PyOD: A Python Toolbox for Scalable Outlier Detection

Yue Zhao  
Carnegie Mellon University*
Pittsburgh, PA 15213, USA

Zain Nasrullah  
University of Toronto
Toronto, ON M5S 2E4, Canada

Zheng Li  
Northeastern University Toronto
Toronto, ON M5X 1E2, Canada

Editor: Alexandre Gramfort

Abstract

PyOD is an open-source Python toolbox for performing scalable outlier detection on multivariate data. Uniquely, it provides access to a wide range of outlier detection algorithms, including established outlier ensembles and more recent neural network-based approaches, under a single, well-documented API designed for use by both practitioners and researchers. With robustness and scalability in mind, best practices such as unit testing, continuous integration, code coverage, maintainability checks, interactive examples and parallelization are emphasized as core components in the toolbox’s development. PyOD is compatible with both Python 2 and 3 and can be installed through Python Package Index (PyPI) or https://github.com/yzhao062/pyod.

Keywords: anomaly detection, outlier detection, outlier ensembles, neural networks, machine learning, data mining, Python

1. Introduction

Outlier detection, also known as anomaly detection, refers to the identification of rare items, events or observations which differ from the general distribution of a population. Since the ground truth is often absent in such tasks, dedicated outlier detection algorithms are extremely valuable in fields which process large amounts of unlabelled data and require a means to reliably perform pattern recognition (Akoglu et al., 2012). Industry applications include fraud detection in finance (Ahmed et al., 2016), fault diagnosis in mechanics (Shin et al., 2005), intrusion detection in network security (Garcia-Teodoro et al., 2009) and pathology detection in medical imaging (Baur et al., 2018).

To help approach these problems, established outlier detection packages exist in various programming languages such as ELKI Data Mining (Achtert et al., 2010) and RapidMiner (Hofmann and Klinkenberg, 2013) in Java and outliers (Komsta and Komsta, 2011) in R. However, Python, one of the most important languages in machine learning, still lacks a dedicated toolkit for outlier detection. Existing implementations either stand as single

* Work initialized while at University of Toronto.

©2019 Yue Zhao, Zain Nasrullah and Zheng Li.
License: CC-BY 4.0, see https://creativecommons.org/licenses/by/4.0/ Attribution requirements are provided at http://jmlr.org/papers/v20/19-011.html.
Table 1: Select outlier detection models in PyOD

<table>
<thead>
<tr>
<th>Method</th>
<th>Category</th>
<th>JIT Enabled</th>
<th>Multi-core</th>
</tr>
</thead>
<tbody>
<tr>
<td>LOF (Breunig et al., 2000)</td>
<td>Proximity</td>
<td>No</td>
<td>Yes</td>
</tr>
<tr>
<td>kNN (Ramaswamy et al., 2000)</td>
<td>Proximity</td>
<td>No</td>
<td>Yes</td>
</tr>
<tr>
<td>AvgkNN (Angiulli and Pizzuti, 2002)</td>
<td>Proximity</td>
<td>No</td>
<td>Yes</td>
</tr>
<tr>
<td>CBLOF (He et al., 2003)</td>
<td>Proximity</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>OCSVM (Schölkopf et al., 2001)</td>
<td>Linear Model</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>LOCI (Papadimitriou et al., 2003)</td>
<td>Proximity</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>PCA (Shyu et al., 2003)</td>
<td>Linear Model</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>MCD (Hardin and Rocke, 2004)</td>
<td>Linear Model</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>Feature Bagging (Lazarevic and Kumar, 2005)</td>
<td>Ensembling</td>
<td>No</td>
<td>Yes</td>
</tr>
<tr>
<td>ABOD (Kriegel et al., 2008)</td>
<td>Proximity</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>Isolation Forest (Liu et al., 2008)</td>
<td>Ensembling</td>
<td>No</td>
<td>Yes</td>
</tr>
<tr>
<td>HBOS (Goldstein and Dengel, 2012)</td>
<td>Proximity</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>SOS (Janssens et al., 2012)</td>
<td>Proximity</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>AutoEncoder (Sakurada and Yairi, 2014)</td>
<td>Neural Net</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>AOM (Aggarwal and Sathe, 2015)</td>
<td>Ensembling</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>MOA (Aggarwal and Sathe, 2015)</td>
<td>Ensembling</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>SO-GAAL (Liu et al., 2019)</td>
<td>Neural Net</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>MO-GAAL (Liu et al., 2019)</td>
<td>Neural Net</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>XGBOD (Zhao and Hryniewicki, 2018b)</td>
<td>Ensembling</td>
<td>No</td>
<td>Yes</td>
</tr>
<tr>
<td>LSCP (Zhao et al., 2019)</td>
<td>Ensembling</td>
<td>No</td>
<td>No</td>
</tr>
</tbody>
</table>

algorithm tools like PyNomaly (Constantinou, 2018) or exist as part of a general-purpose framework like scikit-learn (Pedregosa et al., 2011) which does not cater specifically to anomaly detection. To fill this gap, we propose and implement PyOD—a comprehensive Python toolbox for scalable outlier detection.

