- **hidden_neurons** *(list, optional (default=[64, 32]))* – The number of neurons per hidden layers. If use ae is True, neurons will be reversed eg. [64, 32] -> [64, 32, 32, 64, n_features]
- **hidden_activation** *(str, optional (default='relu'))* – Activation function to use for hidden layers. All hidden layers are forced to use the same type of activation. See https://keras.io/activations/
- **output_activation** *(str, optional (default='sigmoid'))* – Activation function to use for output layer. See https://keras.io/activations/
- **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.

  For verbose >= 1, model summary may be printed.

- **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.

  - **model_** The underlying DeppSVDD 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_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 ignored in unsupervised methods.

**Parameters**

- **X** (*numpy array of shape (n_samples, n_features]*) – The input samples.
- **y** (*Ignored*) – Not used, present for API consistency by convention.

**Returns**

- **self** – Fitted estimator.

**Return type**

- *object*

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

DEPRECATED

Fit detector first and then predict whether a particular sample is an outlier or not. *y* is ignored in unsupervised models.

**Parameters**

- **X** [*numpy array of shape (n_samples, n_features)*] The input samples.
- **y** [**Ignored**] Not used, present for API consistency by convention.

**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.

Deprecated since version 0.6.9: *fit_predict* will be removed in pyod 0.8.0.; 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** [**Ignored**] Not used, present for API consistency by convention.
- **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 0.8.0.; 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` (`bool`, `optional (default=True)`) – 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, return_confidence=False)`

Predict if a particular sample is an outlier or not.

Parameters

- `X` (`numpy array of shape (n_samples, n_features)`) – The input samples.
- `return_confidence` (`boolean, optional (default=False)`) – If True, also return the confidence of prediction.

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.
- `confidence` (`numpy array of shape (n_samples,)`) – Only if return_confidence is set to True.

`predict_confidence(X)`

Predict the model’s confidence in making the same prediction under slightly different training sets. See [BPVD20].

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

Returns `confidence` – For each observation, tells how consistently the model would make the same prediction if the training set was perturbed. Return a probability, ranging in [0,1].

Return type `numpy array of shape (n_samples,)`

`predict_proba(X, method='linear', return_confidence=False)`

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’.
- `return_confidence` (`boolean, optional (default=False)`) – If True, also return the confidence of prediction.

Returns `outlier_probability` – 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]. Note it depends on the number of classes, which is by default 2 classes ([proba of normal, proba of outliers]).

Return type `numpy array of shape (n_samples, n_classes)`
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.ecod module

Unsupervised Outlier Detection Using Empirical Cumulative Distribution Functions (ECOD)

class pyod.models.ecod.ECOD(contamination=0.1, n_jobs=1)

Bases: pyod.models.base.BaseDetector

ECOD class for Unsupervised Outlier Detection Using Empirical Cumulative Distribution Functions (ECOD) ECOD is a parameter-free, highly interpretable outlier detection algorithm based on empirical CDF functions. See [BLZH+21] 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_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.

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. 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,)
**explain_outlier**(*ind*, *columns=None*, *cutoffs=None*, *feature_names=None*, *file_name=None*, *file_type=None*)

Plot dimensional outlier graph for a given data point within the dataset.

**Parameters**

- **ind** (*int*) – The index of the data point one wishes to obtain a dimensional outlier graph for.
- **columns** (*list*) – Specify a list of features/dimensions for plotting. If not specified, use all features.
- **cutoffs** (*list of floats in (0., 1), optional (default=[0.95, 0.99])*) – The significance cutoff bands of the dimensional outlier graph.
- **feature_names** (*list of strings*) – The display names of all columns of the dataset, to show on the x-axis of the plot.
- **file_name** (*string*) – The name to save the figure
- **file_type** (*string*) – The file type to save the figure

**Returns**

Plot – The dimensional outlier graph for data point with index ind.

**Return type** matplotlib plot

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

Fit detector. y is ignored in unsupervised methods.

- **param X** The input samples.
- **type X** numpy array of shape (n_samples, n_features)

- **param y** Not used, present for API consistency by convention.
- **type y** Ignored

**Returns** self – Fitted estimator.

**Return type** object

**DEPRECATED**

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

**DEPRECATED**

Fit detector first and then predict whether a particular sample is an outlier or not. y is ignored in unsupervised models.

- **param X** The input samples.
- **type X** numpy array of shape (n_samples, n_features)

- **param y** Not used, present for API consistency by convention.

- **outlier_labels** [*param y*] 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 0.8.0.; it will be replaced by calling fit function first and then accessing labels_. attribute for consistency.

**DEPRECATED**

**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.

- **param X** The input samples.
- **type X** numpy array of shape (n_samples, n_features)

- **param y** Not used, present for API consistency by convention.

- **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 0.8.0.; 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 (bool, optional (default=True)) – 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, return_confidence=False)

Predict if a particular sample is an outlier or not.

Parameters

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

• return_confidence (boolean, optional (default=False)) – If True, also return the confidence of prediction.

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.

• confidence (numpy array of shape (n_samples,)) – Only if return_confidence is set to True.

predict_confidence(X)

Predict the model’s confidence in making the same prediction under slightly different training sets. See [BPVD20].

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

Returns confidence – For each observation, tells how consistently the model would make the same prediction if the training set was perturbed. Return a probability, ranging in [0,1].

Return type numpy array of shape (n_samples,)

predict_proba(X, method='linear', return_confidence=False)

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’.
• `return_confidence` *(boolean, optional (default=False)) – If True, also return the confidence of prediction.*

**Returns outlier_probability** – 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]. Note it depends on the number of classes, which is by default 2 classes ([proba of normal, proba of outliers]).

**Return type** `numpy array of shape (n_samples, n_classes)`

**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.ecod.ecdf(X)`

Calculated the empirical CDF of a given dataset.

**:param X:** The training dataset. 
**:type X:** `numpy array of shape (n_samples, n_features)`

**Returns** `ecdf(X)` – Empirical CDF of X

**Return type** `float`

**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, combination='average', verbose=0, estimator_params=None)`

**Bases:** `pyod.models.base.BaseDetector`

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

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

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

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

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

**Parameters**
• **base_estimator** *(object or None, optional (default=None)) –* The base estimator to fit on random subsets of the dataset. If None, then the base estimator is a LOF detector.

• **n_estimators** *(int, optional (default=10)) –* The number of base estimators in the ensemble.

• **contamination** *(float in (0., 0.5), optional (default=0.1)) –* The amount of contamination of the data set, i.e. the proportion of outliers in the data set. Used when fitting to define the threshold on the decision function.

• **max_features** *(int or float, optional (default=1.0)) –* The number of features to draw from X to train each base estimator.
  - If int, then draw `max_features` features.
  - If float, then draw `max_features * X.shape[1]` features.

• **bootstrap_features** *(bool, optional (default=False)) –* Whether features are drawn with replacement.

• **check_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.

Depreciated since version 0.6.9: **check_estimator** will be removed in pyod 0.8.0.; 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 ignored in unsupervised methods.

Parameters
• X (numpy array of shape (n_samples, n_features)) – The input samples.
• y (Ignored) – Not used, present for API consistency by convention.

Returns self – Fitted estimator.

Return type object

fit_predict(X, y=None)

DEPRECATED

Fit detector first and then predict whether a particular sample is an outlier or not. y is ignored in unsupervised models.

Parameters
• X [numpy array of shape (n_samples, n_features)] The input samples.
• y [Ignored] Not used, present for API consistency by convention.

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 0.8.0; 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 [Ignored] Not used, present for API consistency by convention.

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 0.8.0.; 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** (bool, optional (default=True)) – 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, return_confidence=False)

Predict if a particular sample is an outlier or not.

**Parameters**
- **X** (numpy array of shape (n_samples, n_features)) – The input samples.
- **return_confidence** (boolean, optional (default=False)) – If True, also return the confidence of prediction.

**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.
- **confidence** (numpy array of shape (n_samples,)) – Only if return_confidence is set to True.

`predict_confidence`(X)

Predict the model’s confidence in making the same prediction under slightly different training sets. See [BPVD20].

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

**Returns**
- **confidence** – For each observation, tells how consistently the model would make the same prediction if the training set was perturbed. Return a probability, ranging in [0,1].

**Return type**
- numpy array of shape (n_samples,)

`predict_proba`(X, method='linear', return_confidence=False)

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’.
- **return_confidence** *(boolean, optional (default=False))* – If True, also return the confidence of prediction.

**Returns outlier_probability** – 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]. Note it depends on the number of classes, which is by default 2 classes ([proba of normal, proba of outliers]).

**Return type** numpy array of shape *(n_samples, n_classes)*

**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.hbos module**

Histogram-Based Outlier Detection (HBOS)

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

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

Two versions of HBOS are supported: - Static number of bins: uses a static number of bins for all features. - Automatic number of bins: every feature uses a number of bins deemed to be optimal according to the Birge-Rozenblac method ([BBirgeR06]).

**Parameters**

- **n_bins** *(int or string, optional (default=10))* – The number of bins. “auto” uses the birge-rozenblac method for automatic selection of the optimal number of bins for each feature.
- **alpha** *(float in (0, 1), optional (default=0.1))* – The regularizer for preventing overflow.
- **tol** *(float in (0, 1), optional (default=0.5))* – 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 ignored in unsupervised methods.

**Parameters**

- **X**  (numpy array of shape (n_samples, n_features)) – The input samples.
- **y**  (Ignored) – Not used, present for API consistency by convention.

**Returns**  self – Fitted estimator.

**Return type**  object

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

Fit detector first and then predict whether a particular sample is an outlier or not. y is ignored in unsupervised models.

**Parameters**

- **X**  [numpy array of shape (n_samples, n_features)] The input samples.
- **y**  [Ignored] Not used, present for API consistency by convention.

**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**  object

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

Deprecation warning: This function is deprecated in pyod 0.8.0. It will be removed in pyod 0.8.1. Use fit and decision_function instead.

**Parameters**

- **X**  (numpy array of shape (n_samples, n_features)) – The training input samples.
- **y**  (numpy array of shape (n_samples,)) – The true labels.
- **scoring**  (str) – Scoring metric to evaluate the quality of a fitted anomaly detector. It should be one of the built-in metrics or a custom scorer.

**Returns**  score – Score of the model.

**Return type**  float
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** [Ignored] Not used, present for API consistency by convention.
- **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 0.8.0; 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** *(bool, optional (default=True))* – 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, return_confidence=False)*
Predict if a particular sample is an outlier or not.

**Parameters**
- **X** *(numpy array of shape (n_samples, n_features)) – The input samples.*
- **return_confidence** *(boolean, optional (default=False)) – If True, also return the confidence of prediction.*

**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.*
- **confidence** *(numpy array of shape (n_samples,)) – Only if return_confidence is set to True.*

**predict_confidence** *(X)*
Predict the model’s confidence in making the same prediction under slightly different training sets. See [BPVD20].

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

**Returns**
- **confidence** – For each observation, tells how consistently the model would make the same prediction if the training set was perturbed. Return a probability, ranging in [0,1].

**Return type**
**numpy array of shape (n_samples,)**

**predict_proba** *(X, method='linear', return_confidence=False)*
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’.
- **return_confidence** (boolean, optional(default=False)) – If True, also return the confidence of prediction.

Returns **outlier_probability** – 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]. Note it depends on the number of classes, which is by default 2 classes ([proba of normal, proba of outliers]).

Return type  numpy array of shape (n_samples, n_classes)

**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.

**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 $max_samples \times X.shape[0]$ 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 $max_features \times X.shape[1]$ features.

• bootstrap ($bool$, 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 ignored in unsupervised methods.

**Parameters**

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

**Returns**
- **self** – Fitted estimator.

**Return type** object

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

Fit detector first and then predict whether a particular sample is an outlier or not. y is ignored in unsupervised models.

**Parameters**

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

**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 warning: fit_predict will be removed in pyod 0.8.0; 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** [Ignored] Not used, present for API consistency by convention.
- **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 0.8.0.; 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** *(bool, optional (default=True))* – 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, return_confidence=False)*

Predict if a particular sample is an outlier or not.

