# PyGOD Documentation

***Release 1.1.0***

**PyGOD Team**

**Feb 04, 2024**

# GETTING STARTED

PyGOD is a **Python library** for **graph outlier detection** (anomaly detection). This exciting yet challenging field has many key applications, e.g., detecting suspicious activities in social networks [DLS+20] and security systems [CCL+21].

PyGOD includes **10+** graph outlier detection algorithms. For consistency and accessibility, PyGOD is developed on top of PyTorch Geometric (PyG) and PyTorch, and follows the API design of PyOD. See examples below for detecting outliers with PyGOD in 5 lines!

**PyGOD is featured for**:

- **Unified APIs, detailed documentation, and interactive examples** across various graph-based algorithms.
- **Comprehensive coverage** of 10+ graph outlier detectors.
- **Full support of detections at multiple levels**, such as node-, edge-, and graph-level tasks.
- **Scalable design for processing large graphs** via mini-batch and sampling.
- **Streamline data processing with PyG**–fully compatible with PyG data objects.

**Outlier Detection Using PyGOD with 5 Lines of Code**:

```
# train a dominant detector
from pygod.detector import DOMINANT

model = DOMINANT(num_layers=4, epoch=20)  # hyperparameters can be set here
model.fit(train_data)  # input data is a PyG data object

# get outlier scores on the training data (transductive setting)
score = model.decision_score_

# predict labels and scores on the testing data (inductive setting)
pred, score = model.predict(test_data, return_score=True)
```

| Abbr | Year | Backbone | Sampling | Class |
|---|---|---|---|---|
| SCAN | 2007 | Clustering | No | *pygod.detector.SCAN* |
| GAE | 2016 | GNN+AE | Yes | *pygod.detector.GAE* |
| Radar | 2017 | MF | No | *pygod.detector.Radar* |
| ANOMALOUS | 2018 | MF | No | *pygod.detector.ANOMALOUS* |
| ONE | 2019 | MF | No | *pygod.detector.ONE* |
| DOMINANT | 2019 | GNN+AE | Yes | *pygod.detector.DOMINANT* |
| DONE | 2020 | MLP+AE | Yes | *pygod.detector.DONE* |
| AdONE | 2020 | MLP+AE | Yes | *pygod.detector.AdONE* |
| AnomalyDAE | 2020 | GNN+AE | Yes | *pygod.detector.AnomalyDAE* |
| GAAN | 2020 | GAN | Yes | *pygod.detector.GAAN* |
| DMGD | 2020 | GNN+AE | Yes | *pygod.detector.DMGD* |
| OCGNN | 2021 | GNN | Yes | *pygod.detector.OCGNN* |
| CoLA | 2021 | GNN+AE+SSL | Yes | *pygod.detector.CoLA* |
| GUIDE | 2021 | GNN+AE | Yes | *pygod.detector.GUIDE* |
| CONAD | 2022 | GNN+AE+SSL | Yes | *pygod.detector.CONAD* |
| GADNR | 2024 | GNN+AE | Yes | *pygod.detector.GADNR* |

CHAPTER

ONE

# INSTALLATION

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

```
pip install pygod            # normal install
pip install --upgrade pygod  # or update if needed
```

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

```
git clone https://github.com/pygod-team/pygod.git
cd pygod
pip install .
```

**Required Dependencies**:

- python>=3.8
- numpy>=1.24.3
- scikit-learn>=1.2.2
- scipy>=1.10.1
- networkx>=3.1

**Note on PyG and PyTorch Installation**: PyGOD depends on torch and torch_geometric (including its optional dependencies). To streamline the installation, PyGOD does **NOT** install these libraries for you. Please install them from the above links for running PyGOD:

- torch>=2.0.0
- torch_geometric>=2.3.0

CHAPTER

TWO

# A BLITZ INTRODUCTION

## 2.1 Detector Example

In this tutorial, you will learn the basic workflow of PyGOD with an example of DOMINANT. This tutorial assumes that you have basic familiarity with PyTorch and PyTorch Geometric (PyG).

(Time estimate: 5 minutes)

### 2.1.1 Data Loading

PyGOD use `torch_geometric.data.Data` to handle the data. Here, we use Cora, a PyG built-in dataset, as an example. To load your own dataset into PyGOD, you can refer to creating your own datasets tutorial in PyG.

```
import torch_geometric.transforms as T
from torch_geometric.datasets import Planetoid

data = Planetoid('./data/Cora', 'Cora', transform=T.NormalizeFeatures())[0]
```

```
Downloading https://github.com/kimiyoung/planetoid/raw/master/data/ind.cora.x
Downloading https://github.com/kimiyoung/planetoid/raw/master/data/ind.cora.tx
Downloading https://github.com/kimiyoung/planetoid/raw/master/data/ind.cora.allx
Downloading https://github.com/kimiyoung/planetoid/raw/master/data/ind.cora.y
Downloading https://github.com/kimiyoung/planetoid/raw/master/data/ind.cora.ty
Downloading https://github.com/kimiyoung/planetoid/raw/master/data/ind.cora.ally
Downloading https://github.com/kimiyoung/planetoid/raw/master/data/ind.cora.graph
Downloading https://github.com/kimiyoung/planetoid/raw/master/data/ind.cora.test.index
Processing...
Done!
```

Because there is no ground truth label of outliers in Cora, we follow the method used by DOMINANT to inject 100 contextual outliers and 100 structure outliers into the graph. Note: If your dataset already contains the outliers you want to detect, you don't have to inject more outliers.

```
import torch
from pygod.generator import gen_contextual_outlier, gen_structural_outlier

data, ya = gen_contextual_outlier(data, n=100, k=50)
data, ys = gen_structural_outlier(data, m=10, n=10)
data.y = torch.logical_or(ys, ya).long()
```

We also provide various type of built-in datasets. You can load them by passing the name of the dataset to `load_data` function. See data repository for more details.

```
from pygod.utils import load_data

data = load_data('inj_cora')
data.y = data.y.bool()
```

### 2.1.2 Initialization

You can use any detector by simply initializing without passing any arguments. Default hyperparameters are ready for you. Of course, you can also customize the parameters by passing arguments. Here, we use `pygod.detector.DOMINANT` as an example.

```
from pygod.detector import DOMINANT

detector = DOMINANT(hid_dim=64, num_layers=4, epoch=100)
```

### 2.1.3 Training

To train the detector with the loaded data, simply feed the `torch_geometric.data.Data` object into the detector via `fit`.

```
detector.fit(data)
```

```
DOMINANT(act=<function relu at 0x7fb2fbf98e50>,
         backbone=<class 'torch_geometric.nn.models.basic_gnn.GCN'>,
         batch_size=2708, compile_model=False, contamination=0.1,
         dropout=0.0, epoch=100, gpu=None, hid_dim=64, lr=0.004,
         num_layers=4, num_neigh=[-1, -1, -1, -1], save_emb=False,
         sigmoid_s=False, verbose=0, weight=0.5, weight_decay=0.0)
```

### 2.1.4 Inference

After training, the detector is ready to use. You can use the detector to predict the labels, raw outlier scores, probability of the outlierness, and prediction confidence. Here, we use the loaded data as an example.

```
pred, score, prob, conf = detector.predict(data,
                                           return_pred=True,
                                           return_score=True,
                                           return_prob=True,
                                           return_conf=True)
print('Labels:')
print(pred)

print('Raw scores:')
print(score)

print('Probability:')
print(prob)
```

(continues on next page)

(continued from previous page)

```
print('Confidence:')
print(conf)
```

```
Labels:
tensor([0, 0, 0,  ..., 0, 0, 0])
Raw scores:
tensor([1.0303, 0.9657, 1.2201,  ..., 0.6121, 1.1309, 1.1349])
Probability:
tensor([0.0743, 0.0634, 0.1063,  ..., 0.0038, 0.0913, 0.0919])
Confidence:
tensor([1., 1., 1.,  ..., 1., 1., 1.])
```

### 2.1.5 Evaluation

To evaluate the performance outlier detector with AUC score, you can:

```
from pygod.metric import eval_roc_auc

auc_score = eval_roc_auc(data.y, score)
print('AUC Score:', auc_score)
```

```
AUC Score: 0.7669429876501437
```

**Total running time of the script:** (0 minutes 44.496 seconds)

## 2.2 Score Conversion

Currently, the majority of outlier detectors are on node level. However, in some real-world applications, we may interest in the outlier edges or outlier graphs. In this tutorial, we introduce score converters provided in `pygod.utils` module, including `to_edge_score` and `to_graph_score`.

(Time estimate: 3 minutes)

### 2.2.1 Data Loading

We first load the data from PyGOD with `load_data` function.

```
from pygod.utils import load_data

data = load_data('weibo')
print(data)
```

```
Data(x=[8405, 400], edge_index=[2, 407963], y=[8405], train_mask=[8405], test_
↪mask=[8405], val_mask=[8405])
```

### 2.2.2 Detector Training

Initialize and train a detector in PyGOD. Here, we use `pygod.detector.DOMINANT` as an example. For faster demonstration, we set `epoch` to 3.

```
from pygod.detector import DOMINANT

detector = DOMINANT(epoch=3)
detector.fit(data)
```

```
DOMINANT(act=<function relu at 0x7fb2fbf98e50>,
         backbone=<class 'torch_geometric.nn.models.basic_gnn.GCN'>,
         batch_size=8405, compile_model=False, contamination=0.1,
         dropout=0.0, epoch=3, gpu=None, hid_dim=64, lr=0.004,
         num_layers=4, num_neigh=[-1, -1, -1, -1], save_emb=False,
         sigmoid_s=False, verbose=0, weight=0.5, weight_decay=0.0)
```

### 2.2.3 Obtaining Node Score

After training, we obtain raw outlier scores for each node with `predict`. The shape of `node_score` is `(N, )`.

```
node_score = detector.predict(data, return_pred=False, return_score=True)
print(node_score.shape)
```

```
torch.Size([8405])
```

### 2.2.4 Converting Score to Edge Level

To detect outlier edges, we convert the outlier scores on node level to edge level. The shape of `edge_score` is `(E, )`.

```
from pygod.utils import to_edge_score

edge_score = to_edge_score(node_score, data.edge_index)
print(edge_score.shape)
```

```
torch.Size([407963])
```

### 2.2.5 Converting Score to Graph Level

To detect outlier graphs, we convert the outlier scores on node level to graph level for each graph. `graph_score` is a scalar for each `Data` object. Here, we give an example for scoring a list of graph.

```
from pygod.utils import to_graph_score

data_list = [data, data, data]
graph_scores = []
for data in data_list:
    node_score = detector.predict(data, return_pred=False, return_score=True)
    graph_score = to_graph_score(node_score)
```

(continues on next page)

(continued from previous page)

```
    graph_scores.append(graph_score.item())

print(graph_scores)
```

```
[12258.41796875, 12258.41796875, 12258.41796875]
```

**Total running time of the script:** (0 minutes 31.379 seconds)

CHAPTER

# THREE

# API CHEATSHEET

The following APIs are applicable for all detectors for easy use.