Compared to existing libraries, PyOD has six distinct advantages. Firstly, it contains more than 20 algorithms which cover both classical techniques such as local outlier factor and recent neural network architectures such as autoencoders or adversarial models. Secondly, PyOD implements combination methods for merging the results of multiple detectors and outlier ensembles which are an emerging set of models. Thirdly, PyOD includes a unified API, detailed documentation and interactive examples across all algorithms for clarity and ease of use. Fourthly, all models are covered by unit testing with cross platform continuous integration, code coverage and code maintainability checks. Fifthly, optimization instruments are employed when possible: just-in-time (JIT) compilation and parallelization are enabled in select models for scalable outlier detection. Lastly, PyOD is compatible with both Python 2 and 3 across major operating systems (Windows, Linux and MacOS). Popular detection algorithms implemented in PyOD (version 0.7.0) are summarized in Table 1.
2. Project Focus

**Build robustness.** Continuous integration tools (*Travis CI, AppVeyor and CircleCI*) are leveraged to conduct automated testing under various versions of Python and operating systems. Tests are executed daily, when commits are made to the development and master branches, or when a pull request is opened.

**Quality assurance.** The project follows the *PEP8* standard; maintainability is ensured by *CodeClimate*, an automated code review and quality assurance tool. Additionally, code blocks with high cognitive complexity are actively refactored and a standard set of unit tests exist to ensure 95% overall code coverage. These design choices enhance collaboration and this standard is enforced on all pull requests for consistency.

**Community-based development.** *PyOD*’s code repository is hosted on GitHub\(^1\) to facilitate collaboration. At the time of this writing, eight contributors have helped develop the code base and others have contributed in the form of bug reports and feature requests.

**Documentation and examples.** Comprehensive documentation is developed using *sphinx* and *numpydoc* and rendered using *Read the Docs*\(^2\). It includes detailed API references, an installation guide, code examples and algorithm benchmarks. An interactive Jupyter notebook is also hosted on *Binder* allowing others to experiment prior to installation.

**Project relevance.** *PyOD* has been used in various academic and commercial projects (*Zhao and Hryniewicki, 2018a; Ramakrishnan et al., 2019; Krishnan and Wu, 2019*). The GitHub repository receives more than 10,000 monthly views and its PyPI downloads exceed 6,000 per month.

3. Library Design and Implementation

*PyOD* is compatible with both Python 2 and 3 using *six*; it relies on *numpy, scipy* and *scikit-learn* as well. Neural networks such as autoencoders and SO_GAAL additionally require *Keras*. To enhance model scalability, select algorithms (Table 1) are optimized with JIT using *numba*. Parallelization for multi-core execution is also available for a set of algorithms using *joblib*. Inspired by *scikit-learn*’s API design (Buitinck et al., 2013), all implemented outlier detection algorithms inherit from a base class with the same interface: (i) *fit* processes the train data and computes the necessary statistics; (ii) *decision_function* generates raw outlier scores for unseen data after the model is fitted; (iii) *predict* returns a binary class label corresponding to each input sample instead of the raw outlier score and (iv) *predict_proba* offers the result as a probability using either normalization or Unification (Kriegel et al., 2011). Within this framework, new models are easy to implement by taking advantage of inheritance and polymorphism. Base methods can then be overridden as necessary.

Once a detector has been fitted on a training set, the corresponding train outlier scores and binary labels are accessible using its *decision_scores* and *labels* attributes. Once fitted, the model’s *predict, decision_function and predict_proba* methods may be called for use on new data. An example showcasing the ease of use of this API is shown in Code Snippet 1 with an angle-based outlier detector (ABOD).

---

1. https://github.com/yzhao062/pyod
2. https://pyod.readthedocs.io
>>> from pyod.models.abod import ABOD
>>> from pyod.utils.data import generate_data
>>> from pyod.utils.data import evaluate_print
>>> from pyod.utils.example import visualize

>>> X_train, y_train, X_test, y_test = generate_data(
...     n_train=200, n_test=100, n_features=2)

>>> clf = ABOD(method="fast")  # initialize detector
>>> clf.fit(X_train)

>>> y_test_pred = clf.predict(X_test)  # binary labels
>>> y_test_scores = clf.decision_function(X_test)  # raw outlier scores
>>> y_test_proba = clf.predict_proba(X_test)  # outlier probability

>>> evaluate_print("ABOD", y_test, y_test_scores)  # performance evaluation
ABOD Performance; ROC: 0.934; Precision at n: 0.902

>>> visualize(y_test, y_test_scores)  # prediction visualization

Code Snippet 1: Demo of PyOD API with the ABOD detector

In addition to the outlier detection algorithms, a set of helper and utility functions (generate_data, evaluate_print and visualize) are included in the library for quick model exploration and evaluation. The two-dimensional artificial data used in the example is created by generate_data which generates inliers from a Gaussian distribution and outliers from a uniform distribution. An example of using visualize is shown in Figure 1.

Figure 1: Demonstration of using PyOD in visualizing prediction result
4. Conclusion and Future Plans

This paper presents PyOD, a comprehensive toolbox built in Python for scalable outlier detection. It includes more than 20 classical and emerging detection algorithms and is being used in both academic and commercial projects. As avenues for future work, we plan to enhance the toolbox by implementing models that work well with time series and geospatial data, improving computational efficiency through distributed computing and addressing engineering challenges such as handling sparse matrices or memory limitations.

Acknowledgments

We thank the editor and anonymous reviewers for their constructive comments. This work was partly supported by Mitacs through the Mitacs Accelerate program.

References


Yezheng Liu, Zhe Li, Chong Zhou, Yuanchun Jiang, Jiannshun Sun, Meng Wang, and Xiangnan He. Generative adversarial active learning for unsupervised outlier detection. *IEEE Transactions on Knowledge and Data Engineering (TKDE)*, 2019.


Yue Zhao and Maciej K Hryniewicki. XGBOD: improving supervised outlier detection with unsupervised representation learning. In *International Joint Conference on Neural Networks (IJCNN)*, Rio de Janeiro, Brazil, July 2018b. IEEE.