- **Parameters**
  - **X** *(numpy array of shape (n_samples, n_features))* – The input samples.
  - **return_confidence** *(boolean, optional (default=False))* – If True, also return the confidence of prediction.

- **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.
  - **confidence** *(numpy array of shape (n_samples,))* – Only if return_confidence is set to True.

### `predict_confidence` *(X)*

Predict the model’s confidence in making the same prediction under slightly different training sets. See [BPVD20].

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

- **Returns**
  - **confidence** – For each observation, tells how consistently the model would make the same prediction if the training set was perturbed. Return a probability, ranging in [0,1].

- **Return type**
  - numpy array of shape (n_samples,)

### `predict_proba` *(X, method='linear', return_confidence=False)*

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’.
- **return_confidence** (boolean, optional (default=False)) – If True, also return the confidence of prediction.

Returns outlier_probability – 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]. Note it depends on the number of classes, which is by default 2 classes ([proba of normal, proba of outliers]).

Return type numpy array of shape (n_samples, n_classes)

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 [BAP02, BRRS00] 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.

Deprecation since version 0.74: **algorithm** is deprecated in PyOD 0.7.4 and will not be possible in 0.7.6. It has to use BallTree for consistency.

- **leaf_size** *(int, optional (default = 30)) – Leaf size passed to BallTree. 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

- **metric_params** *(dict, optional (default = None)) – Additional keyword arguments for the metric function.

- **n_jobs** *(int, optional (default = 1)) – The number of parallel jobs to run for neighbors search. If -1, then the number of jobs is set to the number of CPU cores. Affects only kneighbors and kneighbors_graph methods.

**decision_scores_**

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 ignored in unsupervised methods.

Parameters

- X (numpy array of shape (n_samples, n_features)) – The input samples.
- y (Ignored) – Not used, present for API consistency by convention.

Returns self – Fitted estimator.

Return type object

fit_predict(X, y=None)
DEPRECATED

Fit detector first and then predict whether a particular sample is an outlier or not. y is ignored in unsupervised models.

X [numpy array of shape (n_samples, n_features)] The input samples.
y [Ignored] Not used, present for API consistency by convention.

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 0.8.0; 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 [Ignored] Not used, present for API consistency by convention.

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 0.8.0.; 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 (bool, optional (default=True)) – 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, return_confidence=False)
Predict if a particular sample is an outlier or not.

Parameters
- X (numpy array of shape (n_samples, n_features)) – The input samples.
- return_confidence (boolean, optional (default=False)) – If True, also return the confidence of prediction.

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.
- confidence (numpy array of shape (n_samples,)) – Only if return_confidence is set to True.

predict_confidence(X)
Predict the model’s confidence in making the same prediction under slightly different training sets. See [BPVD20].

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

Returns
- confidence – For each observation, tells how consistently the model would make the same prediction if the training set was perturbed. Return a probability, ranging in [0,1].

Return type
- numpy array of shape (n_samples,)

predict_proba(X, method='linear', return_confidence=False)
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’.

• return_confidence (boolean, optional(default=False)) – If True, also return the confidence of prediction.

Returns outlier_probability – 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]. Note it depends on the number of classes, which is by default 2 classes ([proba of normal, proba of outliers]).

Return type numpy array of shape (n_samples, n_classes)

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.lmdd module
Linear Model Deviation-base outlier detection (LMDD).

class pyod.models.lmdd.LMDD(contamination=0.1, n_iter=50, dis_measure='aad', random_state=None)
Bases: pyod.models.base.BaseDetector

Linear Method for Deviation-based Outlier Detection.

LMDD employs the concept of the smoothing factor which indicates how much the dissimilarity can be reduced by removing a subset of elements from the data-set. Read more in the [BAAR96].

Note: this implementation has minor modification to make it output scores instead of labels.

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_iter (int, optional (default=50)) – Number of iterations where in each iteration, the process is repeated after randomizing the order of the input. Note that n_iter is a very important factor that affects the accuracy. The higher the better the accuracy and the longer the execution.

• dis_measure (str, optional (default='aad')) – Dissimilarity measure to be used in calculating the smoothing factor for points, options available:
  – ’aad’: Average Absolute Deviation
  – ’var’: Variance
  – ’iqr’: Interquartile Range

• 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.

**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 ignored in unsupervised methods.

*Parameters*

- X (numpy array of shape (n_samples, n_features)) – The input samples.
- y *(Ignored)* – Not used, present for API consistency by convention.

*Returns* self – Fitted estimator.

*Return type* object

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

Fit detector first and then predict whether a particular sample is an outlier or not. y is ignored in unsupervised models.

*Parameters*

- X [numpy array of shape (n_samples, n_features)] The input samples.
- y [Ignored] Not used, present for API consistency by convention.

*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.

Deprecated since version 0.6.9: fit_predict will be removed in pyod 0.8.0.; 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** [Ignored] Not used, present for API consistency by convention.
- **scoring** [str, optional (default='roc_auc_score')] Evaluation metric:
  - 'roc_auc_score': ROC score
  - 'prc_n_score': Precision @ rank n score

score : float

Deprecation notice: fit_predict_score will be removed in pyod 0.8.0; 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 (bool, optional (default=True)) – 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*, *return_confidence=False*)

Predict if a particular sample is an outlier or not.

Parameters

- **X** (numpy array of shape (n_samples, n_features)) – The input samples.
- **return_confidence** (boolean, optional (default=False)) – If True, also return the confidence of prediction.

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.
- **confidence** (numpy array of shape (n_samples,)) – Only if return_confidence is set to True.

**predict_confidence**(*X*)

Predict the model’s confidence in making the same prediction under slightly different training sets. See [BPVD20].

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

Returns confidence – For each observation, tells how consistently the model would make the same prediction if the training set was perturbed. Return a probability, ranging in [0,1].

Return type numpy array of shape (n_samples,)

**predict_proba**(*X*, method='linear', *return_confidence=False*)

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’.*
- **return_confidence** *(boolean, optional (default=False)) – If True, also return the confidence of prediction.*

Returns **outlier_probability** – 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]. Note it depends on the number of classes, which is by default 2 classes ([proba of normal, proba of outliers]).

Return type **numpy array of shape (n_samples, n_classes)**

**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.loda module**

Loda: Lightweight on-line detector of anomalies Adapted from tilitools (https://github.com/nicococo/tilitools) by

**class** pyod.models.loda.LODA**(contamination=0.1, n_bins=10, n_random_cuts=100)**

Bases: `pyod.models.base.BaseDetector`


Two versions of LODA are supported:
- Static number of bins: uses a static number of bins for all random cuts.
- Automatic number of bins: every random cut uses a number of bins deemed to be optimal according to the Birge-Rozenblac method ([BBirgeR06]).

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_bins** *(int or string, optional (default = 10)) – The number of bins for the histogram. If set to “auto”, the Birge-Rozenblac method will be used to automatically determine the optimal number of bins.*
- **n_random_cuts** *(int, optional (default = 100)) – The number of random cuts.*
**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,)

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

Fit detector. \( y \) is ignored in unsupervised methods.

**Parameters**

- \( X \) (numpy array of shape (n_samples, n_features)) – The input samples.

- \( y \) (Ignored) – Not used, present for API consistency by convention.

**Returns** self – Fitted estimator.

**Return type** object

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

DEPRECATED

Fit detector first and then predict whether a particular sample is an outlier or not. \( y \) is ignored in unsupervised models.

\( X \) [numpy array of shape (n_samples, n_features)] The input samples.

\( y \) [Ignored] Not used, present for API consistency by convention.

**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: \( \text{fit_predict} \) will be removed in pyod 0.8.0.; it will be replaced by calling \( \text{fit} \) function first and then accessing \( \text{labels}_\) 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 \( (n_{\text{samples}}, n_{\text{features}}) \)] The input samples.

y [Ignored] Not used, present for API consistency by convention.

scoring [str, optional (default='roc_auc_score')] Evaluation metric:

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

score : float

Deprecated since version 0.6.9: fit_predict_score will be removed in pyod 0.8.0.; 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 (bool, optional (default=True)) – 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, return_confidence=False)

Predict if a particular sample is an outlier or not.

Parameters

- X (numpy array of shape \( (n_{\text{samples}}, n_{\text{features}}) \)) – The input samples.
- return_confidence (boolean, optional (default=False)) – If True, also return the confidence of prediction.

Returns

- outlier_labels (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.
- confidence (numpy array of shape \( (n_{\text{samples}},) \)) – Only if return_confidence is set to True.

predict_confidence(X)

Predict the model’s confidence in making the same prediction under slightly different training sets. See [BPVD20].

Parameters X (numpy array of shape \( (n_{\text{samples}}, n_{\text{features}}) \)) – The input samples.

Returns confidence – For each observation, tells how consistently the model would make the same prediction if the training set was perturbed. Return a probability, ranging in [0,1].

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

predict_proba(X, method='linear', return_confidence=False)

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’.
- return_confidence (boolean, optional(default=False)) – If True, also return the confidence of prediction.

Returns outlier_probability – 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]. Note it depends on the number of classes, which is by default 2 classes (proba of normal, proba of outliers).

Return type  
numpy array of shape (n_samples, n_classes)

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, novelty=True)
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:


See the documentation for scipy.spatial.distance for details on these metrics: [http://docs.

- **p** *(integer, optional (default = 2)) – Parameter for the Minkowski metric from
  sklearn.metrics.pairwise.pairwise_distances. When p = 1, this is equivalent to using manhat-
  tan_distance (l1), and euclidean_distance (l2) for p = 2. For arbitrary p, minkowski_distance
  (l_p) is used. See [http://scikit-learn.org/stable/modules/generated/sklearn.metrics.pairwise.
pairwise_distances)

- **metric_params** *(dict, optional (default = None)) – Additional keyword argu-
  ments 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 fit-
  ting 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 neigh-
  bors search. If -1, then the number of jobs is set to the number of CPU cores. Affects only
  kneighbors and kneighbors_graph methods.

- **novelty** *(bool (default=False)) – By default, LocalOutlierFactor is only meant to be
  used for outlier detection (novelty=False). Set novelty to True if you want to use LocalOut-
  lierFactor for novelty detection. In this case be aware that that you should only use predict,
  decision_function and score_samples on new unseen data and not on the training set.

`n_neighbors_`

The actual number of neighbors used for kneighbors queries.

**Type** int

`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 ignored in unsupervised methods.

Parameters

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

• y (Ignored) – Not used, present for API consistency by convention.

Returns self – Fitted estimator.

Return type object

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

Parameters

X [numpy array of shape (n_samples, n_features)] The input samples.

y [Ignored] Not used, present for API consistency by convention.

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 0.8.0; 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 [Ignored] Not used, present for API consistency by convention.

scoring [str, optional (default='roc_auc_score')] Evaluation metric:
• ‘roc_auc_score’: ROC score
• 'pre_n_score': Precision @ rank n score

score : float

Deprecated since version 0.6.9: fit_predict_score will be removed in pyod 0.8.0.; 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 (bool, optional (default=True)) – 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,  
    return_confidence=False
)
Predict if a particular sample is an outlier or not.

Parameters

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

    • return_confidence (boolean, optional(default=False)) – If True, also return the confidence of prediction.

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.

    • confidence (numpy array of shape (n_samples,)) – Only if return_confidence is set to True.

predict_confidence(  
    X
)
Predict the model’s confidence in making the same prediction under slightly different training sets. See [BPVD20].

Parameters

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

Returns

    confidence – For each observation, tells how consistently the model would make the same prediction if the training set was perturbed. Return a probability, ranging in [0,1].

Return type

    numpy array of shape (n_samples,)

predict_proba(  
    X,  
    method='linear', return_confidence=False
)
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’.

• **return_confidence** *(boolean, optional (default=False))* – If True, also return the confidence of prediction.

**Returns** *outlier_probability* – 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]. Note it depends on the number of classes, which is by default 2 classes ([proba of normal, proba of outliers]).