- *pygod.detector.Detector.fit()*: Fit the detector with train data.
- *pygod.detector.Detector.predict()*: Predict on test data (train data if not provided) using the fitted detector.

Key Attributes of a fitted detector:

- *pygod.detector.Detector.decision_score_*: The outlier scores of the input data. Outliers tend to have higher scores.
- *pygod.detector.Detector.label_*: The binary labels of the input data. 0 stands for inliers and 1 for outliers.
- *pygod.detector.Detector.threshold_* : The determined threshold for binary classification. Scores above the threshold are outliers.

**Input of PyGOD**: Please pass in a PyG Data object. See PyG data processing examples.

## 3.1 Base Detector

`Detector` is the abstract class for all detectors:

**class** pygod.detector.**Detector**(*contamination=0.1*, *verbose=0*)

Bases: ABC

Abstract class for all outlier detection algorithms.

**Parameters**

- **contamination** (*float, optional*) – The amount of contamination of the dataset in (0., 0.5], i.e., the proportion of outliers in the dataset. Used when fitting to define the threshold on the decision function. Default: `0.1`.
- **verbose** (*int, optional*) – Verbosity mode. Range in [0, 3]. Larger value for printing out more log information. Default: `0`.

**decision_score_**

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

**Type**
torch.Tensor

**threshold_**

The threshold is based on `contamination`. It is the $N\times$ `contamination` most abnormal samples in `decision_score_`. The threshold is calculated for generating binary outlier labels.

**Type**
float

**label_**

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

**Type**
torch.Tensor

**abstract decision_function**(*data*, *label=None*)

Predict raw outlier scores of testing data using the fitted detector. Outliers are assigned with higher outlier scores.

**Parameters**

- **data** (*torch_geometric.data.Data*) – The testing graph.
- **label** (*torch.Tensor, optional*) – The optional outlier ground truth labels used for testing. Default: `None`.

**Returns**
**score** – The outlier scores of shape $N$.

**Return type**
torch.Tensor

**abstract fit**(*data*, *label=None*)

Fit detector with training data.

**Parameters**

- **data** (*torch_geometric.data.Data*) – The training graph.
- **label** (*torch.Tensor, optional*) – The optional outlier ground truth labels used to monitor the training progress. They are not used to optimize the unsupervised model. Default: `None`.

**Returns**
**self** – Fitted detector.

**Return type**
object

**predict**(*data=None*, *label=None*, *return_pred=True*, *return_score=False*, *return_prob=False*, *prob_method='linear'*, *return_conf=False*)

Prediction for testing data using the fitted detector. Return predicted labels by default.

**Parameters**

- **data** (*torch_geometric.data.Data, optional*) – The testing graph. If `None`, the training data is used. Default: `None`.
- **label** (*torch.Tensor, optional*) – The optional outlier ground truth labels used for testing. Default: `None`.
- **return_pred** (*bool, optional*) – Whether to return the predicted binary labels. The labels are determined by the outlier contamination on the raw outlier scores. Default: `True`.

- **return_score** (*bool, optional*) – Whether to return the raw outlier scores. Default: `False`.
- **return_prob** (*bool, optional*) – Whether to return the outlier probabilities. Default: `False`.
- **prob_method** (*str, optional*) – The method to convert the outlier scores to probabilities. Two approaches are possible:

  1. `'linear'`: simply use min-max conversion to linearly transform the outlier scores into the range of [0,1]. The model must be fitted first.

  2. `'unify'`: use unifying scores, see [KKSZ11].

  Default: `'linear'`.
- **return_conf** (*boolean, optional*) – Whether to return the model's confidence in making the same prediction under slightly different training sets. See [PVD20]. Default: `False`.

**Returns**

- **pred** (*torch.Tensor*) – The predicted binary outlier labels of shape $N$. 0 stands for inliers and 1 for outliers. Only available when `return_label=True`.
- **score** (*torch.Tensor*) – The raw outlier scores of shape $N$. Only available when `return_score=True`.
- **prob** (*torch.Tensor*) – The outlier probabilities of shape $N$. Only available when `return_prob=True`.
- **conf** (*torch.Tensor*) – The prediction confidence of shape $N$. Only available when `return_conf=True`.

**abstract process_graph**(*data*)

Data preprocessing for the input graph.

**Parameters**

**data** (*torch_geometric.data.Data*) – The input graph.

## 3.2 Deep Detector

By inherit `Detector` class, we also provide base deep detector class for deep learning based detectors to ease the implementation.

**class** pygod.detector.**DeepDetector**(*hid_dim=64*, *num_layers=2*, *dropout=0.0*, *weight_decay=0.0*, *act=<function relu>*, *backbone=<class 'torch_geometric.nn.models.basic_gnn.GIN'>*, *contamination=0.1*, *lr=0.004*, *epoch=100*, *gpu=-1*, *batch_size=0*, *num_neigh=-1*, *verbose=0*, *gan=False*, *save_emb=False*, *compile_model=False*, ***kwargs*)

Abstract class for deep outlier detection algorithms.

**Parameters**

- **hid_dim** (*int, optional*) – Hidden dimension of model. Default: `64`.
- **num_layers** (*int, optional*) – Total number of layers in model. Default: `2`.
- **dropout** (*float, optional*) – Dropout rate. Default: `0.`.

- **weight_decay** (*float, optional*) – Weight decay (L2 penalty). Default: 0..
- **act** (*callable activation function or None, optional*) – Activation function if not None. Default: torch.nn.functional.relu.
- **backbone** (*torch.nn.Module*) – The backbone of the deep detector implemented in PyG. Default: torch_geometric.nn.GIN.
- **contamination** (*float, optional*) – The amount of contamination of the dataset in (0., 0.5], i.e., the proportion of outliers in the dataset. Used when fitting to define the threshold on the decision function. Default: 0.1.
- **lr** (*float, optional*) – Learning rate. Default: 0.004.
- **epoch** (*int, optional*) – Maximum number of training epoch. Default: 100.
- **gpu** (*int*) – GPU Index, -1 for using CPU. Default: -1.
- **batch_size** (*int, optional*) – Minibatch size, 0 for full batch training. Default: 0.
- **num_neigh** (*int, optional*) – Number of neighbors in sampling, -1 for all neighbors. Default: -1.
- **gan** (*bool, optional*) – Whether using adversarial training. Default: False.
- **verbose** (*int, optional*) – Verbosity mode. Range in [0, 3]. Larger value for printing out more log information. Default: 0.
- **save_emb** (*bool, optional*) – Whether to save the embedding. Default: False.
- **compile_model** (*bool, optional*) – Whether to compile the model with torch_geometric.compile. Default: False.
- ****kwargs** – Other parameters for the backbone.

**decision_score_**

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

**Type**
torch.Tensor

**threshold_**

The threshold is based on contamination. It is the $N\times$ contamination most abnormal samples in decision_score_. The threshold is calculated for generating binary outlier labels.

**Type**
float

**label_**

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

**Type**
torch.Tensor

**emb**

The learned node hidden embeddings of shape $N\times$ hid_dim. Only available when save_emb is True. When the detector has not been fitted, emb is None. When the detector has multiple embeddings, emb is a tuple of torch.Tensor.

**Type**
torch.Tensor or tuple of torch.Tensor or None

**abstract forward_model**(*data*)

Forward pass of the neural network detector.

**Parameters**
**data** (*torch_geometric.data.Data*) – The input graph.

**Returns**

- **loss** (*torch.Tensor*) – The loss of the current batch.
- **score** (*torch.Tensor*) – The outlier scores of the current batch.

**abstract init_model**(***kwargs*)

Initialize the neural network detector.

**Returns**
**model** – The initialized neural network detector.

**Return type**
torch.nn.Module

**abstract process_graph**(*data*)

Data preprocessing for the input graph.

