PyGOD: A Python Library for Graph Outlier Detection

Kay Liu1∗, Yingtong Dou1,2∗, Xueying Ding3, Xiyang Hu4, Ruitong Zhang5, Hao Peng6,7, Lichao Sun8, Philip S. Yu1

1 University of Illinois Chicago, 2 Visa Research, 3 Carnegie Mellon University, 4 Arizona State University, 5 Alibaba Group, 6 Kunming University of Science and Technology, 7 Shantou University, 8 Lehigh University

Abstract

PyGOD is an open-source Python library for detecting outliers in graph data. As the first comprehensive library of its kind, PyGOD supports a wide array of leading graph-based methods for outlier detection under an easy-to-use, well-documented API designed for use by both researchers and practitioners. PyGOD provides modularized components of the different detectors implemented so that users can easily customize each detector for their purposes. To ease the construction of detection workflows, PyGOD offers numerous commonly used utility functions. To scale computation to large graphs, PyGOD supports functionalities for deep models such as sampling and mini-batch processing. PyGOD uses best practices in fostering code reliability and maintainability, including unit testing, continuous integration, and code coverage. To facilitate accessibility, PyGOD is released under a BSD 2-Clause license at https://pygod.org and at the Python Package Index (PyPI).

Keywords: outlier detection, anomaly detection, graph learning, graph neural networks

1. Introduction

Outlier detection (OD), also known as anomaly detection, is a key machine learning task to identify deviant samples from the general data distribution (Aggarwal, 2017; Li et al., 2022). With the increasing importance of graph data in both research and real-world applications (Ding et al., 2021b; Huang et al., 2021; Fu et al., 2021; Zhou et al., 2021; Xu et al., 2022), detecting outliers with graph-based methods, particularly graph neural networks (GNNs), has recently garnered considerable attention (Ma et al., 2021; Ding et al., 2019b, 2021a; Liu et al., 2022) with many applications such as detecting suspicious activities in social networks (Sun et al., 2022; Dou et al., 2020) and security systems (Cai et al., 2021).

Although there is a long list of libraries for detecting outliers in tabular and time-series data in multiple programming languages (e.g., PyOD (Zhao et al., 2019), SUOD (Zhao...))
Table 1: Implemented graph outlier detectors in PyGOD v1.1.0.

et al., 2021a), PyODDS (Li et al., 2020), ELKI (Achtert et al., 2010), OutlierDetection.jl (Muhr et al., 2022), PyTOD (Zhao et al., 2022), TODS (Lai et al., 2021), Telemanom (Hundman et al., 2018)), there is no specialized library for graph outlier detection.

To bridge this gap, we design the first comprehensive Python Graph Outlier Detection library called PyGOD, with a couple of key technical advancements and contributions. First, it covers a wide array of algorithms with various backbones, including clustering, matrix factorization (MF), generative adversarial networks (GANs), autoencoders (AEs), GNNs, and self-supervised learning (SSL). PyGOD already supports more than fifteen representative algorithms as shown in Table 1. Second, PyGOD implements these detection models with a unified API so that the user only needs to prepare the data in a predefined graph format, at which point all outlier detectors in PyGOD can process the data. Third, PyGOD offers flexible and modularized components of the different outlier detectors implemented, enabling users to customize these detectors according to individual needs. Moreover, PyGOD provides many commonly used utility functions to ease the construction of graph outlier detection workflows. Fourth, PyGOD can scale outlier detection to large graphs using sampling and mini-batch processing. With a focus on code clarity and quality, we provide comprehensive API documentation and examples. Additionally, we provide unit tests with cross-platform continuous integration along with code coverage and maintainability checks.

2. Library Design and Implementation

Dependency. PyGOD builds for Python 3.8+ and depends on the popular PyTorch (Paszke et al., 2019) and PyTorch Geometric (PyG) (Fey and Lenssen, 2019) packages for graph learning on both CPUs and GPUs. Additionally, PyGOD uses NumPy (Harris et al., 2020), SciPy (Virtanen et al., 2020), and scikit-learn (Pedregosa et al., 2011), and NetworkX (Hagberg et al., 2008) for data manipulation.

API Design. As shown in Figure 1, inspired by the API design of scikit-learn (Buitinck et al., 2013) and PyOD (Zhao et al., 2019), all detection algorithms in PyGOD inherit from
Figure 1: Demonstration of PyGOD’s unified API design.