**Return type** *numpy array of shape (n_samples, n_classes)*

**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.loci module**

Local Correlation Integral (LOCI). Part of the codes are adapted from [https://github.com/Cloudy10/loci](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()
>>> clf.fit(X_train)
LOCI(alpha=0.5, contamination=0.1, k=None)
```

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

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

    Parameters

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

    Returns

    anomaly_scores – The anomaly score of the input samples.

    Return type  numpy array of shape (n_samples,)

fit(X, y=None)
    Fit the model using X as training data.

    Parameters

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

    y (Ignored) – Not used, present for API consistency by convention.

    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. y is ignored in unsupervised models.

    Parameters

    X [numpy array of shape (n_samples, n_features)] The input samples.

    y [Ignored] Not used, present for API consistency by convention.
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 0.8.0.; 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 [Ignored] Not used, present for API consistency by convention.
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 0.8.0.; 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 (bool, optional (default=True)) – 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, return_confidence=False)
Predict if a particular sample is an outlier or not.

Parameters
  • X (numpy array of shape (n_samples, n_features)) – The input samples.
  • return_confidence (boolean, optional(default=False)) – If True, also return the confidence of prediction.

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.
  • confidence (numpy array of shape (n_samples,)) – Only if return_confidence is set to True.

predict_confidence(X)
Predict the model’s confidence in making the same prediction under slightly different training sets. See [BPVD20].

Parameters X (numpy array of shape (n_samples, n_features)) – The input samples.
Returns confidence – For each observation, tells how consistently the model would make the same prediction if the training set was perturbed. Return a probability, ranging in [0,1].

Return type numpy array of shape (n_samples,)

predict_proba(X, method='linear', return_confidence=False)
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’.
• return_confidence (boolean, optional(default=False)) – If True, also return the confidence of prediction.

Returns outlier_probability – 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]. Note it depends on the number of classes, which is by default 2 classes ([proba of normal, proba of outliers]).

Return type numpy array of shape (n_samples, n_classes)

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 $n_{samples} \times \text{contamination}$ 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)
```

(continues on next page)
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 ignored in unsupervised methods.

Parameters

- X (numpy array of shape (n_samples, n_features)) – The input samples.
- y (Ignored) – Not used, present for API consistency by convention.

Returns

self – Fitted estimator.

Return type

object

fit_predict(X, y=None)

DEPRECATED

Fit detector first and then predict whether a particular sample is an outlier or not. y is ignored in unsupervised models.

Parameters

- X (numpy array of shape (n_samples, n_features)) – The input samples.
- y (Ignored) – Not used, present for API consistency by convention.

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.

Deprecated since version 0.6.9: fit_predict will be removed in pyod 0.8.0; 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 (Ignored) – Not used, present for API consistency by convention.
- 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 0.8.0.; 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 (bool, optional (default=True)) – 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, return_confidence=False)

Predict if a particular sample is an outlier or not.

Parameters

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

• return_confidence (boolean, optional (default=False)) – If True, also return the confidence of prediction.

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.

• confidence (numpy array of shape (n_samples,)) – Only if return_confidence is set to True.

predict_confidence(X)

Predict the model’s confidence in making the same prediction under slightly different training sets. See [BPVD20].

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

Returns confidence – For each observation, tells how consistently the model would make the same prediction if the training set was perturbed. Return a probability, ranging in [0,1].

Return type numpy array of shape (n_samples,)

predict_proba(X, method='linear', return_confidence=False)

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’.
• **return_confidence** *(boolean, optional (default=False)) – If True, also return the confidence of prediction.*

**Returns outlier_probability** – 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]. Note it depends on the number of classes, which is by default 2 classes ([proba of normal, proba of outliers]).

**Return type** numpy array of shape (n_samples, n_classes)

**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.mad module**

Median Absolute deviation (MAD) Algorithm. Strictly for Univariate Data.

class **pyod.models.mad.MAD** *(threshold=3.5)*

Bases: pyod.models.base.BaseDetector

Median Absolute Deviation: for measuring the distances between data points and the median in terms of median distance. See [BIH93] for details.

**Parameters**

**threshold**(float, optional (default=3.5)) – The modified z-score to use as a threshold. Observations with a modified z-score (based on the median absolute deviation) greater than this value will be classified as outliers.

**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 modified z-score to use as a threshold. Observations with a modified z-score (based on the median absolute deviation) greater than this value will be classified as outliers.

**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. Note that n_features must equal 1.*
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 ignored in unsupervised methods.

Parameters

- **X** *(numpy array of shape (n_samples, n_features))* – The input samples. Note that n_features must equal 1.
- **y** *(Ignored)* – Not used, present for API consistency by convention.

Returns **self** – Fitted estimator.

Return type **object**

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

DEPRECATED

Fit detector first and then predict whether a particular sample is an outlier or not. y is ignored in unsupervised models.

X [numpy array of shape (n_samples, n_features)] The input samples.

y [Ignored] Not used, present for API consistency by convention.

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 0.8.0.; 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 [Ignored] Not used, present for API consistency by convention.

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 0.8.0.; 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** *(bool, optional (default=True))* – 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, return_confidence=False)

Predict if a particular sample is an outlier or not.

**Parameters**

- **X** *(numpy array of shape (n_samples, n_features))* – The input samples.
- **return_confidence** *(boolean, optional (default=False))* – If True, also return the confidence of prediction.

**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.
- **confidence** *(numpy array of shape (n_samples,))* – Only if return_confidence is set to True.

**predict_confidence**(X)

Predict the model’s confidence in making the same prediction under slightly different training sets. See [BPVD20].

**Parameters**

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

**Returns**

- **confidence** – For each observation, tells how consistently the model would make the same prediction if the training set was perturbed. Return a probability, ranging in [0,1].

**Return type** numpy array of shape (n_samples,)

**predict_proba**(X, method='linear', return_confidence=False)

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’.
- **return_confidence** *(boolean, optional (default=False))* – If True, also return the confidence of prediction.

**Returns**

- **outlier_probability** – 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]. Note it depends on the number of classes, which is by default 2 classes ([proba of normal, proba of outliers]).

**Return type** numpy array of shape (n_samples, n_classes)

**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)

```python
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 [BHR04, BRD99] 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** (`bool`) – 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: \[\text{n\_sample + n\_features + 1}\] / 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 sam-
  ples 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 ignored in unsupervised methods.

  Parameters
    • X (numpy array of shape (n_samples, n_features)) – The input samples.
    • y (Ignored) – Not used, present for API consistency by convention.

  Returns
    self – Fitted estimator.

    Return type
    object
**fit_predict** \(X, y=None\)

DEPRECATED

**Fit detector first and then predict whether a particular sample is an outlier or not.** \(y\) is ignored in unsupervised models.

- **X** [numpy array of shape (n_samples, n_features)] The input samples.
- **y** [Ignored] Not used, present for API consistency by convention.

**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 0.8.0; 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** [Ignored] Not used, present for API consistency by convention.
- **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 0.8.0; 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** (bool, optional (default=True)) – 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** \(X, return_confidence=False\)

Predict if a particular sample is an outlier or not.

**Parameters**
- **X** [numpy array of shape (n_samples, n_features)] – The input samples.
- **return_confidence** (boolean, optional (default=False)) – If True, also return the confidence of prediction.

**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.

• **confidence** *(numpy array of shape (n_samples,)) –* Only if return confidence is set to True.

**predict_confidence**(*X*)  
Predict the model’s confidence in making the same prediction under slightly different training sets. See [BPVD20].

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

**Returns**  
* confidence –* For each observation, tells how consistently the model would make the same prediction if the training set was perturbed. Return a probability, ranging in [0,1].

**Return type** numpy array of shape (n_samples,)

**predict_proba**(*X, method='linear', return_confidence=False*)  
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’.

• **return_confidence** *(boolean, optional(default=False)) –* If True, also return the confidence of prediction.

**Returns**  
* outlier_probability –* 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]. Note it depends on the number of classes, which is by default 2 classes ([proba of normal, proba of outliers]).

**Return type** numpy array of shape (n_samples, n_classes)

**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
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,)

2.7. API Reference
**fit**(X, y=None)

Fit detector. `y` is ignored in unsupervised methods.

**Parameters**

- `X` *(numpy array of shape (n_samples, n_features))* – The input samples.
- `y` *(Ignored)* – Not used, present for API consistency by convention.

**Returns**

- **self** – Fitted estimator.

**Return type**

- object

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

DEPRECATED

Fit detector first and then predict whether a particular sample is an outlier or not. `y` is ignored in unsupervised models.

**Parameters**

- `X` *(numpy array of shape (n_samples, n_features))* – The input samples.
- `y` *(Ignored)* – Not used, present for API consistency by convention.

**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 0.8.0.; 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` *(Ignored)* – Not used, present for API consistency by convention.
- `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 0.8.0.; 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` *(bool, optional (default=True))* – 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, return_confidence=False)

Predict if a particular sample is an outlier or not.
Parameters

- **X** (numpy array of shape (n_samples, n_features)) – The input samples.
- **return_confidence** (boolean, optional(default=False)) – If True, also return the confidence of prediction.

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.
- **confidence** (numpy array of shape (n_samples,)) – Only if return_confidence is set to True.

**predict_confidence(X)**

Predict the model’s confidence in making the same prediction under slightly different training sets. See [BPVD20].

Parameters

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

Returns

- **confidence** – For each observation, tells how consistently the model would make the same prediction if the training set was perturbed. Return a probability, ranging in [0,1].

Return type

- numpy array of shape (n_samples,)

**predict_proba(X, method='linear', return_confidence=False)**

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’.
- **return_confidence** (boolean, optional(default=False)) – If True, also return the confidence of prediction.

Returns

- **outlier_probability** – 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]. Note it depends on the number of classes, which is by default 2 classes ([proba of normal, proba of outliers]).

Return type

- numpy array of shape (n_samples, n_classes)

**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.

```python
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)
```

Bases: `pyod.models.base.BaseDetector`

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

Estimate the support of a high-dimensional distribution.


**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** (bool, 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 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 ignored in unsupervised methods.

Parameters

- **X** (numpy array of shape (n_samples, n_features)) – The input samples.
- **y** (Ignored) – Not used, present for API consistency by convention.
- **sample_weight** (array-like, shape (n_samples,)) – Per-sample weights. Rescale C per sample. Higher weights force the classifier to put more emphasis on these points.

Returns self – Fitted estimator.

Return type object
fit_predict($X$, $y=None$)
DEPRECATED

Fit detector first and then predict whether a particular sample is an outlier or not. $y$ is ignored in unsupervised models.

$X$ [numpy array of shape (n_samples, n_features)] The input samples.
$y$ [Ignored] Not used, present for API consistency by convention.

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 0.8.0.; 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.
$y$ [Ignored] Not used, present for API consistency by convention.
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 0.8.0.; 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 (bool, optional (default=True)) – 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($X$, return_confidence=False)
Predict if a particular sample is an outlier or not.

Parameters

  • $X$ (numpy array of shape (n_samples, n_features)) – The input samples.
  • return_confidence (boolean, optional(default=False)) – If True, also return the confidence of prediction.

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.

• **confidence** *(numpy array of shape (n_samples,))* – Only if return_confidence is set to True.

**predict_confidence**(X)

Predict the model’s confidence in making the same prediction under slightly different training sets. See [BPVD20].

**Parameters**

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

**Returns**

confidence – For each observation, tells how consistently the model would make the same prediction if the training set was perturbed. Return a probability, ranging in [0,1].

**Return type**

numpy array of shape (n_samples,)

**predict_proba**(X, method='linear', return_confidence=False)

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’.

• return_confidence *(boolean, optional(default=False))* – If True, also return the confidence of prediction.

**Returns**

outlier_probability – 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]. Note it depends on the number of classes, which is by default 2 classes ([proba of normal, proba of outliers]).

**Return type**

numpy array of shape (n_samples, n_classes)

**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
Principal Component Analysis (PCA) Outlier Detector

```python
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 [BAgg15, BSCSC03] 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:
  ```python
n_components == min(n_samples, n_features)
  ```

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

- **n_selected_components** (`int, optional (default=None)`) – Number of selected principal components for calculating the outlier scores. It is not necessarily equal to the total number of the principal components. If not set, use all principal components.