**Parameters**
**data** (*torch_geometric.data.Data*) – The input graph.

CHAPTER

# FOUR

# EFFICIENT GPU TRAINING

To train deep detectors efficiently, we usually use CUDA to accelerate the detector training on GPU. PyGOD provides `gpu` parameter for `DeepDetector`. During initialization, we can set `gpu` to the index of the GPU that is available. By default, `gpu=-1`, which means train the detector on CPU. Here is an example of initialize `DOMINANT` with the first GPU (index of `0`):

```
DOMINANT(gpu=0)
```

However, training deep detectors on large-scale graphs can be memory-intensive, especially on the detectors relying on adjacency matrix reconstruction. At this time, full batch training may result in out-of-memory (OOM) error. As such, we divide the large graph into minibatches, and train the detector on each batch. PyGOD provides `batch_size` parameter for `DeepDetector`, where users are able to adjust the size of each batch for various GPU memory. We recommend users setting `batch_size` to largest value that will not cause OOM. For instance, we would like to train `DOMINANT` with the batches of 64 nodes:

```
DOMINANT(gpu=0, batch_size=64)
```

Unlike other data modalities, the output of each node in graphs rely on its neighbors. In PyGOD implementation, we adopt the data loader `torch_geometric.loader.NeighborLoader` in PyG to load both the center nodes and the neighbor nodes for minibatches. But the computation on neighbor nodes will lead to significant overhead and reduce the efficiency in the detector training. Thus, we neighbor sampling is crucial to reduce the overhead. PyGOD provides `num_neigh` parameter for `DeepDetector`. We can specify how many neighbors are sampled at each layer of the detector. The default value of `num_neigh` is `-1`, indicating sample all neighbors of the center node. If we want to sample 5 neighbors at each layer, we can initialize `DOMINANT` like:

```
DOMINANT(gpu=0, batch_size=64, num_neigh=5)
```

We can also sample different number of neighbors at each layer by setting `num_neigh` as a list, but the length of the list has to match with the number of layers `num_layers`:

```
DOMINANT(gpu=0, batch_size=64, num_layers=2, num_neigh=[5, 3])
```

To learn more, read PyG's tutorial on Scaling GNNs via Neighbor Sampling.

CHAPTER

FIVE

# PYGOD.DETECTOR

| | |
|---|---|
| *AdONE* | Adversarial Outlier Aware Attributed Network Embedding |
| *ANOMALOUS* | A Joint Modeling Approach for Anomaly Detection on Attributed Networks |
| *AnomalyDAE* | Dual Autoencoder for Anomaly Detection on Attributed Networks |
| *CoLA* | Anomaly Detection on Attributed Networks via Contrastive Self-Supervised Learning |
| *CONAD* | Contrastive Attributed Network Anomaly Detection |
| *DMGD* | Deep Multiclass Graph Description |
| *DOMINANT* | Deep Anomaly Detection on Attributed Networks |
| *DONE* | Deep Outlier Aware Attributed Network Embedding |
| *GAAN* | Generative Adversarial Attributed Network Anomaly Detection |
| *GADNR* | Graph Anomaly Detection via Neighborhood Reconstruction |
| *GAE* | Graph Autoencoder |
| *GUIDE* | Higher-order Structure based Anomaly Detection on Attributed Networks |
| *OCGNN* | One-Class Graph Neural Networks for Anomaly Detection in Attributed Networks |
| *ONE* | Outlier Aware Network Embedding for Attributed Networks |
| *Radar* | Residual Analysis for Anomaly Detection in Attributed Networks |
| *SCAN* | Structural Clustering Algorithm for Networks |

## 5.1 AdONE

**class** pygod.detector.**AdONE**(*hid_dim=64*, *num_layers=4*, *dropout=0.0*, *weight_decay=0.0*, *act=<function relu>*, *backbone=None*, *w1=0.2*, *w2=0.2*, *w3=0.2*, *w4=0.2*, *w5=0.2*, *contamination=0.1*, *lr=0.004*, *epoch=100*, *gpu=-1*, *batch_size=0*, *num_neigh=-1*, *save_emb=False*, *compile_model=False*, *verbose=0*, ***kwargs*)

Bases: *DeepDetector*

Adversarial Outlier Aware Attributed Network Embedding

AdONE consists of an attribute autoencoder and a structure autoencoder. It estimates five loss to optimize the

model, including an attribute proximity loss, an attribute homophily loss, a structure proximity loss, a structure homophily loss, and an alignment loss. It calculates three outlier scores, and averages them as an overall score.

**Note:** This detector is transductive only. Using `predict` with unseen data will train the detector from scratch.

See [BVM20] for details.

**Parameters**

- **hid_dim** (*int, optional*) – Hidden dimension of model. Default: `64`.
- **num_layers** (*int, optional*) – Total number of layers in model. A half (floor) of the layers are for the encoder, the other half (ceil) of the layers are for decoders. Default: `4`.
- **dropout** (*float, optional*) – Dropout rate. Default: `0.`.
- **weight_decay** (*float, optional*) – Weight decay (L2 penalty). Default: `0.`.
- **act** (*callable activation function or None, optional*) – Activation function if not None. Default: `torch.nn.functional.relu`.
- **backbone** (*torch.nn.Module*) – The backbone of AdONE is fixed to be MLP. Changing of this parameter will not affect the model. Default: `None`.
- **w1** (*float, optional*) – Weight of structure proximity loss. Default: `0.2`.
- **w2** (*float, optional*) – Weight of structure homophily loss. Default: `0.2`.
- **w3** (*float, optional*) – Weight of attribute proximity loss. Default: `0.2`.
- **w4** (*float, optional*) – Weight of attribute homophily loss. Default: `0.2`.
- **w5** (*float, optional*) – Weight of alignment loss. Default: `0.2`.
- **contamination** (*float, optional*) – The amount of contamination of the dataset in (0., 0.5], i.e., the proportion of outliers in the dataset. Used when fitting to define the threshold on the decision function. Default: `0.1`.
- **lr** (*float, optional*) – Learning rate. Default: `0.004`.
- **epoch** (*int, optional*) – Maximum number of training epoch. Default: `100`.
- **gpu** (*int*) – GPU Index, -1 for using CPU. Default: `-1`.
- **batch_size** (*int, optional*) – Minibatch size, 0 for full batch training. Default: `0`.
- **num_neigh** (*int, optional*) – Number of neighbors in sampling, -1 for all neighbors. Default: `-1`.
- **verbose** (*int, optional*) – Verbosity mode. Range in [0, 3]. Larger value for printing out more log information. Default: `0`.
- **save_emb** (*bool, optional*) – Whether to save the embedding. Default: `False`.
- **compile_model** (*bool, optional*) – Whether to compile the model with `torch_geometric.compile`. Default: `False`.
- ****kwargs** – Other parameters for the backbone model.

`decision_score_`

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

**Type**
torch.Tensor

**threshold_**

The threshold is based on `contamination`. It is the $N\times$ `contamination` most abnormal samples in `decision_score_`. The threshold is calculated for generating binary outlier labels.

**Type**
float

**label_**

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

**Type**
torch.Tensor

**emb**

The learned node hidden embeddings of shape $N\times$ `hid_dim`. Only available when `save_emb` is `True`. When the detector has not been fitted, `emb` is `None`. When the detector has multiple embeddings, `emb` is a tuple of torch.Tensor.

**Type**
torch.Tensor or tuple of torch.Tensor or None

**attribute_score_**

Attribute outlier score.

**Type**
torch.Tensor

**structural_score_**

Structural outlier score.

**Type**
torch.Tensor

**combined_score_**

Combined outlier score.

**Type**
torch.Tensor

**fit**(*data*, *label=None*)

Fit detector with training data.

**Parameters**

- **data** (*torch_geometric.data.Data*) – The training graph.
- **label** (*torch.Tensor, optional*) – The optional outlier ground truth labels used to monitor the training progress. They are not used to optimize the unsupervised model. Default: `None`.

**Returns**
**self** – Fitted detector.

**Return type**
object

**predict**(*data=None*, *label=None*, *return_pred=True*, *return_score=False*, *return_prob=False*, *prob_method='linear'*, *return_conf=False*, *return_emb=False*)

Prediction for testing data using the fitted detector. Return predicted labels by default.

**Parameters**

- **data** (*torch_geometric.data.Data, optional*) – The testing graph. If `None`, the training data is used. Default: `None`.
- **label** (*torch.Tensor, optional*) – The optional outlier ground truth labels used for testing. Default: `None`.
- **return_pred** (*bool, optional*) – Whether to return the predicted binary labels. The labels are determined by the outlier contamination on the raw outlier scores. Default: `True`.
- **return_score** (*bool, optional*) – Whether to return the raw outlier scores. Default: `False`.
- **return_prob** (*bool, optional*) – Whether to return the outlier probabilities. Default: `False`.
- **prob_method** (*str, optional*) – The method to convert the outlier scores to probabilities. Two approaches are possible:

  1. `'linear'`: simply use min-max conversion to linearly transform the outlier scores into the range of [0,1]. The model must be fitted first.

  2. `'unify'`: use unifying scores, see [KKSZ11].

  Default: `'linear'`.
- **return_conf** (*boolean, optional*) – Whether to return the model's confidence in making the same prediction under slightly different training sets. See [PVD20]. Default: `False`.
- **return_emb** (*bool, optional*) – Whether to return the learned node representations. Default: `False`.

**Returns**

- **pred** (*torch.Tensor*) – The predicted binary outlier labels of shape $N$. 0 stands for inliers and 1 for outliers. Only available when `return_label=True`.
- **score** (*torch.Tensor*) – The raw outlier scores of shape $N$. Only available when `return_score=True`.
- **prob** (*torch.Tensor*) – The outlier probabilities of shape $N$. Only available when `return_prob=True`.
- **conf** (*torch.Tensor*) – The prediction confidence of shape $N$. Only available when `return_conf=True`.

## 5.2 ANOMALOUS

**class** pygod.detector.**ANOMALOUS**(*gamma=1.0*, *weight_decay=0.0*, *lr=0.004*, *epoch=100*, *gpu=-1*, *contamination=0.1*, *verbose=0*)

Bases: *Detector*

A Joint Modeling Approach for Anomaly Detection on Attributed Networks

ANOMALOUS is an anomaly detector with CUR decomposition and residual analysis.

**Note:** This detector is transductive only. Using `predict` with unseen data will train the detector from scratch.

See [PLL+18] for details.