a base class with the same API interface: (i) `fit` trains the detector, gets the outlier scores (higher means more outlying) and outlier prediction on the input data, and generates necessary statistics for prediction (in the inductive setting); (ii) `predict` leverages the trained model to predict on the input data in the inductive setting (no input data are required in the transductive setting). `predict` returns a binary prediction (0 for normal samples and 1 for outliers), a raw outlier score, a probability of a sample being an outlier (using the method by Kriegel et al. (2011)), and a confidence score (Perini et al., 2020) based on users’ needs. The usage of the above APIs is demonstrated in Code Demo 1.

```python
1  from pygod.utils import load_data  # import data function
2  data = load_data("inj_cora")  # load built-in dataset
3
4  from pygod.detector import DOMINANT  # import the detector
5  model = DOMINANT(num_layers=4)  # initialize the detector
6  model.fit(data)  # train with data
7
8  pred, score = model.predict(data,  # predict labels by default
9      return_score=True)  # and raw outlier scores
10
11  from pygod.metric import eval_roc_auc, eval_f1  # import the metric
12  eval_f1(data.y.bool(), pred)  # evaluate by F1
13  eval_roc_auc(data.y.bool(), score)  # evaluate by AUC
```

Code Demo 1: Using DOMINANT (Ding et al., 2019a) on Cora (Morris et al., 2020).

**Streamlined Graph Learning with PyG.** We choose to develop PyGOD on top of the popular PyG library for multiple reasons. First, this reduces the complexity of processing graph data for users. That is, PyGOD only requires the input data to be in the standard graph data format in PyG¹. Second, most of the detectors use GNNs (Kipf and Welling, 2017) as their backbone (see Table 1), where PyG already provides an optimized implementation. Third, PyG is the most popular GNN library with advanced functions like graph sampling and distributed training. Under the PyG framework, we implement mini-batch

---
processing and/or sampling for selected models to accommodate learning with large graphs as shown in Table 1.

**Modularized components and helpful utility functions.** To minimize code redundancy and improve reusability, PyGOD employs modularization in the implementation of deep detectors, dividing different components into `nn.conv`, `nn.encoder`, `nn.decoder`, and `nn.functional`. Additionally, a set of helpful utility functions is designed to facilitate graph outlier detection. In terms of tasks, PyGOD includes `utils.to_edge_score` and `utils.to_graph_score` to enable the adaptation of any node level model to edge level and (sub)graph level outlier detection. In terms of evaluation, PyGOD offers common metrics for graph outlier detection in the `metric` module. In terms of data, PyGOD provides built-in example data sets through `utils.load_data`. Moreover, it offers outlier generator methods in the `generator` module for injecting both contextual and structural outliers (Ding et al., 2019a). This serves as a solution for model evaluation and benchmarking. For more details, please refer to PyGOD documentation.

3. Library Robustness and Accessibility

**Robustness and Quality.** While building PyGOD, we follow the best practices of system design and software development. First, we leverage the continuous integration by GitHub Actions to automate the testing process under various Python versions and operating systems. In addition to the scheduled daily test, both commits and pull requests trigger the unit testing. Notably, we enforce all code to have over 99% coverage. By following the PEP8 standard, we enforce a consistent coding style and naming convention, which facilitates community collaboration and code readability.

**Accessibility and Community.** PyGOD comes with detailed API documentation rendered by Read the Docs. The documentation includes an installation guide as well as interactive examples in Jupyter notebooks. To facilitate community contribution, the project is hosted on GitHub with a friendly contribution guide and issue reporting mechanism. At the time of publishing, PyGOD has been widely used in numerous real-world applications including Twitter bot detection (Feng et al., 2022) and financial fraud detection (Huang et al., 2022), with more than 1,200 GitHub stars and 20,000 PyPI downloads.

4. Conclusion and Future Plans

In this paper, we present the first comprehensive library for graph outlier detection, called PyGOD. PyGOD supports a wide range of detection algorithms with a unified API, rich documentation, and robust code design. These features make it valuable for both academic research and industry applications. The development plan of PyGOD will focus on multiple aspects: (i) enabling detectors to acquire domain knowledge by incorporating different amounts of supervision signals; (ii) optimizing its scalability with the latest advancement in graphs (Jia et al., 2020); and (iii) incorporating automated machine learning to enable intelligent model selection and hyperparameter tuning (Zhao et al., 2021b).

---

2. Documentation: https://docs.pygod.org/
3. Continuous integration by GitHub Actions: https://github.com/pygod-team/pygod/actions
Acknowledgments

The authors who are affiliated with the University of Illinois Chicago are supported in part by NSF under grant III-2106758 and POSE-2346158. Philip S. Yu and Hao Peng are the corresponding authors.

References


L. Sun, Y. Dou, C. Yang, K. Zhang, J. Wang, S. Y. Philip, L. He, and B. Li. Adversarial attack and defense on graph data: a survey. *IEEE Transactions on Knowledge and Data Engineering (TKDE)*, 2022.


Z. Xu, X. Huang, Y. Zhao, Y. Dong, and J. Li. Contrastive attributed network anomaly detection with data augmentation. In *Pacific-Asia Conference on Knowledge Discovery and Data Mining (PAKDD)*, 2022.


Y. Zhao, G. H. Chen, and Z. Jia. TOD: GPU-accelerated outlier detection via tensor operations. In International Conference on Very Large Data Bases (VLDB), volume 16, 2022.