- **contamination** (`float in (0., 0.5), optional (default=0.1)`) – The amount of contamination of the data set, i.e. the proportion of outliers in the data set. Used when fitting to define the threshold on the decision function.

- **copy** (`bool (default True)`) – If False, data passed to fit are overwritten and running fit(X).transform(X) will not yield the expected results, use fit_transform(X) instead.

- **whiten** (`bool, optional (default False)`) – When True (False by default) the components_ vectors are multiplied by the square root of n_samples and then divided by the singular values to ensure uncorrelated outputs with unit component-wise variances. Whitening will remove some information from the transformed signal (the relative variance scales of the components) but can sometime improve the predictive accuracy of the downstream estimators by making their data respect some hard-wired assumptions.

- **svd_solver** (`string {'auto', 'full', 'arpack', 'randomized'}`) –
  ```python
  auto : the solver is selected by a default policy based on X.shape and n_components: if the input data is larger than 500x500 and the number of components to extract is lower than
  ```
80% of the smallest dimension of the data, then the more efficient ‘randomized’ method is
enabled. Otherwise the exact full SVD is computed and optionally truncated afterwards.

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

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

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

• **tol** (float >= 0, optional (default .0)) – Tolerance for singular values computed
by svd_solver == ‘arpack’.

• **iterated_power** (int >= 0, or ‘auto’, (default ‘auto’)) – Number of iterations
for the power method computed by svd_solver == ‘randomized’.

• **random_state** (int, RandomState instance or None, optional (default None)) – If int, random_state is the seed used by the random number generator; If RandomState instance, random_state is the random number generator; If None, the random number generator is the RandomState instance used by `np.random`. Used when svd_solver == ‘arpack’ or ‘randomized’.

• **weighted** (bool, optional (default=True)) – If True, the eigenvalues are used in
score computation. The eigenvectors with 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

**components_**
Principal axes in feature space, representing the directions of maximum variance in the data. The compo-
nents 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 * 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,)

property 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 ignored in unsupervised methods.

Parameters

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

• y (Ignored) – Not used, present for API consistency by convention.

Returns self – Fitted estimator.
**Return type**  object

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

DEPRECATED

Fit detector first and then predict whether a particular sample is an outlier or not. *y* is ignored in unsupervised models.

- **X**  [numpy array of shape (n_samples, n_features)] The input samples.
- **y**  [Ignored] Not used, present for API consistency by convention.

**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 0.8.0.; 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**  [Ignored] Not used, present for API consistency by convention.
- **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 0.8.0.; 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** *(bool, optional (default=True)) –* 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

**property 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](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*, *return_confidence=False*)

Predict if a particular sample is an outlier or not.
Parameters

- **X** *(numpy array of shape (n_samples, n_features)) – The input samples.*
- **return_confidence** *(boolean, optional (default=False)) – If True, also return the confidence of prediction.*

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.*
- **confidence** *(numpy array of shape (n_samples,)) – Only if return_confidence is set to True.*

`predict_confidence(X)`

Predict the model’s confidence in making the same prediction under slightly different training sets. See [BPVD20].

Parameters

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

Returns **confidence** – For each observation, tells how consistently the model would make the same prediction if the training set was perturbed. Return a probability, ranging in [0,1].

Return type **numpy array of shape (n_samples,)**

`predict_proba(X, method='linear', return_confidence=False)`

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’.*
- **return_confidence** *(boolean, optional (default=False)) – If True, also return the confidence of prediction.*

Returns **outlier_probability** – 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]. Note it depends on the number of classes, which is by default 2 classes ([proba of normal, proba of outliers]).

Return type **numpy array of shape (n_samples, n_classes)**

`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**
**Rotation-based Outlier Detector (ROD)**

**class** `pyod.models.rod.ROD(contamination=0.1, parallel_execution=False)`

Bases: `pyod.models.base.BaseDetector`

Rotation-based Outlier Detection (ROD), is a robust and parameter-free algorithm that requires no statistical distribution assumptions and works intuitively in three-dimensional space, where the 3D-vectors, representing the data points, are rotated about the geometric median two times counterclockwise using Rodrigues rotation formula. The results of the rotation are parallelepipeds where their volumes are mathematically analyzed as cost functions and used to calculate the Median Absolute Deviations to obtain the outlying score. For high dimensions > 3, the overall score is calculated by taking the average of the overall 3D-subspaces scores, that were resulted from decomposing the original data space. See [BABC20] 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.*

- **parallel_execution** *(bool, optional (default=False)) – If set to True, the algorithm will run in parallel, for a better execution time. It is recommended to set this parameter to True ONLY for high dimensional data > 10, and if a proper hardware is available.*

**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 ignored in unsupervised methods.

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

  - **y (Ignored) – Not used, present for API consistency by convention.***
Returns `self` – Fitted estimator.

Return type `object`

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

DEPRECATED

Fit detector first and then predict whether a particular sample is an outlier or not. *y* is ignored in unsupervised models.

- **X** [numpy array of shape (n_samples, n_features)] The input samples.
- **y** [Ignored] Not used, present for API consistency by convention.
- **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 0.8.0.; 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** [Ignored] Not used, present for API consistency by convention.
- **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 0.8.0.; 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** (bool, optional (default=True)) – 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*, *return_confidence=False*)

Predict if a particular sample is an outlier or not.

**Parameters**
- **X** (numpy array of shape (n_samples, n_features)) – The input samples.
- **return_confidence** (boolean, optional (default=False)) – If True, also return the confidence of prediction.

**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.

• **confidence** *(numpy array of shape (n_samples,))* – Only if return_confidence is set to True.

**predict_confidence(X)**
Predict the model’s confidence in making the same prediction under slightly different training sets. See [BPVD20].

**Parameters**

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

**Returns**

- **confidence** – For each observation, tells how consistently the model would make the same prediction if the training set was perturbed. Return a probability, ranging in [0,1].

**Return type** *numpy array of shape (n_samples,)*

**predict_proba(X, method='linear', return_confidence=False)**
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’.

- **return_confidence** *(boolean, optional(default=False))* – If True, also return the confidence of prediction.

**Returns**

- **outlier_probability** – 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]. Note it depends on the number of classes, which is by default 2 classes ([proba of normal, proba of outliers]).

**Return type** *numpy array of shape (n_samples, n_classes)*

**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.rod.angle(v1, v2)**
find the angle between two 3D vectors.

**Parameters**

- **v1** *(list, first vector)* –

- **v2** *(list, second vector)* –
Returns angle

Return type float, the angle

pyod.models.rod.euclidean(v1, v2, c=False)

Find the euclidean distance between two vectors or between a vector and a collection of vectors.

Parameters

• v1 (list, first 3D vector or collection of vectors) –
• v2 (list, second 3D vector) –
• c (bool (default=False), if True, it means the v1 is a list of vectors.) –

Returns

• list of list of euclidean distances if c==True.
• Otherwise float (the euclidean distance)

pyod.models.rod.geometric_median(x, eps=1e-05)

Find the multivariate geometric L1-median by applying Vardi and Zhang algorithm.

Parameters

• x (array-like, the data points) –
• eps (float (default=1e-5), a threshold to indicate when to stop) –

Returns gm

Return type array, Geometric L1-median

pyod.models.rod.mad(costs, median=None)

Apply the robust median absolute deviation (MAD) to measure the inconsistency/variability of the rotation costs.

Parameters

• costs (list of rotation costs) –
• median (float (default=None), MAD median) –

Returns z – the modified z scores

Return type float

pyod.models.rod.process_sub(subspace, gm, median, scaler1, scaler2)

Apply ROD on a 3D subSpace then process it with sigmoid to compare apples to apples

Parameters

• subspace (array-like, 3D subspace of the data) –
• gm (list, the geometric median) –
• median (float, MAD median) –
• scaler1 (obj, MinMaxScaler of Angles group 1) –
• scaler2 (obj, MinMaxScaler of Angles group 2) –

Returns

Return type ROD decision scores with sigmoid applied, gm, scaler1, scaler2
pyod.models.rod.rod_3D(x, gm=None, median=None, scaler1=None, scaler2=None)

Find ROD scores for 3D Data. note that gm, scaler1 and scaler2 will be returned “as they are” and without being changed if the model has been fit already

Parameters

• x (array-like, 3D data points.) –
• gm (list (default=None), the geometric median) –
• median (float (default=None), MAD median) –
• scaler1 (obj (default=None), MinMaxScaler of Angles group 1) –
• scaler2 (obj (default=None), MinMaxScaler of Angles group 2) –

Returns

Return type decision_scores, gm, scaler1, scaler2

pyod.models.rod.rod_nD(X, parallel, gm=None, median=None, data_scaler=None, angles_scalers1=None, angles_scalers2=None)

Find ROD overall scores when Data is higher than 3D: # scale dataset using Robust Scaler # decompose the full space into a combinations of 3D subspaces, # Apply ROD on each combination, # squish scores per subspace, so we compare apples to apples, # calculate average of ROD scores of all subspaces per observation.

Note that if gm, data_scaler, angles_scalers1, angles_scalers2 are None, that means it is a fit() process and they will be calculated and returned to the class to be saved for future prediction. Otherwise, if they are not None, then it is a prediction process.

Parameters

• X (array-like, data points) –
• parallel (bool, True runs the algorithm in parallel) –
• gm (list (default=None), the geometric median) –
• median (list (default=None), MAD medians) –
• data_scaler (obj (default=None), RobustScaler of data) –
• angles_scalers1 (list (default=None), MinMaxScalers of Angles group 1) –
• angles_scalers2 (list (default=None), MinMaxScalers of Angles group 2) –

Returns

Return type ROD decision scores, gm, median, data_scaler, angles_scalers1, angles_scalers2

pyod.models.rod.scale_angles(gammas, scaler1=None, scaler2=None)

Scale all angles in which angles <= 90 degree will be scaled within [0 - 54.7] and angles > 90 will be scaled within [90 - 126]

Parameters

• gammas (list, angles) –
• scaler1 (obj (default=None), MinMaxScaler of Angles group 1) –
• scaler2 (obj (default=None), MinMaxScaler of Angles group 2) –
Returns

Return type: scaled angles, scaler1, scaler2

```
pyod.models.rod.sigmoid(x)
```

Implementation of Sigmoid function

**Parameters**

- **x** (array-like, decision scores) –

**Returns**

Return type: array-like, x after applying sigmoid

---

**pyod.models.sod module**

Subspace Outlier Detection (SOD)

```py
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 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,)

```python
fit(X, y=None)
```  
Fit detector. y is ignored in unsupervised methods.

**Parameters**

- X *(numpy array of shape (n_samples, n_features)) – The input samples.*
- y *(Ignored) – Not used, present for API consistency by convention.*

**Returns**

self – Fitted estimator.

**Return type**

object

```python
fit_predict(X, y=None)
```  
DEPRECATED

**Fit detector first and then predict whether a particular sample is an outlier or not.** y is ignored in unsupervised models.

- X [numpy array of shape (n_samples, n_features)] The input samples.
- y [Ignored] Not used, present for API consistency by convention.

**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 0.8.0.; it will be replaced by calling `fit` function first and then accessing `labels_` attribute for consistency.

```python
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 [Ignored] Not used, present for API consistency by convention.
- 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 0.8.0.; 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.

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


**Parameters**

- **deep (bool, optional (default=True))** – 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, return_confidence=False)

Predict if a particular sample is an outlier or not.

Parameters

- **X** (*numpy array of shape (n_samples, n_features)*) – The input samples.
- **return_confidence** (*boolean, optional (default=False)*) – If True, also return the confidence of prediction.

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.
- **confidence** (*numpy array of shape (n_samples,)*) – Only if return_confidence is set to True.

**predict_confidence**(X)

Predict the model’s confidence in making the same prediction under slightly different training sets. See [BPVD20].

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

Returns **confidence** – For each observation, tells how consistently the model would make the same prediction if the training set was perturbed. Return a probability, ranging in [0,1].

Return type *numpy array of shape (n_samples,)*

**predict_proba**(X, method='linear', return_confidence=False)

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’.
- **return_confidence** (*boolean, optional (default=False)*) – If True, also return the confidence of prediction.