**Parameters**

- **gamma** (*float, optional*) – Weight of node feature w.r.t. structure. Default: 1.
- **weight_decay** (*float, optional*) – Weight decay (L2 penalty). Default: 0..
- **lr** (*float, optional*) – Learning rate. Default: 0.004.
- **epoch** (*int, optional*) – Maximum number of training epoch. Default: 100.
- **gpu** (*int*) – GPU Index, -1 for using CPU. Default: -1.
- **contamination** (*float, optional*) – Valid in (0., 0.5). The proportion of outliers in the data set. Used when fitting to define the threshold on the decision function. Default: 0.1.
- **verbose** (*int, optional*) – Verbosity mode. Range in [0, 3]. Larger value for printing out more log information. Default: 0.

**fit**(*data*, *label=None*)

Fit detector with training data.

**Parameters**

- **data** (*torch_geometric.data.Data*) – The training graph.
- **label** (*torch.Tensor, optional*) – The optional outlier ground truth labels used to monitor the training progress. They are not used to optimize the unsupervised model. Default: None.

**Returns**

**self** – Fitted detector.

**Return type**

object

**predict**(*data=None*, *label=None*, *return_pred=True*, *return_score=False*, *return_prob=False*, *prob_method='linear'*, *return_conf=False*)

Prediction for testing data using the fitted detector. Return predicted labels by default.

**Parameters**

- **data** (*torch_geometric.data.Data, optional*) – The testing graph. If None, the training data is used. Default: None.
- **label** (*torch.Tensor, optional*) – The optional outlier ground truth labels used for testing. Default: None.
- **return_pred** (*bool, optional*) – Whether to return the predicted binary labels. The labels are determined by the outlier contamination on the raw outlier scores. Default: True.
- **return_score** (*bool, optional*) – Whether to return the raw outlier scores. Default: False.
- **return_prob** (*bool, optional*) – Whether to return the outlier probabilities. Default: False.
- **prob_method** (*str, optional*) – The method to convert the outlier scores to probabilities. Two approaches are possible:

  1. 'linear': simply use min-max conversion to linearly transform the outlier scores into the range of [0,1]. The model must be fitted first.

  2. 'unify': use unifying scores, see [KKSZ11].

  Default: 'linear'.

- **return_conf** (*boolean, optional*) – Whether to return the model's confidence in making the same prediction under slightly different training sets. See [PVD20]. Default: `False`.

**Returns**

- **pred** (*torch.Tensor*) – The predicted binary outlier labels of shape $N$. 0 stands for inliers and 1 for outliers. Only available when `return_label=True`.
- **score** (*torch.Tensor*) – The raw outlier scores of shape $N$. Only available when `return_score=True`.
- **prob** (*torch.Tensor*) – The outlier probabilities of shape $N$. Only available when `return_prob=True`.
- **conf** (*torch.Tensor*) – The prediction confidence of shape $N$. Only available when `return_conf=True`.

## 5.3 AnomalyDAE

**class** pygod.detector.**AnomalyDAE**(*emb_dim=64*, *hid_dim=64*, *num_layers=4*, *dropout=0.0*, *weight_decay=0.0*, *act=<function relu>*, *backbone=None*, *alpha=0.5*, *theta=1.0*, *eta=1.0*, *contamination=0.1*, *lr=0.004*, *epoch=5*, *gpu=-1*, *batch_size=0*, *num_neigh=-1*, *verbose=0*, *save_emb=False*, *compile_model=False*, ***kwargs*)

Bases: *DeepDetector*

Dual Autoencoder for Anomaly Detection on Attributed Networks

AnomalyDAE is an anomaly detector that consists of a structure autoencoder and an attribute autoencoder to learn both node embedding and attribute embedding jointly in latent space. The structural autoencoder uses Graph Attention layers. The reconstruction mean square error of the decoders are defined as structure anomaly score and attribute anomaly score, respectively, with two additional penalties on the reconstructed adj matrix and node attributes (force entries to be nonzero).

See [FZL20] for details.

**Parameters**

- **emb_dim** (*int, optional*) – Embedding dimension of model. Default: `64`.
- **hid_dim** (*int, optional*) – Hidden dimension of model. Default: `64`.
- **num_layers** (*int, optional*) – Total number of layers of AnomalyDAE is fixed to be 4. Changing of this parameter will not affect the model. Default: `4`.
- **dropout** (*float, optional*) – Dropout rate. Default: `0.`.
- **weight_decay** (*float, optional*) – Weight decay (L2 penalty). Default: `0.`.
- **act** (*callable activation function or None, optional*) – Activation function if not None. Default: `torch.nn.functional.relu`.
- **backbone** (*torch.nn.Module*) – The backbone of AnomalyDAE is fixed. Changing of this parameter will not affect the model. Default: `None`.
- **alpha** (*float, optional*) – Weight between reconstruction of node feature and structure. Default: `0.5`.
- **theta** (*float, optional*) – The additional penalty for nonzero attribute. Default: `1.`.

- **eta** (*float, optional*) – The additional penalty for nonzero structure. Default: `1.`.
- **contamination** (*float, optional*) – The amount of contamination of the dataset in (0., 0.5], i.e., the proportion of outliers in the dataset. Used when fitting to define the threshold on the decision function. Default: `0.1`.
- **lr** (*float, optional*) – Learning rate. Default: `0.004`.
- **epoch** (*int, optional*) – Maximum number of training epoch. Default: `100`.
- **gpu** (*int*) – GPU Index, -1 for using CPU. Default: `-1`.
- **batch_size** (*int, optional*) – Minibatch size, 0 for full batch training. Default: `0`.
- **num_neigh** (*int, optional*) – Number of neighbors in sampling, -1 for all neighbors. Default: `-1`.
- **verbose** (*int, optional*) – Verbosity mode. Range in [0, 3]. Larger value for printing out more log information. Default: `0`.
- **save_emb** (*bool, optional*) – Whether to save the embedding. Default: `False`.
- **compile_model** (*bool, optional*) – Whether to compile the model with `torch_geometric.compile`. Default: `False`.
- ****kwargs** – Other parameters for the backbone model.

**decision_score_**

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

**Type**
torch.Tensor

**threshold_**

The threshold is based on `contamination`. It is the $N\times$ `contamination` most abnormal samples in `decision_score_`. The threshold is calculated for generating binary outlier labels.

**Type**
float

**label_**

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

**Type**
torch.Tensor

**emb**

The learned node hidden embeddings of shape $N\times$ `hid_dim`. Only available when `save_emb` is `True`. When the detector has not been fitted, `emb` is `None`. When the detector has multiple embeddings, `emb` is a tuple of torch.Tensor.

**Type**
torch.Tensor or tuple of torch.Tensor or None

**fit**(*data*, *label=None*)

Fit detector with training data.

**Parameters**

- **data** (*torch_geometric.data.Data*) – The training graph.

- **label** (*torch.Tensor, optional*) – The optional outlier ground truth labels used to monitor the training progress. They are not used to optimize the unsupervised model. Default: `None`.

**Returns**
**self** – Fitted detector.

**Return type**
object

**predict**(*data=None*, *label=None*, *return_pred=True*, *return_score=False*, *return_prob=False*, *prob_method='linear'*, *return_conf=False*, *return_emb=False*)

Prediction for testing data using the fitted detector. Return predicted labels by default.

**Parameters**

- **data** (*torch_geometric.data.Data, optional*) – The testing graph. If `None`, the training data is used. Default: `None`.
- **label** (*torch.Tensor, optional*) – The optional outlier ground truth labels used for testing. Default: `None`.
- **return_pred** (*bool, optional*) – Whether to return the predicted binary labels. The labels are determined by the outlier contamination on the raw outlier scores. Default: `True`.
- **return_score** (*bool, optional*) – Whether to return the raw outlier scores. Default: `False`.
- **return_prob** (*bool, optional*) – Whether to return the outlier probabilities. Default: `False`.
- **prob_method** (*str, optional*) – The method to convert the outlier scores to probabilities. Two approaches are possible:

  1. `'linear'`: simply use min-max conversion to linearly transform the outlier scores into the range of [0,1]. The model must be fitted first.

  2. `'unify'`: use unifying scores, see [KKSZ11].

  Default: `'linear'`.
- **return_conf** (*boolean, optional*) – Whether to return the model's confidence in making the same prediction under slightly different training sets. See [PVD20]. Default: `False`.
- **return_emb** (*bool, optional*) – Whether to return the learned node representations. Default: `False`.

**Returns**

- **pred** (*torch.Tensor*) – The predicted binary outlier labels of shape $N$. 0 stands for inliers and 1 for outliers. Only available when `return_label=True`.
- **score** (*torch.Tensor*) – The raw outlier scores of shape $N$. Only available when `return_score=True`.
- **prob** (*torch.Tensor*) – The outlier probabilities of shape $N$. Only available when `return_prob=True`.
- **conf** (*torch.Tensor*) – The prediction confidence of shape $N$. Only available when `return_conf=True`.

## 5.4 CoLA

class pygod.detector.**CoLA**(*hid_dim=64*, *num_layers=4*, *dropout=0.0*, *weight_decay=0.0*, *act=<function relu>*, *backbone=<class 'torch_geometric.nn.models.basic_gnn.GCN'>*, *contamination=0.1*, *lr=0.004*, *epoch=100*, *gpu=-1*, *batch_size=0*, *num_neigh=-1*, *verbose=0*, *save_emb=False*, *compile_model=False*, ***kwargs*)

Bases: *DeepDetector*

Anomaly Detection on Attributed Networks via Contrastive Self-Supervised Learning

CoLA is a contrastive self-supervised learning based method for graph anomaly detection. This implementation is base on random neighbor sampling instead of random walk sampling in the original paper.

See [LLP+21] for details.