Returns **outlier_probability** – 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]. Note it depends on the number of classes, which is by default 2 classes ([proba of normal, proba of outliers]).

Return type *numpy array of shape (n_samples, n_classes)*

**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.
pyod Documentation, Release 0.9.7


Returns self

Return type object

pyod.models.so_gaal module

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

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

Bases: pyod.models.base.BaseDetector

Single-Objective Generative Adversarial Active Learning.

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

Parameters

- **X** (*numpy array of shape (n_samples, n_features]*) – The input samples.
- **y** (*Ignored*) – Not used, present for API consistency by convention.

Returns **self** – Fitted estimator.

Return type *object*

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

DEPRECATED

Fit detector first and then predict whether a particular sample is an outlier or not. *y* is ignored in unsupervised models.

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

**y** [*Ignored*] Not used, present for API consistency by convention.

**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 0.8.0.; 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** [*Ignored*] Not used, present for API consistency by convention.

**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 0.8.0.; 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** (*bool, optional (default=True]*) – 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, return_confidence=False)*

Predict if a particular sample is an outlier or not.

**Parameters**

- X *(numpy array of shape (n_samples, n_features)) – The input samples.*
- return_confidence *(boolean, optional (default=False)) – If True, also return the confidence of prediction.*

**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.*
- confidence *(numpy array of shape (n_samples,)) – Only if return_confidence is set to True.*

**predict_confidence** *(X)*

Predict the model’s confidence in making the same prediction under slightly different training sets. See [BPVD20].

**Parameters**

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

**Returns**

- confidence – For each observation, tells how consistently the model would make the same prediction if the training set was perturbed. Return a probability, ranging in [0,1].

**Return type** numpy array of shape (n_samples,)

**predict_proba** *(X, method='linear', return_confidence=False)*

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’.*
- return_confidence *(boolean, optional (default=False)) – If True, also return the confidence of prediction.*

**Returns**

- outlier_probability – 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]. Note it depends on the number of classes, which is by default 2 classes ([proba of normal, proba of outliers]).

**Return type** numpy array of shape (n_samples, n_classes)

**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.
pyod.models.sos module

Stochastic Outlier Selection (SOS). Part of the codes are adapted from https://github.com/jeroenjanssens/scikit-sos
class pyod.models.sos.SOS(contamination=0.1, perplexity=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 neighbours. 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 ignored in unsupervised methods.

Parameters

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

Returns self – Fitted estimator.

Return type object

fit_predict(X, y=None)
DEPRECATED

Fit detector first and then predict whether a particular sample is an outlier or not. y is ignored in unsupervised models.

X [numpy array of shape (n_samples, n_features)] The input samples.

y [Ignored] Not used, present for API consistency by convention.

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 0.8.0.; 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 [Ignored] Not used, present for API consistency by convention.
* 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 0.8.0.; 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**

```
depth (bool, optional (default=True)) – 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, return_confidence=False)
```

Predict if a particular sample is an outlier or not.

**Parameters**

* X [numpy array of shape (n_samples, n_features)] – The input samples.
* return_confidence (boolean, optional (default=False)) – If True, also return the confidence of prediction.

**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.
* confidence [numpy array of shape (n_samples,)] – Only if return_confidence is set to True.

```
predict_confidence(X)
```

Predict the model’s confidence in making the same prediction under slightly different training sets. See [BPVD20].

**Parameters**

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

**Returns**

```
confidence – For each observation, tells how consistently the model would make the same prediction if the training set was perturbed. Return a probability, ranging in [0,1].
```

Return type numpy array of shape (n_samples,)
**predict_proba**(*X, method='linear', return_confidence=False*)

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’.*
- **return_confidence** *(boolean, optional (default=False)) – If True, also return the confidence of prediction.*

**Returns**

- **outlier_probability** – 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]. Note it depends on the number of classes, which is by default 2 classes ([proba of normal, proba of outliers]).

**Return type**

- numpy array of shape (n_samples, n_classes)

**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.suod module**

**SUOD**

**class** pyod.models.suod.SUOD *(base_estimators=None, contamination=0.1, combination='average', n_jobs=None, rp_clf_list=None, rp_ng_clf_list=None, rp_flag_global=True, target_dim_frac=0.5, jl_method='basic', bps_flag=True, approx_clf_list=None, approx_ng_clf_list=None, approx_flag_global=True, approx_clf=None, cost_forecast_loc_fit=None, cost_forecast_loc_pred=None, verbose=False)*

**Bases:** pyod.models.base.BaseDetector

SUOD (Scalable Unsupervised Outlier Detection) is an acceleration framework for large scale unsupervised outlier detector training and prediction. See [BZHC+21] for details.

**Parameters**

- **base_estimators** *(list, length must be greater than 1) – A list of base estimators. Certain methods must be present, e.g., fit and predict.*
- **combination** *(str, optional (default=’average’)) – Decide how to aggregate the results from multiple models:
  - ”average” : average the results from all base detectors
  - ”maximization” : output the max value across all base detectors

---

2.7. API Reference 127
- **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_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 the number of jobs that can actually run in parallel.

- **rp_clf_list** (*list, optional (default=None]*) – The list of outlier detection models to use random projection. The detector name should be consistent with PyOD.

- **rp_ng_clf_list** (*list, optional (default=None]*) – The list of outlier detection models NOT to use random projection. The detector name should be consistent with PyOD.

- **rp_flag_global** (*bool, optional (default=True]*) – If set to False, random projection is turned off for all base models.

- **target_dim_frac** (*float in (0., 1), optional (default=0.5]*) – The target compression ratio.

- **jl_method** (*string, optional (default = 'basic')*) – The JL projection method:
  - "basic": each component of the transformation matrix is taken at random in N(0,1).
  - "discrete", each component of the transformation matrix is taken at random in {-1,1}.
  - "circulant": the first row of the transformation matrix is taken at random in N(0,1), and each row is obtained from the previous one by a one-left shift.
  - "toeplitz": the first row and column of the transformation matrix is taken at random in N(0,1), and each diagonal has a constant value taken from these first vector.

- **bps_flag** (*bool, optional (default=True]*) – If set to False, balanced parallel scheduling is turned off.

- **approx_clf_list** (*list, optional (default=None]*) – The list of outlier detection models to use pseudo-supervised approximation. The detector name should be consistent with PyOD.

- **approx_ng_clf_list** (*list, optional (default=None]*) – The list of outlier detection models NOT to use pseudo-supervised approximation. The detector name should be consistent with PyOD.

- **approx_flag_global** (*bool, optional (default=True]*) – If set to False, pseudo-supervised approximation is turned off.

- **approx_clf** (*object, optional (default: sklearn RandomForestRegressor]*) – The supervised model used to approximate unsupervised models.

- **cost_forecast_loc_fit** (*str, optional*) – The location of the pretrained cost prediction forecast for training.

- **cost_forecast_loc_pred** (*str, optional*) – The location of the pretrained cost prediction forecast for prediction.

- **verbose** (*int, optional (default=0]*) – Controls the verbosity of the building process.

**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 detectors.

    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 ignored in unsupervised methods.

    Parameters

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

    • y (Ignored) – Not used, present for API consistency by convention.

    Returns self – Fitted estimator.

    Return type  object

fit_predict(X, y=None)
    DEPRECATED

    Fit detector first and then predict whether a particular sample is an outlier or not. y is ignored in unsupervised models.

    X [numpy array of shape (n_samples, n_features)] The input samples.

    y [Ignored] Not used, present for API consistency by convention.

    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 0.8.0.; 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 [Ignored] Not used, present for API consistency by convention.

    scoring [str, optional (default='roc_auc_score')] Evaluation metric:
• 'roc_auc_score': ROC score
• 'pre_n_score': Precision @ rank n score

score : float

Deprecated since version 0.6.9: fit_predict_score will be removed in pyod 0.8.0.; 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

Parameters

Parameters

Parameters

Return type
mapping of string to any

predict(X, return_confidence=False)
Predict if a particular sample is an outlier or not.

Parameters

Parameters

Parameters

Parameters

Returns

Returns

Returns

Return type

predict_confidence(X)
Predict the model’s confidence in making the same prediction under slightly different training sets. See [BPVD20].

Parameters

Parameters

Parameters

Returns

Returns

Return type

predict_proba(X, method='linear', return_confidence=False)
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

Parameters

Parameters
• **method** (str, optional (default='linear')) – probability conversion method. It must be one of ‘linear’ or ‘unify’.

• **return_confidence** (boolean, optional (default=False)) – If True, also return the confidence of prediction.

**Returns outlier_probability** – 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]. Note it depends on the number of classes, which is by default 2 classes ([proba of normal, proba of outliers]).

**Return type** numpy array of shape (n_samples, n_classes)

**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.vae module**

Variational Auto Encoder (VAE) and beta-VAE for Unsupervised Outlier Detection


**class** pyod.models.vae.VAE(encoder_neurons=None, decoder_neurons=None, latent_dim=2, hidden_activation='relu', output_activation='sigmoid', loss=<function mean_squared_error>, optimizer='adam', epochs=100, batch_size=32, dropout_rate=0.2, l2_regularizer=0.1, validation_size=0.1, preprocessing=True, verbose=1, random_state=None, contamination=0.1, gamma=1.0, capacity=0.0)

**Bases:** pyod.models.base.BaseDetector

Variational auto encoder Encoder maps X onto a latent space Z Decoder samples Z from N(0,1) VAE_loss = Reconstruction_loss + KL_loss


beta VAE In Loss, the emphasis is on KL_loss and capacity of a bottleneck: VAE_loss = Reconstruction_loss + gamma*KL_loss


**Parameters**

• **encoder_neurons** (list, optional (default=[128, 64, 32])) – The number of neurons per hidden layer in encoder.

• **decoder_neurons** (list, optional (default=[32, 64, 128])) – The number of neurons per hidden layer in decoder.
• **hidden_activation** *(str, optional (default='relu'))* – Activation function to use for hidden layers. All hidden layers are forced to use the same type of activation. See https://keras.io/activations/

• **output_activation** *(str, optional (default='sigmoid'))* – Activation function to use for output layer. See https://keras.io/activations/

• **loss** *(str or obj, optional (default=keras.losses.mean_squared_error))* – String (name of objective function) or objective function. See https://keras.io/losses/

• **gamma** *(float, optional (default=1.0))* – Coefficient of beta VAE regime. Default is regular VAE.

• **capacity** *(float, optional (default=0.0))* – Maximum capacity of a loss bottle neck.

• **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))* – verbose mode.
  – 0 = silent
  – 1 = progress bar
  – 2 = one line per epoch.

  For verbose >= 1, model summary may be printed.

• **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 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_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. y is ignored in unsupervised models.

Parameters

• X [numpy array of shape (n_samples, n_features)] The input samples.

y [Ignored] Not used, present for API consistency by convention.

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.
Degradated since version 0.6.9: *fit_predict* will be removed in pyod 0.8.0.; 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** [Ignored] Not used, present for API consistency by convention.
- **scoring** [str, optional (default='roc_auc_score')] Evaluation metric:
  - 'roc_auc_score': ROC score
  - 'prc_n_score': Precision @ rank n score

-score : float

Degradated since version 0.6.9: *fit_predict_score* will be removed in pyod 0.8.0.; 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** (bool, optional (default=True)) – 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, return_confidence=False)

Predict if a particular sample is an outlier or not.

- **Parameters**
  - **X** (numpy array of shape (n_samples, n_features)) – The input samples.
  - **return_confidence** (boolean, optional(default=False)) – If True, also return the confidence of prediction.

- **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.
  - **confidence** (numpy array of shape (n_samples,)) – Only if return_confidence is set to True.

**predict_confidence**(X)

Predict the model’s confidence in making the same prediction under slightly different training sets. See [BPVD20].

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

- **Returns**
  - **confidence** – For each observation, tells how consistently the model would make the same prediction if the training set was perturbed. Return a probability, ranging in [0,1].

- **Return type** numpy array of shape (n_samples,)
**predict_proba** *(X, method='linear', return_confidence=False)*

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’.*
- **return_confidence** *(boolean, optional (default=False)) – If True, also return the confidence of prediction.*

**Returns**

- **outlier_probability** – 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]. Note it depends on the number of classes, which is by default 2 classes ([proba of normal, proba of outliers]).

**Return type** *numpy array of shape (n_samples, n_classes)*

**sampling**(args)

Reparametrisation by sampling from Gaussian, N(0,1) To sample from epsilon = Norm(0,1) instead of from likelihood Q(z|X) with latent variables z: \( z = z_{\text{mean}} + \sqrt{\text{var}} \times \epsilon \).\)

**Parameters**

- **args** *(tensor)* – Mean and log of variance of Q(z|X).

**Returns**

- **z** – Sampled latent variable.

**Return type** *tensor*

**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*

**vae_loss**(inputs, outputs, z_mean, z_log)

Loss = Recreation loss + Kullback-Leibler loss for probability function divergence (ELBO). gamma > 1 and capacity != 0 for beta-VAE
XGBOD class for outlier detection. It first uses the passed in unsupervised outlier detectors to extract richer representation of the data and then concatenates 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 perform 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** *(bool) – 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** *(#) –
• **kwargs (dict, optional) – Keyword arguments for XGBoost Booster object. Full documentation of parameters can be found here: https://github.com/dmlc/xgboost/blob/master/doc/parameter.rst. Attempting to set a parameter via the constructor args and **kwargs dict simultaneously will result in a TypeError.

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

**n_detector_**
The number of unsupervised of detectors used.

  Type int

**clf_**
The XGBoost classifier.

  Type 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,)

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

  Type int, either 0 or 1

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

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

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

  Returns anomaly_scores – The anomaly score of the input samples.
  Return type numpy array of shape (n_samples,)

**fit**(X, y)
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. y is ignored in unsupervised models.

- **X** [numpy array of shape (n_samples, n_features)] The input samples.
- y [Ignored] Not used, present for API consistency by convention.

**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 0.8.0.; 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 (Ignored) – Not used, present for API consistency by convention.
- **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 (bool, optional (default=True)) – If True, will return the parameters for this estimator and contained subobjects that are estimators.

**Returns** params – Parameter names mapped to their values.

Return type mapping of string to any

**predict**(*X*)

Predict if a particular sample is an outlier or not. Calling xgboost **predict** function.

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

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

Return type numpy array of shape (n_samples,)

**predict_confidence**(*X*)

Predict the model’s confidence in making the same prediction under slightly different training sets. See [BPVD20].
**Parameters** \( X \) (*numpy array of shape \((n_{\text{samples}}, n_{\text{features}})\)*) – The input samples.

**Returns** confidence – For each observation, tells how consistently the model would make the same prediction if the training set was perturbed. Return a probability, ranging in \([0,1]\).

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

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

**Parameters** \( X \) (*numpy array of shape \((n_{\text{samples}}, n_{\text{features}})\)*) – 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 \((n_{\text{samples}},)\)

**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 \(<\text{component}>__<\text{parameter}>\) so that it’s possible to update each component of a nested object.


**Returns** self

**Return type** object

### Module contents

### References

#### 2.7.2 Utility Functions

**pyod.utils.data module**

Utility functions for manipulating data

**pyod.utils.data.check_consistent_shape\((X_{\text{train}}, y_{\text{train}}, X_{\text{test}}, y_{\text{test}}, y_{\text{train\_pred}}, y_{\text{test\_pred}})\)**
Internal shape to check input data shapes are consistent.

**Parameters**

- \( X_{\text{train}} \) (*numpy array of shape \((n_{\text{samples}}, n_{\text{features}})\)*) – The training samples.
- \( y_{\text{train}} \) (*list or array of shape \((n_{\text{samples}},)\)*) – The ground truth of training samples.
- \( X_{\text{test}} \) (*numpy array of shape \((n_{\text{samples}}, n_{\text{features}})\)*) – The test samples.
- \( y_{\text{test}} \) (*list or array of shape \((n_{\text{samples}},)\)*) – The ground truth of test samples.
- \( y_{\text{train\_pred}} \) (*numpy array of shape \((n_{\text{samples}}, n_{\text{features}})\)*) – The predicted binary labels of the training samples.
- \( y_{\text{test\_pred}} \) (*numpy array of shape \((n_{\text{samples}}, n_{\text{features}})\)*) – The predicted binary labels of the test samples.
Returns

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

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

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

**Parameters**

- **clf_name** *(str) – The name of the detector.*
- **y** *(list or numpy array of shape (n_samples,)) – The ground truth. Binary (0: inliers, 1: outliers).*
- **y_pred** *(list or numpy array of shape (n_samples,)) – The raw outlier scores as returned by a fitted model.*

**pyod.utils.data.generate_data** *(n_train=1000, n_test=500, n_features=2, contamination=0.1, train_only=False, offset=10, 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. “X_train, X_test, y_train, y_test” are returned.

**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, X_test, y_train, y_test”, while passing behaviour=’old’ returns “X_train, y_train, X_test, 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.

Deprecation since version 0.7.2.: behaviour parameter will be deprecated in 0.7.2 and removed in 0.9.0.
• `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`.*

**Returns**

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

`pyod.utils.data.generate_data_categorical(n_train=1000, n_test=500, n_features=2, n_informative=2, n_category_in=2, n_category_out=2, contamination=0.1, shuffle=True, random_state=None)`

Utility function to generate synthesized categorical data.

**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 for each sample.*
- `n_informative` *(int in (1, n_features), optional (default=2)) – The number of informative features in the outlier points. The higher the easier the outlier detection should be. Note that n_informative should not be less than or equal n_features.*
- `n_category_in` *(int in (1, n_inliers), optional (default=2)) – The number of categories in the inlier points.*
- `n_category_out` *(int in (1, n_outliers), optional (default=2)) – The number of categories in the outlier points.*
- `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.*
- `shuffle` *(bool, optional(default=True)) – If True, inliers will be shuffled which makes more noisy distribution.*
- `random_state` *(int, RandomState instance or None, optional (default=None)) – If int, random_state is the seed used by the random number generator; If RandomState instance, random_state is the random number generator; If None, the random number generator is the RandomState instance used by `np.random`.*

**Returns**

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

`pyod.utils.data.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_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

```python
def 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.
```

```python
def 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

```python
def pairwise_distances_no_broadcast(X, Y):
    # Utility function to calculate row-wise euclidean distance of two matrix. Different from pair-wise calculation, this function would not broadcast.
    # For instance, X and Y are both (4,3) matrices, the function would return a distance vector with shape (4,), instead of (4,4).
    # Parameters
    # • X (array of shape (n_samples, n_features)) – First input samples
```
• Y (array of shape (n_samples, n_features)) – Second input samples

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

Return type array of shape (n_samples,)

pyod.utils.stat_models.pearsonr_mat(mat, w=None)
Utility function to calculate pearson matrix (row-wise).

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

Returns pear_mat – Row-wise pearson score matrix.

Return type numpy array of shape (n_samples, n_samples)

pyod.utils.stat_models.wpearsonr(x, y, w=None)
Utility function to calculate the weighted Pearson correlation of two samples.

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

Parameters
• x (array, shape (n,)) – Input x.
• y (array, shape (n,)) – Input y.
• w (array, shape (n,)) – Weights w.

Returns scores – Weighted Pearson Correlation between x and y.

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

pyod.utils.utility module

A set of utility functions to support outlier detection.

pyod.utils.utility.argmaxn(value_list, n, order='desc')
Return the index of top n elements in the list if order is set to 'desc', otherwise return the index of n smallest ones.

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

Returns index_list – The index of the top n elements.

Return type numpy array of shape (n,)

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

Parameters detector (pyod.models) – Detector instance for which the check is performed.
pyod.utils.utility.check_parameter(param, low=-2147483647, high=2147483647, param_name='', include_left=False, include_right=False)

Check if an input is within the defined range.

Parameters

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

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

Return type **bool** or raise errors

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

Randomly draw feature indices. Internal use only.

Modified from sklearn/ensemble/bagging.py

Parameters

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

Returns **feature_indices** – Indices for features to bag

Return type **numpy array, shape (n_samples,)**

pyod.utils.utility.generate_indices(random_state, bootstrap, n_population, n_samples)

Draw randomly sampled indices. Internal use only.

See sklearn/ensemble/bagging.py

Parameters

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

Returns **indices** – randomly drawn indices

Return type **numpy array, shape (n_samples,)**
**pyod.utils.utility.get_diff_elements**(li1, li2)
get the elements in li1 but not li2, and vice versa

**Parameters**
- li1 (list or numpy array) – Input list 1.
- li2 (list or numpy array) – Input list 2.

**Returns** difference – The difference between li1 and li2.

**Return type** list

**pyod.utils.utility.get_intersection**(lst1, lst2)
get the overlapping between two lists

**Parameters**
- li1 (list or numpy array) – Input list 1.
- li2 (list or numpy array) – Input list 2.

**Returns** difference – The overlapping between li1 and li2.

**Return type** list

**pyod.utils.utility.get_label_n**(y, y_pred, n=None)
Function to turn raw outlier scores into binary labels by assign 1 to top n outlier scores.

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

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

**Return type** numpy array of shape (n_samples,)

**Examples**

```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])
```

**pyod.utils.utility.get_list_diff**(li1, li2)
get the elements in li1 but not li2. li1-li2

**Parameters**
- li1 (list or numpy array) – Input list 1.
- li2 (list or numpy array) – Input list 2.

**Returns** difference – The difference between li1 and li2.

**Return type** list
pyod.utils.utility.get_optimal_n_bins(X, upper_bound=None, epsilon=1)
Determine optimal number of bins for a histogram using the Birge Rozenblac method (see [BBirgeR06] for details.)

See https://doi.org/10.1051/ps:2006001

Parameters

- **X** (array-like of shape (n_samples, n_features)) – The samples to determine the optimal number of bins for.
- **upper_bound** (int, *default=None*) – The maximum value of n_bins to be considered. If set to None, np.sqrt(X.shape[0]) will be used as upper bound.
- **epsilon** (float, *default = 1*) – A stabilizing term added to the logarithm to prevent division by zero.

Returns **optimal_n_bins** – The optimal value of n_bins according to the Birge Rozenblac method

Return type int

pyod.utils.utility.invert_order(scores, method='multiplication')
Invert the order of a list of values. The smallest value becomes the largest in the inverted list. This is useful while combining multiple detectors since their score order could be different.

Parameters

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

Returns **inverted_scores** – The inverted list

Return type numpy array of shape (n_samples,)

Examples

```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 outlier scores to binary labels (0 or 1).

Parameters

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

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

Return type numpy array of shape (n_samples,)

`pyod.utils.utility.standardizer(X, X_t=None, keep_scalar=False)`

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

Parameters

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

Returns

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

2.7.3 Module contents

2.8 Known Issues & Warnings

This is the central place to track known issues.

2.8.1 Installation

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

SO_GAAL and MO_GAAL may only work under Python 3.5+.

2.8.3 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 anomalies/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.

2.9 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.

## 2.10 Citations & Achievements

### 2.10.1 Citing PyOD

PyOD paper is published in *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:


### 2.10.2 Scientific Work Using or Referencing PyOD

We are appreciated that PyOD has been increasingly referred and cited in scientific works. Since its release, PyOD has been used in hundred of academic projects. See an incomplete list here.

### 2.10.3 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

**KDnuggets**: An Overview of Outlier Detection Methods from PyOD

**Towards Data Science**: Anomaly Detection for Dummies

**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

Lecture on anomaly detection with PyOD by Dr. Hadi Fanaee: Anomaly Detection Lecture

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

Miscellaneous:

- PythonAwesome
- awesome-python
- PapersWithCode

### 2.11 Frequently Asked Questions

#### 2.11.1 What is the Next?

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

- GPU support (it is noted that keras with TensorFlow backend will automatically run on GPU; auto_encoder_example.py takes around 96.95 seconds on a RTX 2060 GPU).
- 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.

#### 2.11.2 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 :)

2.11.3 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.