**Parameters**

- **hid_dim** (*int, optional*) – Hidden dimension of model. Default: `64`.
- **num_layers** (*int, optional*) – Total number of layers in model. Default: `4`.
- **dropout** (*float, optional*) – Dropout rate. Default: `0.`.
- **weight_decay** (*float, optional*) – Weight decay (L2 penalty). Default: `0.`.
- **act** (*callable activation function or None, optional*) – Activation function if not None. Default: `torch.nn.functional.relu`.
- **backbone** (*torch.nn.Module*) – The backbone of the deep detector implemented in PyG. Default: `torch_geometric.nn.GCN`.
- **contamination** (*float, optional*) – The amount of contamination of the dataset in (0., 0.5], i.e., the proportion of outliers in the dataset. Used when fitting to define the threshold on the decision function. Default: `0.1`.
- **lr** (*float, optional*) – Learning rate. Default: `0.004`.
- **epoch** (*int, optional*) – Maximum number of training epoch. Default: `100`.
- **gpu** (*int*) – GPU Index, -1 for using CPU. Default: `-1`.
- **batch_size** (*int, optional*) – Minibatch size, 0 for full batch training. Default: `0`.
- **num_neigh** (*int, optional*) – Number of neighbors in sampling, -1 for all neighbors. Default: `-1`.
- **verbose** (*int, optional*) – Verbosity mode. Range in [0, 3]. Larger value for printing out more log information. Default: `0`.
- **save_emb** (*bool, optional*) – Whether to save the embedding. Default: `False`.
- **compile_model** (*bool, optional*) – Whether to compile the model with `torch_geometric.compile`. Default: `False`.
- ****kwargs** – Other parameters for the backbone.

**decision_score_**

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

**Type**
torch.Tensor

**threshold_**

The threshold is based on `contamination`. It is the $N\times$ `contamination` most abnormal samples in `decision_score_`. The threshold is calculated for generating binary outlier labels.

**Type**
float

**label_**

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

**Type**
torch.Tensor

**emb**

The learned node hidden embeddings of shape $N\times$ `hid_dim`. Only available when `save_emb` is `True`. When the detector has not been fitted, `emb` is `None`. When the detector has multiple embeddings, `emb` is a tuple of torch.Tensor.

**Type**
torch.Tensor or tuple of torch.Tensor or None

**fit**(*data*, *label=None*)

Fit detector with training data.

**Parameters**

- **data** (*torch_geometric.data.Data*) – The training graph.
- **label** (*torch.Tensor, optional*) – The optional outlier ground truth labels used to monitor the training progress. They are not used to optimize the unsupervised model. Default: `None`.

**Returns**
**self** – Fitted detector.

**Return type**
object

**predict**(*data=None*, *label=None*, *return_pred=True*, *return_score=False*, *return_prob=False*, *prob_method='linear'*, *return_conf=False*, *return_emb=False*)

Prediction for testing data using the fitted detector. Return predicted labels by default.

**Parameters**

- **data** (*torch_geometric.data.Data, optional*) – The testing graph. If `None`, the training data is used. Default: `None`.
- **label** (*torch.Tensor, optional*) – The optional outlier ground truth labels used for testing. Default: `None`.
- **return_pred** (*bool, optional*) – Whether to return the predicted binary labels. The labels are determined by the outlier contamination on the raw outlier scores. Default: `True`.
- **return_score** (*bool, optional*) – Whether to return the raw outlier scores. Default: `False`.
- **return_prob** (*bool, optional*) – Whether to return the outlier probabilities. Default: `False`.
- **prob_method** (*str, optional*) – The method to convert the outlier scores to probabilities. Two approaches are possible:

1. 'linear': simply use min-max conversion to linearly transform the outlier scores into the range of [0,1]. The model must be fitted first.

2. 'unify': use unifying scores, see [KKSZ11].

Default: 'linear'.

- **return_conf** (*boolean, optional*) – Whether to return the model's confidence in making the same prediction under slightly different training sets. See [PVD20]. Default: False.
- **return_emb** (*bool, optional*) – Whether to return the learned node representations. Default: False.

**Returns**

- **pred** (*torch.Tensor*) – The predicted binary outlier labels of shape $N$. 0 stands for inliers and 1 for outliers. Only available when return_label=True.
- **score** (*torch.Tensor*) – The raw outlier scores of shape $N$. Only available when return_score=True.
- **prob** (*torch.Tensor*) – The outlier probabilities of shape $N$. Only available when return_prob=True.
- **conf** (*torch.Tensor*) – The prediction confidence of shape $N$. Only available when return_conf=True.

## 5.5 CONAD

class pygod.detector.**CONAD**(*hid_dim=64*, *num_layers=4*, *dropout=0.0*, *weight_decay=0.0*, *act=<function relu>*, *sigmoid_s=False*, *backbone=<class 'torch_geometric.nn.models.basic_gnn.GCN'>*, *contamination=0.1*, *lr=0.004*, *epoch=100*, *gpu=-1*, *batch_size=0*, *num_neigh=-1*, *weight=0.5*, *eta=0.5*, *margin=0.5*, *r=0.2*, *m=50*, *k=50*, *f=10*, *verbose=0*, *save_emb=False*, *compile_model=False*, ***kwargs*)

Bases: *DeepDetector*

Contrastive Attributed Network Anomaly Detection

CONAD is an anomaly detector consisting of a shared graph convolutional encoder, a structure reconstruction decoder, and an attribute reconstruction decoder. The model is trained with both contrastive loss and structure/attribute reconstruction loss. The reconstruction mean square error of the decoders are defined as structure anomaly score and attribute anomaly score, respectively.

See [XHZ+22] for details.

**Parameters**

- **hid_dim** (*int, optional*) – Hidden dimension of model. Default: 64.
- **num_layers** (*int, optional*) – Total number of layers in model. Default: 4.
- **dropout** (*float, optional*) – Dropout rate. Default: 0..
- **weight_decay** (*float, optional*) – Weight decay (L2 penalty). Default: 0..
- **act** (*callable activation function or None, optional*) – Activation function if not None. Default: torch.nn.functional.relu.

- **sigmoid_s** (*bool, optional*) – Whether to use sigmoid function to scale the reconstructed structure. Default: `False`.
- **backbone** (*torch.nn.Module, optional*) – The backbone of the deep detector implemented in PyG. Default: `torch_geometric.nn.GCN`.
- **contamination** (*float, optional*) – The amount of contamination of the dataset in (0., 0.5], i.e., the proportion of outliers in the dataset. Used when fitting to define the threshold on the decision function. Default: `0.1`.
- **lr** (*float, optional*) – Learning rate. Default: `0.004`.
- **epoch** (*int, optional*) – Maximum number of training epoch. Default: `100`.
- **gpu** (*int*) – GPU Index, -1 for using CPU. Default: `-1`.
- **batch_size** (*int, optional*) – Minibatch size, 0 for full batch training. Default: `0`.
- **num_neigh** (*int, optional*) – Number of neighbors in sampling, -1 for all neighbors. Default: `-1`.
- **weight** (*float, optional*) – Weight between reconstruction of node feature and structure. Default: `0.5`.
- **eta** (*float, optional*) – Weight between contrastive and reconstruction.Default: `0.5`.
- **margin** (*float, optional*) – Margin in margin ranking loss. Default: `0.5`.
- **r** (*float, optional*) – The rate of augmented anomalies. Default: `0.2`.
- **m** (*int, optional*) – For densely connected nodes, the number of edges to add. Default: `50`.
- **k** (*int, optional*) – Same as k in `pygod.generator.gen_contextual_outlier`. Default: `50`.
- **f** (*int, optional*) – For disproportionate nodes, the scale factor applied on their attribute value. Default: `10`.
- **verbose** (*int, optional*) – Verbosity mode. Range in [0, 3]. Larger value for printing out more log information. Default: `0`.
- **save_emb** (*bool, optional*) – Whether to save the embedding. Default: `False`.
- **compile_model** (*bool, optional*) – Whether to compile the model with `torch_geometric.compile`. Default: `False`.
- ****kwargs** (*optional*) – Additional arguments for the backbone.

**decision_score_**

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

**Type**
torch.Tensor

**threshold_**

The threshold is based on `contamination`. It is the $N\times$ `contamination` most abnormal samples in `decision_score_`. The threshold is calculated for generating binary outlier labels.

**Type**
float

**label_**

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

**Type**
torch.Tensor

**emb**

The learned node hidden embeddings of shape $N \times$ hid_dim. Only available when save_emb is True. When the detector has not been fitted, emb is None. When the detector has multiple embeddings, emb is a tuple of torch.Tensor.

**Type**
torch.Tensor or tuple of torch.Tensor or None

**fit**(*data*, *label=None*)

Fit detector with training data.

**Parameters**

- **data** (*torch_geometric.data.Data*) – The training graph.
- **label** (*torch.Tensor, optional*) – The optional outlier ground truth labels used to monitor the training progress. They are not used to optimize the unsupervised model. Default: None.

**Returns**
**self** – Fitted detector.

**Return type**
object

**predict**(*data=None*, *label=None*, *return_pred=True*, *return_score=False*, *return_prob=False*, *prob_method='linear'*, *return_conf=False*, *return_emb=False*)

Prediction for testing data using the fitted detector. Return predicted labels by default.

**Parameters**

- **data** (*torch_geometric.data.Data, optional*) – The testing graph. If None, the training data is used. Default: None.
- **label** (*torch.Tensor, optional*) – The optional outlier ground truth labels used for testing. Default: None.
- **return_pred** (*bool, optional*) – Whether to return the predicted binary labels. The labels are determined by the outlier contamination on the raw outlier scores. Default: True.
- **return_score** (*bool, optional*) – Whether to return the raw outlier scores. Default: False.
- **return_prob** (*bool, optional*) – Whether to return the outlier probabilities. Default: False.
- **prob_method** (*str, optional*) – The method to convert the outlier scores to probabilities. Two approaches are possible:

  1. 'linear': simply use min-max conversion to linearly transform the outlier scores into the range of [0,1]. The model must be fitted first.