2.12 About us

2.12.1 Core Development Team

Yue Zhao (Ph.D. Student @ Carnegie Mellon University):
- Initialized the project in 2017
- Homepage
- LinkedIn (Yue Zhao)

Zain Nasrullah (Data Scientist at RBC; MSc in Computer Science from University of Toronto):
- Joined in 2018
- LinkedIn (Zain Nasrullah)

Winston (Zheng) Li (Founder of arima, Part-time Instructor @ Northeastern University):
- Joined in 2018
- LinkedIn (Winston Li)

Yahya Almardeny (Software Systems & Machine Learning Engineer @ TSSG):
- Joined in 2019
- LinkedIn (Yahya Almardeny)

Antônio Pedro Camargo (University of Campinas)
- Joined in 2020 (our Conda maintainer)
- GitHub (Antônio Pedro Camargo)

Dr Andrij Vasylenko (Research Associate @ University of Liverpool)
- Joined in 2020 (implemented the VAE and extend to Beta-VAE)
- Homepage (Dr Andrij Vasylenko)

Roel Bouman (Ph.D. Student @ Radboud University):
- Joined in 2021
- LinkedIn (Roel Bouman)

Rafał Bodziony (Data Scientist):
• Joined in 2021 (implemented DeepSVDD)
• LinkedIn (Roel Bouman)

References


python, 148
pyod, 139
pyod.models, 139
pyod.models.abod, 20
pyod.models.auto_encoder, 23
pyod.models.auto_encoder_torch, 27
pyod.models.base, 17
pyod.models.cblof, 42
pyod.models.cof, 47
pyod.models.combination, 50
pyod.models.copod, 52
pyod.models.deep_svd, 55
pyod.models.ecod, 59
pyod.models.feature_bagging, 62
pyod.models.hbos, 66
pyod.models.iforest, 69
pyod.models.knn, 73
pyod.models.lmdd, 77
pyod.models.loci, 87
pyod.models.loda, 80
pyod.models.lof, 83
pyod.models.lscp, 90
pyod.models.mad, 94
pyod.models.mcd, 97
pyod.models.mo_gaal, 101
pyod.models.ocsvm, 104
pyod.models.pca, 108
pyod.models.rod, 113
pyod.models.so_gaal, 121
pyod.models.sod, 118
pyod.models.sos, 124
pyod.models.suod, 127
pyod.models.vae, 131
pyod.models.xgbod, 136
pyod.utils.data, 139
pyod.utils.example, 143
pyod.utils.stat_models, 143
pyod.utils.utility, 144
INDEX

A
ABOD (class in pyod.models.abod), 20
add_module() (pyod.models.auto_encoder_torch.inner_autoencoder method), 31
angle() (in module pyod.models.rod), 115
aom() (in module pyod.models.combination), 50
apply() (pyod.models.auto_encoder_torch.inner_autoencoder method), 31
argmaxn() (in module pyod.utils.utility), 144
AutoEncoder (class in pyod.models.auto_encoder), 23
AutoEncoder (class in pyod.models.auto_encoder_torch), 27
average() (in module pyod.models.combination), 50

B
BaseDetector (class in pyod.models.base), 17
bfloat16() (pyod.models.auto_encoder_torch.inner_autoencoder method), 32
bin_edges_ (pyod.models.hbos.HBOS attribute), 66
buffers() (pyod.models.auto_encoder_torch.inner_autoencoder method), 32

c
CBLOF (class in pyod.models.cblof), 42
check_consistent_shape() (in module pyod.utils.data), 139
check_detector() (in module pyod.utils.utility), 144
check_parameter() (in module pyod.utils.utility), 144
children() (pyod.models.auto_encoder_torch.inner_autoencoder method), 32
clf_ (pyod.models.xgbod.XGBOD attribute), 137
cluster_centers_ (pyod.models.cblof.CBLOF attribute), 44
cluster_labels_ (pyod.models.cblof.CBLOF attribute), 43
cluster_sizes_ (pyod.models.cblof.CBLOF attribute), 44
clustering_estimator_ (pyod.models.cblof.CBLOF attribute), 43
coeff_ (pyod.models.ocsvm.OC SVM attribute), 105
COF (class in pyod.models.cof), 47
components_ (pyod.models.pca.PCA attribute), 109
compression_rate_ (pyod.models.auto_encoder.AutoEncoder attribute), 24
compression_rate_ (pyod.models.auto_encoder_torch.AutoEncoder attribute), 28
covariance_ (pyod.models.mcd.MCD attribute), 98
cpu() (pyod.models.auto_encoder_torch.inner_autoencoder method), 32
cuda() (pyod.models.auto_encoder_torch.inner_autoencoder method), 32
d
data_visualize() (in module pyod.utils.example), 143
decision_function() (pyod.models.abod.ABOD method), 20
decision_function() (pyod.models.auto_encoder.AutoEncoder attribute), 24
decision_function() (pyod.models.auto_encoder_torch.AutoEncoder attribute), 28
decision_function() (pyod.models.base.BaseDetector method), 17
decision_function() (pyod.models.cblof.CBLOF method), 44
decision_function() (pyod.models.cof.COF method), 47
decision_function() (pyod.models.copod.COPOD method), 52
decision_function() (pyod.models.deep_svdd.DeepSVDD method), 56
decision_function() (pyod.models.ecod.ECOD method), 59
decision_function() (pyod.models.feature_bagging.FeatureBagging method), 64
decision_function() (pyod.models.hbos.HBOS method), 67
decision_function() (pyod.models.iforest.IForest method), 71
decision_function() (pyod.models.knn.KNN method), 75
decision_function() (pyod.models.lmdd.LMDD method), 78
decision_function() (pyod.models.loci.LOCI method), 82
decision_function() (pyod.models.loda.LODA method), 87
decision_function() (pyod.models.lscp.LSCP method), 92
decision_function() (pyod.models.mad.MAD method), 94
decision_function() (pyod.models.mcd.MCD method), 93
decision_function() (pyod.models.mo_gaal.MO_GAAL method), 101
decision_function() (pyod.models.rod.ROD method), 113
decision_function() (pyod.models.sod.SOD method), 118
decision_function() (pyod.models.sos.SOS method), 125
decision_function() (pyod.models.suod.SUOD method), 129
decision_function() (pyod.models.vae.VAE method), 133
decision_scores_ (pyod.models.abod.ABOD attribute), 20
decision_scores_ (pyod.models.auto_encoder.AutoEncoder attribute), 24
decision_scores_ (pyod.models.auto_encoder_torch.AutoEncoder attribute), 28
decision_scores_ (pyod.models.automl.AutoML attribute), 31
decision_scores_ (pyod.models.base.BaseDetector attribute), 16
decision_scores_ (pyod.models.cblof.CBLOF attribute), 44
decision_scores_ (pyod.models.coif.COF attribute), 47
decision_scores_ (pyod.models.feature_bagging.FeatureBagging attribute), 59
decision_scores_ (pyod.models.iforest.IForest attribute), 71
decision_scores_ (pyod.models.knn.KNN attribute), 74
decision_scores_ (pyod.models.lmdd.LMDD attribute), 78
decision_scores_ (pyod.models.loci.LOCI attribute), 87
decision_scores_ (pyod.models.loda.LODA attribute), 80
decision_scores_ (pyod.models.lscp.LSCP attribute), 91
decision_scores_ (pyod.models.mad.MAD attribute), 94
decision_scores_ (pyod.models.mcd.MCD attribute), 98
decision_scores_ (pyod.models.mo_gaal.MO_GAAL attribute), 101
decision_scores_ (pyod.models.ocsvm.OCSVM attribute), 105
decision_scores_ (pyod.models.pca.PCA attribute), 110
decision_scores_ (pyod.models.rod.ROD attribute), 113
decision_scores_ (pyod.models.sod.SOD attribute), 118
decision_scores_ (pyod.models.sos.SOS attribute), 124
decision_scores_ (pyod.models.suod.SUOD attribute), 128
decision_scores_ (pyod.models.vae.VAE attribute), 133
decision_scores_ (pyod.models.xgbod.XGBOD attribute), 137
decision_scores_ (pyod.models.deep_svdd.DeepSVDD attribute), 56
decision_scores_ (pyod.models.ecod.OCVS attribute), 59
decision_scores_ (pyod.models.feature_bagging.FeatureBagging attribute), 63
decision_scores_ (pyod.models.hbos.HBOS attribute), 67
decision_scores_ (pyod.models.iforest.IForest attribute), 71
decision_scores_ (pyod.models.knn.KNN attribute), 74
decision_scores_ (pyod.models.lmdd.LMDD attribute), 78
decision_scores_ (pyod.models.loci.LOCI attribute), 87
decision_scores_ (pyod.models.loda.LODA attribute), 80
decision_scores_ (pyod.models.lof.LOF attribute), 84
decision_scores_ (pyod.models.lscp.LSCP attribute), 91
decision_scores_ (pyod.models.mad.MAD attribute), 94
decision_scores_ (pyod.models.mcd.MCD attribute), 98
decision_scores_ (pyod.models.mo_gaal.MO_GAAL attribute), 101
decision_scores_ (pyod.models.ocsvm.OCSVM attribute), 105
decision_scores_ (pyod.models.pca.PCA attribute), 110
decision_scores_ (pyod.models.rod.ROD attribute), 113
decision_scores_ (pyod.models.sod.SOD attribute), 118
decision_scores_ (pyod.models.sos.SOS attribute), 124
decision_scores_ (pyod.models.suod.SUOD attribute), 128
decision_scores_ (pyod.models.vae.VAE attribute), 133
decision_scores_ (pyod.models.xgbod.XGBOD attribute), 137
DeepSVDD (class in pyod.models.deep_svdd), 55
double() (pyod.models.auto_encoder_torch.inner_autoencoder method), 33
dual_coef_ (pyod.models.ocsvm.OCSVM attribute), 104
dump_patches (pyod.models.auto_encoder_torch.inner_autoencoder attribute), 33
ecdf() (in module pyod.models.ecod), 62
fit_predict_score()
(pyod.models.auto_encoder_torch.AutoEncoder method), 29
fit_predict_score()
(pyod.models.base.BaseDetector method), 18
fit_predict_score()
(pyod.models.cblof.CBLOF method), 45
fit_predict_score()
(pyod.models.cof.COF method), 48
fit_predict_score()
(pyod.models.copod.COPOD method), 53
fit_predict_score()
(pyod.models.deep_svdd.DeepSVDD method), 57
fit_predict_score()
(pyod.models.ecod.ECOD method), 60
fit_predict_score()
(pyod.models.feature_bagging.FeatureBagging method), 64
fit_predict_score()
(pyod.models.hbos.HBOS method), 67
fit_predict_score()
(pyod.models.iforest.IForest method), 71
fit_predict_score()
(pyod.models.knn.KNN method), 75
fit_predict_score()
(pyod.models.lmdd.LMDD method), 78
fit_predict_score()
(pyod.models.loci.LOCI method), 89
fit_predict_score()
(pyod.models.loda.LODA method), 81
fit_predict_score()
(pyod.models.lof.LOF method), 85
fit_predict_score()
(pyod.models.lscp.LSCP method), 92
fit_predict_score()
(pyod.models.mad.MAD method), 95
fit_predict_score()
(pyod.models.mcd.MCD method), 99
fit_predict_score()
(pyod.models.mo_gaal.MO_GAAL method), 102
fit_predict_score()
(pyod.models.ocsvm.OCSVM method), 106
fit_predict_score()
(pyod.models.pca.PCA method), 111
fit_predict_score()
(pyod.models.rod.ROD method), 114
fit_predict_score()
(pyod.models.so_gaal.SO_GAAL method), 122
fit_predict_score()
(pyod.models.sod.SOD method), 119
fit_predict_score()
(pyod.models.sos.SOS method), 126
fit_predict_score()
(pyod.models.suod.SUOD method), 129
fit_predict_score()
(pyod.models.vae.VAE method), 134
float()
(pyod.models.auto_encoder_torch.inner_autoencoder method), 33
forward()
(pyod.models.auto_encoder_torch.inner_autoencoder method), 34
functions
(pyod.models.auto_encoder_torch.PyODDataset attribute), 31
G
generate_bagging_indices() (in module pyod.utils.utility), 145
generate_data() (in module pyod.utils.data), 140
generate_data_categorical() (in module pyod.utils.data), 141
generate_data_clusters() (in module pyod.utils.data), 141
generate_indices() (in module pyod.utils.utility), 145
generate_labels() (in module pyod.utils.utility), 146
generate_median() (in module pyod.utils.rod), 116
get_buffer() (pyod.models.auto_encoder_torch.inner_autoencoder method), 34
get_diff_elements() (in module pyod.utils.utility), 145
get_extra_state() (pyod.models.auto_encoder_torch.inner_autoencoder method), 34
get_intersection() (in module pyod.utils.utility), 146
get_label_n() (in module pyod.utils.utility), 146
get_list_diff() (in module pyod.utils.utility), 146
get_optimal_n_bins() (in module pyod.utils.utility), 146
get_outliers_inliers() (in module pyod.utils.data), 142
get_parameter() (pyod.models.auto_encoder_torch.inner_autoencoder method), 34
get_params() (pyod.models.abod.ABOD method), 21
get_params() (pyod.models.auto_encoder.AutoEncoder method), 25
get_params() (pyod.models.auto_encoder_torch.AutoEncoder method), 29
get_params() (pyod.models.base.BaseDetector method), 18
get_params() (pyod.models.cblof.CBLOF method), 45
get_params() (pyod.models.cof.COF method), 48
get_params() (pyod.models.copod.COPOD method), 53
get_params() (pyod.models.deep_svdd.DeepSVDD method), 57
get_params() (pyod.models.ecod.ECOD method), 61
get_params() (pyod.models.feature_bagging.FeatureBagging method), 65
get_params() (pyod.models.hbos.HBOS method), 68
get_params() (pyod.models.iforest.IForest method), 72
get_params() (pyod.models.knn.KNN method), 76
get_params() (pyod.models.lmdd.LMDD method), 79
get_params() (pyod.models.loda.LODA method), 82
get_params() (pyod.models.lscp.LSCP method), 93
get_params() (pyod.models.mad.MAD method), 95
get_params() (pyod.models.mcd.MCD method), 99
get_params() (pyod.models.mo_gaal.MO_GAAL method), 102
get_params() (pyod.models.ocsvm.OCSVM method), 106
get_params() (pyod.models.pca.PCA method), 111
get_params() (pyod.models.rod.ROD method), 114
get_params() (pyod.models.so_gaal.SO_GAAL method), 122
get_params() (pyod.models.sod.SOD method), 119
get_params() (pyod.models.sos.SOS method), 126
get_params() (pyod.models.suod.SUOD method), 130
get_params() (pyod.models.vae.VAE method), 134
get_params() (pyod.models.xgbod.XGBOD method), 138
get_submodule() (pyod.models.auto_encoder_torch.AutoEncoder method), 34

H
half() (pyod.models.auto_encoder_torch.inner_autoencoder method), 35
 HBOS (class in pyod.models.hbos), 66
hist_(pyod.models.hbos.HBOS attribute), 66
history_ (pyod.models.auto_encoder.AutoEncoder attribute), 24
history_ (pyod.models.auto_encoder_torch.AutoEncoder attribute), 28
history_ (pyod.models.deep_svdd.DeepSVDD attribute), 56
history_ (pyod.models.vae.VAE attribute), 133

I
IForest (class in pyod.models.iforest), 69
inner_autoencoder (class in pyod.models.auto_encoder_torch), 31
intercept_ (pyod.models.ocsvm.OCSVM attribute), 105
invert_order() (in module pyod.utils.utility), 147

K
KNN (class in pyod.models.knn), 73

L
labels_ (pyod.models.abod.ABOD attribute), 20
labels_ (pyod.models.auto_encoder.AutoEncoder attribute), 24
labels_ (pyod.models.auto_encoder_torch.AutoEncoder attribute), 28
labels_ (pyod.models.base.BaseDetector attribute), 17
labels_ (pyod.models.cblof.CBLOF attribute), 44
labels_ (pyod.models.cof.COF attribute), 47
labels_ (pyod.models.copod.COPOD attribute), 52
labels_ (pyod.models.deep_svdd.DeepSVDD attribute), 56
labels_ (pyod.models.ecod.ECOD attribute), 59
labels_ (pyod.models.feature_bagging.FeatureBagging attribute), 63
labels_ (pyod.models.hbos.HBOS attribute), 67
labels_ (pyod.models.iforest.IForest attribute), 71
labels_ (pyod.models.knn.KNN attribute), 75
labels_ (pyod.models.lmdd.LMDD attribute), 78
labels_ (pyod.models.loci.LOCI attribute), 88
labels_ (pyod.models.loda.LODA attribute), 81
labels_ (pyod.models.lof.LOF attribute), 85
labels_ (pyod.models.lscp.LSCP attribute), 91
labels_ (pyod.models.mad.MAD attribute), 94
labels_ (pyod.models.mcd.MCD attribute), 98
labels_ (pyod.models.mo_gaal.MO_GAAL attribute), 101
labels_ (pyod.models.ocsvm.OCSVM attribute), 105
labels_ (pyod.models.pca.PCA attribute), 110
labels_ (pyod.models.rod.ROD attribute), 113
labels_ (pyod.models.so_gaal.SO_GAAL attribute), 121
labels_ (pyod.models.sod.SOD attribute), 118
labels_ (pyod.models.sos.SOS attribute), 124
labels_ (pyod.models.suod.SUOD attribute), 129
labels_ (pyod.models.vae.VAE attribute), 133
labels_ (pyod.models.xgbod.XGBOD attribute), 137
large_cluster_labels_ (pyod.models.cblof.CBLOF attribute), 44
LMDD (class in pyod.models.lmdd), 77
load_state_dict() (pyod.models.auto_encoder_torch.inner_autoencoder method), 35

LMD (class in pyod.models.lscp), 80
LOF (class in pyod.models.lof), 83
LSCP (class in pyod.models.lscp), 90

M
MAD (class in pyod.models.mad), 94
mad() (in module pyod.models.rod), 116
majority_vote() (in module pyod.models.combination), 50
max_samples_ (pyod.models.iforest.IForest attribute), 70
maximization() (in module pyod.models.combination), 51
MCD (class in pyod.models.mcd), 97

Index 165
predict() (pyod.models.mad.MAD method), 96
predict() (pyod.models.mcd.MCD method), 99
predict() (pyod.models.mo_gaal.MO_GAAL method), 102
predict() (pyod.models.ocsvm.OCSVM method), 106
predict() (pyod.models.pca.PCA method), 111
predict() (pyod.models.rod.ROD method), 114
predict() (pyod.models.so_gaal.SO_GAAL method), 123
predict() (pyod.models.sod.SOD method), 120
predict() (pyod.models.sos.SOS method), 126
predict() (pyod.models.suod.SUOD method), 130
predict() (pyod.models.vae.VAE method), 134
predict_confidence() (pyod.models.abod.ABOD method), 22
predict_confidence() (pyod.models.auto_encoder.AutoEncoder method), 26
predict_confidence() (pyod.models.auto_encoder_torch.AutoEncoder method), 30
predict_confidence() (pyod.models.base.BaseDetector method), 19
predict_confidence() (pyod.models.cblof.CBLOF method), 46
predict_confidence() (pyod.models.cof.COF method), 49
predict_confidence() (pyod.models.copod.COPOD method), 54
predict_confidence() (pyod.models.deep_svdd.DeepSVDD method), 58
predict_confidence() (pyod.models.ecod.ECOD method), 61
predict_confidence() (pyod.models.feature_bagging.FeatureBagging method), 65
predict_confidence() (pyod.models hbos.HBOS method), 68
predict_confidence() (pyod.models.iforest.IForest method), 72
predict_confidence() (pyod.models.knn.KNN method), 76
predict_confidence() (pyod.models.lmdd.LMDD method), 79
predict_confidence() (pyod.models.loci.LOCI method), 89
predict_confidence() (pyod.models.loda.LODA method), 82
predict_confidence() (pyod.models.lof.LOF method), 86
predict_confidence() (pyod.models.lscp.LSCP method), 93
predict_confidence() (pyod.models.mad.MAD method), 96
predict_confidence() (pyod.models.mcd.MCD method), 100
predict_confidence() (pyod.models.mo_gaal.MO_GAAL method), 103
predict_confidence() (pyod.models.ocsvm.OCSVM method), 107
predict_confidence() (pyod.models.pca.PCA method), 112
predict_confidence() (pyod.models.rod.ROD method), 115
predict_confidence() (pyod.models.so_gaal.SO_GAAL method), 123
predict_confidence() (pyod.models.sod.SOD method), 120
predict_confidence() (pyod.models.sos.SOS method), 126
predict_confidence() (pyod.models.suod.SUOD method), 130
predict_confidence() (pyod.models.vae.VAE method), 134
predict_confidence() (pyod.models.xgbod.XGBOD method), 138
predict_proba() (pyod.models.abod.ABOD method), 22
predict_proba() (pyod.models.auto_encoder.AutoEncoder method), 26
predict_proba() (pyod.models.auto_encoder_torch.AutoEncoder method), 30
predict_proba() (pyod.models.base.BaseDetector method), 19
predict_proba() (pyod.models.cblof.CBLOF method), 46
predict_proba() (pyod.models.cof.COF method), 49
predict_proba() (pyod.models.copod.COPOD method), 54
predict_proba() (pyod.models.deep_svdd.DeepSVDD method), 58
predict_proba() (pyod.models.ecod.ECOD method), 61
predict_proba() (pyod.models.feature_bagging.FeatureBagging method), 65
predict_proba() (pyod.models hbos.HBOS method), 68
predict_proba() (pyod.models.iforest.IForest method), 72
predict_proba() (pyod.models.knn.KNN method), 76
predict_proba() (pyod.models.lmdd.LMDD method), 79
predict_proba() (pyod.models.loci.LOCI method), 90

Index
Index
threshold_ (pyod.models.feature_bagging.FeatureBagging attribute), 63
threshold_ (pyod.models.hbos.HBOS attribute), 67
threshold_ (pyod.models.iforest.IForest attribute), 71
threshold_ (pyod.models.knn.KNN attribute), 74
threshold_ (pyod.models.lmdd.LMDD attribute), 78
threshold_ (pyod.models.loci.LOCI attribute), 87
threshold_ (pyod.models.loda.LODA attribute), 81
threshold_ (pyod.models.lof_LOF attribute), 84
threshold_ (pyod.models.lscp.LSCP attribute), 91
threshold_ (pyod.models.mad.MAD attribute), 94
threshold_ (pyod.models.mcd.MCD attribute), 98
threshold_ (pyod.models.mo_gaal.MO_GAAL attribute), 101
threshold_ (pyod.models.ocsvm.OCSVM attribute), 105
threshold_ (pyod.models.pca.PCA attribute), 110
threshold_ (pyod.models.rod.ROD attribute), 113
threshold_ (pyod.models.so_gaal.SO_GAAL attribute), 121
threshold_ (pyod.models.sod.SOD attribute), 118
threshold_ (pyod.models.sos.SOS attribute), 124
threshold_ (pyod.models.suod.SUOD attribute), 128
to() (pyod.models.auto_encoder_torch.inner_autoencoder method), 40
to_empty() (pyod.models.auto_encoder_torch.inner_autoencoder method), 41
train() (pyod.models.auto_encoder_torch.inner_autoencoder method), 41
training (pyod.models.auto_encoder_torch.inner_autoencoder attribute), 42
type() (pyod.models.auto_encoder_torch.inner_autoencoder method), 42

V

VAE (class in pyod.models.vae), 131
vae_loss() (pyod.models.vae.VAE method), 135
visualize() (in module pyod.utils.example), 143

W

wppearsonr() (in module pyod.utils.stat_models), 144

X

XGBOD (class in pyod.models.xgbod), 136
xpu() (pyod.models.auto_encoder_torch.inner_autoencoder method), 42

Z

zero_grad() (pyod.models.auto_encoder_torch.inner_autoencoder method), 42