  2. 'unify': use unifying scores, see [KKSZ11].

  Default: 'linear'.

- **return_conf** (*boolean, optional*) – Whether to return the model's confidence in making the same prediction under slightly different training sets. See [PVD20]. Default: `False`.
- **return_emb** (*bool, optional*) – Whether to return the learned node representations. Default: `False`.

**Returns**

- **pred** (*torch.Tensor*) – The predicted binary outlier labels of shape $N$. 0 stands for inliers and 1 for outliers. Only available when `return_label=True`.
- **score** (*torch.Tensor*) – The raw outlier scores of shape $N$. Only available when `return_score=True`.
- **prob** (*torch.Tensor*) – The outlier probabilities of shape $N$. Only available when `return_prob=True`.
- **conf** (*torch.Tensor*) – The prediction confidence of shape $N$. Only available when `return_conf=True`.

## 5.6 DMGD

**class** pygod.detector.**DMGD**(*hid_dim=64*, *num_layers=2*, *dropout=0.0*, *weight_decay=0.0*, *act=<function relu>*, *backbone=<class 'torch_geometric.nn.models.basic_gnn.GCN'>*, *contamination=0.1*, *lr=0.004*, *epoch=100*, *gpu=-1*, *batch_size=0*, *num_neigh=-1*, *alpha=1*, *beta=1*, *gamma=1*, *warmup=2*, *k=2*, *verbose=0*, *save_emb=False*, *compile_model=False*, ***kwargs*)

Bases: *DeepDetector*

Deep Multiclass Graph Description

DMGD is a support vector based multiclass outlier detector. Its backbone is an autoencoder that reconstructs the adjacency matrix of the graph with MSE loss and homophily loss. It applies k-means to cluster the nodes embedding and then uses support vector to detect outliers.

See [BVVM20] for details.

**Parameters**

- **hid_dim** (*int, optional*) – Hidden dimension of model. Default: `64`.
- **num_layers** (*int, optional*) – Total number of layers in model. Default: `2`.
- **dropout** (*float, optional*) – Dropout rate. Default: `0.`.
- **weight_decay** (*float, optional*) – Weight decay (L2 penalty). Default: `0.`.
- **act** (*callable activation function or None, optional*) – Activation function if not None. Default: `torch.nn.functional.relu`.
- **backbone** (*torch.nn.Module*) – The backbone of the deep detector implemented in PyG. Default: `torch_geometric.nn.GCN`.
- **contamination** (*float, optional*) – The amount of contamination of the dataset in (0., 0.5], i.e., the proportion of outliers in the dataset. Used when fitting to define the threshold on the decision function. Default: `0.1`.
- **lr** (*float, optional*) – Learning rate. Default: `0.004`.
- **epoch** (*int, optional*) – Maximum number of training epoch. Default: `100`.

- **gpu** (*int*) – GPU Index, -1 for using CPU. Default: `-1`.
- **batch_size** (*int, optional*) – Minibatch size, 0 for full batch training. Default: `0`.
- **num_neigh** (*int, optional*) – Number of neighbors in sampling, -1 for all neighbors. Default: `-1`.
- **alpha** (*float, optional*) – Weight of the radius loss. Default: `1`.
- **beta** (*float, optional*) – Weight of the reconstruction loss. Default: `1`.
- **gamma** (*float, optional*) – Weight of the homophily loss. Default: `1`.
- **warmup** (*int, optional*) – The number of epochs for warm-up training. Default: `2`.
- **k** (*int, optional*) – The number of clusters. Default: `2`.
- **verbose** (*int, optional*) – Verbosity mode. Range in [0, 3]. Larger value for printing out more log information. Default: `0`.
- **save_emb** (*bool, optional*) – Whether to save the embedding. Default: `False`.
- **compile_model** (*bool, optional*) – Whether to compile the model with `torch_geometric.compile`. Default: `False`.
- ****kwargs** – Other parameters for the backbone model.

**decision_score_**

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

**Type**
torch.Tensor

**threshold_**

The threshold is based on `contamination`. It is the $N\times$ `contamination` most abnormal samples in `decision_score_`. The threshold is calculated for generating binary outlier labels.

**Type**
float

**label_**

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

**Type**
torch.Tensor

**emb**

The learned node hidden embeddings of shape $N\times$ `hid_dim`. Only available when `save_emb` is `True`. When the detector has not been fitted, `emb` is `None`. When the detector has multiple embeddings, `emb` is a tuple of torch.Tensor.

**Type**
torch.Tensor or tuple of torch.Tensor or None

**fit**(*data*, *label=None*)

Fit detector with training data.

**Parameters**

- **data** (*torch_geometric.data.Data*) – The training graph.

- **label** (*torch.Tensor, optional*) – The optional outlier ground truth labels used to monitor the training progress. They are not used to optimize the unsupervised model. Default: `None`.

**Returns**
**self** – Fitted detector.

**Return type**
object

**predict**(*data=None*, *label=None*, *return_pred=True*, *return_score=False*, *return_prob=False*, *prob_method='linear'*, *return_conf=False*, *return_emb=False*)

Prediction for testing data using the fitted detector. Return predicted labels by default.

**Parameters**

- **data** (*torch_geometric.data.Data, optional*) – The testing graph. If `None`, the training data is used. Default: `None`.
- **label** (*torch.Tensor, optional*) – The optional outlier ground truth labels used for testing. Default: `None`.
- **return_pred** (*bool, optional*) – Whether to return the predicted binary labels. The labels are determined by the outlier contamination on the raw outlier scores. Default: `True`.
- **return_score** (*bool, optional*) – Whether to return the raw outlier scores. Default: `False`.
- **return_prob** (*bool, optional*) – Whether to return the outlier probabilities. Default: `False`.
- **prob_method** (*str, optional*) – The method to convert the outlier scores to probabilities. Two approaches are possible:

  1. `'linear'`: simply use min-max conversion to linearly transform the outlier scores into the range of [0,1]. The model must be fitted first.

  2. `'unify'`: use unifying scores, see [KKSZ11].

  Default: `'linear'`.
- **return_conf** (*boolean, optional*) – Whether to return the model's confidence in making the same prediction under slightly different training sets. See [PVD20]. Default: `False`.
- **return_emb** (*bool, optional*) – Whether to return the learned node representations. Default: `False`.

**Returns**

- **pred** (*torch.Tensor*) – The predicted binary outlier labels of shape $N$. 0 stands for inliers and 1 for outliers. Only available when `return_label=True`.
- **score** (*torch.Tensor*) – The raw outlier scores of shape $N$. Only available when `return_score=True`.
- **prob** (*torch.Tensor*) – The outlier probabilities of shape $N$. Only available when `return_prob=True`.
- **conf** (*torch.Tensor*) – The prediction confidence of shape $N$. Only available when `return_conf=True`.

## 5.7 DOMINANT

class pygod.detector.**DOMINANT**(*hid_dim=64*, *num_layers=4*, *dropout=0.0*, *weight_decay=0.0*, *act=<function relu>*, *sigmoid_s=False*, *backbone=<class 'torch_geometric.nn.models.basic_gnn.GCN'>*, *contamination=0.1*, *lr=0.004*, *epoch=100*, *gpu=-1*, *batch_size=0*, *num_neigh=-1*, *weight=0.5*, *verbose=0*, *save_emb=False*, *compile_model=False*, ***kwargs*)

Bases: *DeepDetector*

Deep Anomaly Detection on Attributed Networks

DOMINANT is an anomaly detector consisting of a shared graph convolutional encoder, a structure reconstruction decoder, and an attribute reconstruction decoder. The reconstruction mean squared error of the decoders are defined as structure anomaly score and attribute anomaly score, respectively.

See [DLBL19] for details.

**Parameters**

- **hid_dim** (*int, optional*) – Hidden dimension of model. Default: `64`.
- **num_layers** (*int, optional*) – Total number of layers in model. A half (floor) of the layers are for the encoder, the other half (ceil) of the layers are for decoders. Default: `4`.
- **dropout** (*float, optional*) – Dropout rate. Default: `0.`.
- **weight_decay** (*float, optional*) – Weight decay (L2 penalty). Default: `0.`.
- **act** (*callable activation function or None, optional*) – Activation function if not None. Default: `torch.nn.functional.relu`.
- **sigmoid_s** (*bool, optional*) – Whether to use sigmoid function to scale the reconstructed structure. Default: `False`.
- **backbone** (*torch.nn.Module, optional*) – The backbone of the deep detector implemented in PyG. Default: `torch_geometric.nn.GCN`.
- **contamination** (*float, optional*) – The amount of contamination of the dataset in (0., 0.5], i.e., the proportion of outliers in the dataset. Used when fitting to define the threshold on the decision function. Default: `0.1`.
- **lr** (*float, optional*) – Learning rate. Default: `0.004`.
- **epoch** (*int, optional*) – Maximum number of training epoch. Default: `100`.
- **gpu** (*int*) – GPU Index, -1 for using CPU. Default: `-1`.
- **batch_size** (*int, optional*) – Minibatch size, 0 for full batch training. Default: `0`.
- **num_neigh** (*int, optional*) – Number of neighbors in sampling, -1 for all neighbors. Default: `-1`.
- **weight** (*float, optional*) – Weight between reconstruction of node feature and structure. Default: `0.5`.
- **verbose** (*int, optional*) – Verbosity mode. Range in [0, 3]. Larger value for printing out more log information. Default: `0`.
- **save_emb** (*bool, optional*) – Whether to save the embedding. Default: `False`.
- **compile_model** (*bool, optional*) – Whether to compile the model with `torch_geometric.compile`. Default: `False`.
- ****kwargs** (*optional*) – Additional arguments for the backbone.

**decision_score_**

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

**Type**
torch.Tensor

**threshold_**

The threshold is based on `contamination`. It is the $N\times$ `contamination` most abnormal samples in `decision_score_`. The threshold is calculated for generating binary outlier labels.

**Type**
float

**label_**

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

**Type**
torch.Tensor

**emb**

The learned node hidden embeddings of shape $N\times$ `hid_dim`. Only available when `save_emb` is `True`. When the detector has not been fitted, `emb` is `None`. When the detector has multiple embeddings, `emb` is a tuple of torch.Tensor.

**Type**
torch.Tensor or tuple of torch.Tensor or None

**fit**(*data*, *label=None*)

Fit detector with training data.

**Parameters**

- **data** (*torch_geometric.data.Data*) – The training graph.
- **label** (*torch.Tensor, optional*) – The optional outlier ground truth labels used to monitor the training progress. They are not used to optimize the unsupervised model. Default: `None`.

**Returns**
**self** – Fitted detector.

**Return type**
object

**predict**(*data=None*, *label=None*, *return_pred=True*, *return_score=False*, *return_prob=False*, *prob_method='linear'*, *return_conf=False*, *return_emb=False*)

Prediction for testing data using the fitted detector. Return predicted labels by default.

**Parameters**

- **data** (*torch_geometric.data.Data, optional*) – The testing graph. If `None`, the training data is used. Default: `None`.
- **label** (*torch.Tensor, optional*) – The optional outlier ground truth labels used for testing. Default: `None`.
- **return_pred** (*bool, optional*) – Whether to return the predicted binary labels. The labels are determined by the outlier contamination on the raw outlier scores. Default: `True`.

- **return_score** (*bool, optional*) – Whether to return the raw outlier scores. Default: `False`.
- **return_prob** (*bool, optional*) – Whether to return the outlier probabilities. Default: `False`.
- **prob_method** (*str, optional*) – The method to convert the outlier scores to probabilities. Two approaches are possible:

  1. `'linear'`: simply use min-max conversion to linearly transform the outlier scores into the range of [0,1]. The model must be fitted first.

  2. `'unify'`: use unifying scores, see [KKSZ11].

  Default: `'linear'`.
- **return_conf** (*boolean, optional*) – Whether to return the model's confidence in making the same prediction under slightly different training sets. See [PVD20]. Default: `False`.
- **return_emb** (*bool, optional*) – Whether to return the learned node representations. Default: `False`.

**Returns**

- **pred** (*torch.Tensor*) – The predicted binary outlier labels of shape $N$. 0 stands for inliers and 1 for outliers. Only available when `return_label=True`.
- **score** (*torch.Tensor*) – The raw outlier scores of shape $N$. Only available when `return_score=True`.
- **prob** (*torch.Tensor*) – The outlier probabilities of shape $N$. Only available when `return_prob=True`.
- **conf** (*torch.Tensor*) – The prediction confidence of shape $N$. Only available when `return_conf=True`.

## 5.8 DONE

**class** pygod.detector.**DONE**(*hid_dim=64*, *num_layers=4*, *dropout=0.0*, *weight_decay=0.0*, *act=<function relu>*, *backbone=None*, *w1=0.2*, *w2=0.2*, *w3=0.2*, *w4=0.2*, *w5=0.2*, *contamination=0.1*, *lr=0.004*, *epoch=100*, *gpu=-1*, *batch_size=0*, *num_neigh=-1*, *verbose=0*, *save_emb=False*, *compile_model=False*, ***kwargs*)

Bases: *DeepDetector*

Deep Outlier Aware Attributed Network Embedding

DONE consists of an attribute autoencoder and a structure autoencoder. It estimates five losses to optimize the model, including an attribute proximity loss, an attribute homophily loss, a structure proximity loss, a structure homophily loss, and a combination loss. It calculates three outlier scores, and averages them as an overall scores.

---

**Note:** This detector is transductive only. Using `predict` with unseen data will train the detector from scratch.

---

See [BVM20] for details.

**Parameters**

- **hid_dim** (*int, optional*) – Hidden dimension of model. Default: `64`.

- **num_layers** (*int, optional*) – Total number of layers in model. A half (floor) of the layers are for the encoder, the other half (ceil) of the layers are for decoders. Default: `4`.
- **dropout** (*float, optional*) – Dropout rate. Default: `0.`.
- **weight_decay** (*float, optional*) – Weight decay (L2 penalty). Default: `0.`.
- **act** (*callable activation function or None, optional*) – Activation function if not None. Default: `torch.nn.functional.relu`.
- **backbone** (*torch.nn.Module*) – The backbone of DONE is fixed to be MLP. Changing of this parameter will not affect the model. Default: `None`.
- **w1** (*float, optional*) – Weight of structure proximity loss. Default: `0.2`.
- **w2** (*float, optional*) – Weight of structure homophily loss. Default: `0.2`.
- **w3** (*float, optional*) – Weight of attribute proximity loss. Default: `0.2`.
- **w4** (*float, optional*) – Weight of attribute homophily loss. Default: `0.2`.
- **w5** (*float, optional*) – Weight of combination loss. Default: `0.2`.
- **contamination** (*float, optional*) – The amount of contamination of the dataset in (0., 0.5], i.e., the proportion of outliers in the dataset. Used when fitting to define the threshold on the decision function. Default: `0.1`.
- **lr** (*float, optional*) – Learning rate. Default: `0.004`.
- **epoch** (*int, optional*) – Maximum number of training epoch. Default: `100`.
- **gpu** (*int*) – GPU Index, -1 for using CPU. Default: `-1`.
- **batch_size** (*int, optional*) – Minibatch size, 0 for full batch training. Default: `0`.
- **num_neigh** (*int, optional*) – Number of neighbors in sampling, -1 for all neighbors. Default: `-1`.
- **verbose** (*int, optional*) – Verbosity mode. Range in [0, 3]. Larger value for printing out more log information. Default: `0`.
- **save_emb** (*bool, optional*) – Whether to save the embedding. Default: `False`.
- **compile_model** (*bool, optional*) – Whether to compile the model with `torch_geometric.compile`. Default: `False`.
- ****kwargs** – Other parameters for the backbone model.

**decision_score_**

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

**Type**
torch.Tensor

**threshold_**

The threshold is based on `contamination`. It is the $N\times$ `contamination` most abnormal samples in `decision_score_`. The threshold is calculated for generating binary outlier labels.

**Type**
float

**label_**

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

**Type**
torch.Tensor

**emb**

The learned node hidden embeddings of shape $N \times$ hid_dim. Only available when save_emb is True. When the detector has not been fitted, emb is None. When the detector has multiple embeddings, emb is a tuple of torch.Tensor.

**Type**
torch.Tensor or tuple of torch.Tensor or None

**attribute_score_**

Attribute outlier score.

**Type**
torch.Tensor

**structural_score_**

Structural outlier score.

**Type**
torch.Tensor

**combined_score_**

Combined outlier score.

**Type**
torch.Tensor

**fit**(*data*, *label=None*)

Fit detector with training data.

**Parameters**

- **data** (*torch_geometric.data.Data*) – The training graph.
- **label** (*torch.Tensor, optional*) – The optional outlier ground truth labels used to monitor the training progress. They are not used to optimize the unsupervised model. Default: None.

**Returns**
**self** – Fitted detector.

**Return type**
object

**predict**(*data=None*, *label=None*, *return_pred=True*, *return_score=False*, *return_prob=False*, *prob_method='linear'*, *return_conf=False*, *return_emb=False*)

Prediction for testing data using the fitted detector. Return predicted labels by default.

**Parameters**

- **data** (*torch_geometric.data.Data, optional*) – The testing graph. If None, the training data is used. Default: None.
- **label** (*torch.Tensor, optional*) – The optional outlier ground truth labels used for testing. Default: None.
- **return_pred** (*bool, optional*) – Whether to return the predicted binary labels. The labels are determined by the outlier contamination on the raw outlier scores. Default: True.
- **return_score** (*bool, optional*) – Whether to return the raw outlier scores. Default: False.

- **return_prob** (*bool, optional*) – Whether to return the outlier probabilities. Default: `False`.
- **prob_method** (*str, optional*) – The method to convert the outlier scores to probabilities. Two approaches are possible:

  1. `'linear'`: simply use min-max conversion to linearly transform the outlier scores into the range of [0,1]. The model must be fitted first.

  2. `'unify'`: use unifying scores, see [KKSZ11].

  Default: `'linear'`.
- **return_conf** (*boolean, optional*) – Whether to return the model's confidence in making the same prediction under slightly different training sets. See [PVD20]. Default: `False`.
- **return_emb** (*bool, optional*) – Whether to return the learned node representations. Default: `False`.

**Returns**

- **pred** (*torch.Tensor*) – The predicted binary outlier labels of shape $N$. 0 stands for inliers and 1 for outliers. Only available when `return_label=True`.
- **score** (*torch.Tensor*) – The raw outlier scores of shape $N$. Only available when `return_score=True`.
- **prob** (*torch.Tensor*) – The outlier probabilities of shape $N$. Only available when `return_prob=True`.
- **conf** (*torch.Tensor*) – The prediction confidence of shape $N$. Only available when `return_conf=True`.

## 5.9 GAAN

**class** pygod.detector.**GAAN**(*noise_dim=16*, *hid_dim=64*, *num_layers=4*, *dropout=0.0*, *weight_decay=0.0*, *act=<function relu>*, *backbone=None*, *contamination=0.1*, *lr=0.004*, *epoch=100*, *gpu=-1*, *batch_size=0*, *num_neigh=-1*, *weight=0.5*, *verbose=0*, *save_emb=False*, *compile_model=False*, ***kwargs*)

Bases: *DeepDetector*

Generative Adversarial Attributed Network Anomaly Detection

GAAN is a generative adversarial attribute network anomaly detection framework, including a generator module, an encoder module, a discriminator module, and uses anomaly evaluation measures that consider sample reconstruction error and real sample recognition confidence to make predictions. This model is transductive only.

See [CLW+20] for details.

**Parameters**

- **noise_dim** (*int, optional*) – Input dimension of the Gaussian random noise. Defaults: 16.
- **hid_dim** (*int, optional*) – Hidden dimension of model. Default: 64.
- **num_layers** (*int, optional*) – Total number of layers in model. A half (floor) of the layers are for the generator, the other half (ceil) of the layers are for encoder. Default: 4.

- **dropout** (*float, optional*) – Dropout rate. Default: `0.`.
- **weight_decay** (*float, optional*) – Weight decay (L2 penalty). Default: `0.`.
- **act** (*callable activation function or None, optional*) – Activation function if not None. Default: `torch.nn.functional.relu`.
- **backbone** (*torch.nn.Module*) – The backbone of GAAN is fixed to be MLP. Changing of this parameter will not affect the model. Default: `None`.
- **contamination** (*float, optional*) – The amount of contamination of the dataset in (0., 0.5], i.e., the proportion of outliers in the dataset. Used when fitting to define the threshold on the decision function. Default: `0.1`.
- **lr** (*float, optional*) – Learning rate. Default: `0.004`.
- **epoch** (*int, optional*) – Maximum number of training epoch. Default: `100`.
- **gpu** (*int*) – GPU Index, -1 for using CPU. Default: `-1`.
- **batch_size** (*int, optional*) – Minibatch size, 0 for full batch training. Default: `0`.
- **num_neigh** (*int, optional*) – Number of neighbors in sampling, -1 for all neighbors. Default: `-1`.
- **weight** (*float, optional*) – Weight between reconstruction of node feature and structure. Default: `0.5`.
- **verbose** (*int, optional*) – Verbosity mode. Range in [0, 3]. Larger value for printing out more log information. Default: `0`.
- **save_emb** (*bool, optional*) – Whether to save the embedding. Default: `False`.
- **compile_model** (*bool, optional*) – Whether to compile the model with `torch_geometric.compile`. Default: `False`.
- ****kwargs** – Other parameters for the backbone.

**decision_score_**

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

**Type**
torch.Tensor

**threshold_**

The threshold is based on `contamination`. It is the $N\times$ `contamination` most abnormal samples in `decision_score_`. The threshold is calculated for generating binary outlier labels.

**Type**
float

**label_**

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

**Type**
torch.Tensor

**emb**

The learned node hidden embeddings of shape $N\times$ `hid_dim`. Only available when `save_emb` is `True`. When the detector has not been fitted, `emb` is `None`. When the detector has multiple embeddings, `emb` is a tuple of torch.Tensor.

**Type**
torch.Tensor or tuple of torch.Tensor or None

**fit**(*data*, *label=None*)

Fit detector with training data.

**Parameters**

- **data** (*torch_geometric.data.Data*) – The training graph.
- **label** (*torch.Tensor, optional*) – The optional outlier ground truth labels used to monitor the training progress. They are not used to optimize the unsupervised model. Default: `None`.

**Returns**
**self** – Fitted detector.

**Return type**
object

**predict**(*data=None*, *label=None*, *return_pred=True*, *return_score=False*, *return_prob=False*, *prob_method='linear'*, *return_conf=False*, *return_emb=False*)

Prediction for testing data using the fitted detector. Return predicted labels by default.

**Parameters**

- **data** (*torch_geometric.data.Data, optional*) – The testing graph. If `None`, the training data is used. Default: `None`.
- **label** (*torch.Tensor, optional*) – The optional outlier ground truth labels used for testing. Default: `None`.
- **return_pred** (*bool, optional*) – Whether to return the predicted binary labels. The labels are determined by the outlier contamination on the raw outlier scores. Default: `True`.
- **return_score** (*bool, optional*) – Whether to return the raw outlier scores. Default: `False`.
- **return_prob** (*bool, optional*) – Whether to return the outlier probabilities. Default: `False`.
- **prob_method** (*str, optional*) – The method to convert the outlier scores to probabilities. Two approaches are possible:

  1. `'linear'`: simply use min-max conversion to linearly transform the outlier scores into the range of [0,1]. The model must be fitted first.

  2. `'unify'`: use unifying scores, see [KKSZ11].

  Default: `'linear'`.
- **return_conf** (*boolean, optional*) – Whether to return the model's confidence in making the same prediction under slightly different training sets. See [PVD20]. Default: `False`.
- **return_emb** (*bool, optional*) – Whether to return the learned node representations. Default: `False`.

**Returns**

- **pred** (*torch.Tensor*) – The predicted binary outlier labels of shape $N$. 0 stands for inliers and 1 for outliers. Only available when `return_label=True`.

- **score** (*torch.Tensor*) – The raw outlier scores of shape $N$. Only available when `return_score=True`.
- **prob** (*torch.Tensor*) – The outlier probabilities of shape $N$. Only available when `return_prob=True`.
- **conf** (*torch.Tensor*) – The prediction confidence of shape $N$. Only available when `return_conf=True`.

## 5.10 GADNR

**class** pygod.detector.**GADNR**(*hid_dim=64*, *num_layers=1*, *deg_dec_layers=4*, *fea_dec_layers=3*, *backbone=<class 'torch_geometric.nn.models.basic_gnn.GCN'>*, *sample_size=2*, *sample_time=3*, *neigh_loss='KL'*, *lambda_loss1=0.01*, *lambda_loss2=0.1*, *lambda_loss3=0.8*, *real_loss=True*, *lr=0.01*, *epoch=100*, *dropout=0.0*, *weight_decay=0.0003*, *act=<function relu>*, *gpu=-1*, *batch_size=0*, *num_neigh=-1*, *contamination=0.1*, *verbose=0*, *save_emb=False*, *compile_model=False*, ***kwargs*)

Bases: *DeepDetector*

Graph Anomaly Detection via Neighborhood Reconstruction

GAD-NR is a new type of GAE based on neighborhood reconstruction for graph anomaly detection. GAD-NR aims to reconstruct the entire neighborhood (including local structure, self attributes, and neighbors attributes) around a node based on the corresponding node representation.

See [RSL+24] for details.

**Parameters**

- **hid_dim** (*int, optional*) – Hidden dimension of model. Default: `64`.
- **num_layers** (*int, optional*) – Total number of layers in the backbone encoder model. Default: `1`.
- **deg_dec_layers** (*int, optional*) – The number of layers for the node degree decoder. Default: `4`.
- **fea_dec_layers** (*int, optional*) – The number of layers for the node feature decoder. Default: `3`.
- **backbone** (*torch.nn.Module, optional*) – The backbone of the deep detector implemented in PyG. Default: `torch_geometric.nn.GCN`.
- **sample_size** (*int, optional*) – The number of samples for the neighborhood distribution. Default: `2`.
- **sample_time** (*int, optional*) – The number sample times to remove the noise during node feature and neighborhood distribution reconstruction. Default: `3`.
- **neigh_loss** (*str, optional*) – The neighbor reconstruction loss. `KL` represents the KL divergence loss, `W2` represents the W2 loss. Default: `KL`.
- **lambda_loss1** (*float, optional*) – The weight of the neighborhood reconstruction loss term. Default: `1e-2`.
- **lambda_loss2** (*float, optional*) – The weight of the node feature reconstruction loss term. Default: `1e-3`.

- **lambda_loss3** (*float, optional*) – The weight of the node degree reconstruction loss term. Default: `1e-4`.
- **real_loss** (*bool, optional*) – Whether using the original loss proposed in the paper as the decision score, if not, using the proposed weighted decision score. Default: `True`.
- **lr** (*float, optional*) – Learning rate. Default: `0.01`.
- **epoch** (*int, optional*) – Maximum number of training epoch. Default: `100`.
- **dropout** (*float, optional*) – Dropout rate. Default: `0.`.
- **weight_decay** (*float, optional*) – Weight decay (L2 penalty). Default: `0.0003`.
- **act** (*callable activation function or None, optional*) – Activation function if not None. Default: `torch.nn.functional.relu`.
- **gpu** (*int*) – GPU Index, -1 for using CPU. Default: `-1`.
- **batch_size** (*int, optional*) – Minibatch size, 0 for full batch training. Default: `0`.
- **num_neigh** (*int, optional*) – Number of neighbors in sampling, -1 for all neighbors. Default: `-1`.
- **contamination** (*float, optional*) – The amount of contamination of the dataset in (0., 0.5], i.e., the proportion of outliers in the dataset. Used when fitting to define the threshold on the decision function. Default: `0.1`.
- **verbose** (*int, optional*) – Verbosity mode. Range in [0, 3]. Larger value for printing out more log information. Default: `0`.
- **save_emb** (*bool, optional*) – Whether to save the embedding. Default: `False`.
- **compile_model** (*bool, optional*) – Whether to compile the model with `torch_geometric.compile`. Default: `False`.
- ****kwargs** (*optional*) – Other parameters for the backbone.

**decision_score_**

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

**Type**
torch.Tensor

**threshold_**

The threshold is based on `contamination`. It is the $N\times$ `contamination` most abnormal samples in `decision_score_`. The threshold is calculated for generating binary outlier labels.

**Type**
float

**label_**

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

**Type**
torch.Tensor

**emb**

The learned node hidden embeddings of shape $N\times$ `hid_dim`. Only available when `save_emb` is `True`. When the detector has not been fitted, `emb` is `None`. When the detector has multiple embeddings, `emb` is a tuple of torch.Tensor.

**Type**
torch.Tensor or tuple of torch.Tensor or None

**fit**(*data*, *label=None*, *h_loss_weight=1.0*, *degree_loss_weight=0.0*, *feature_loss_weight=2.5*, *loss_step=20*)

Overwrite the base model fit function since GAD-NR uses multiple personalized loss functions.

**Parameters**

- **data** (*torch_geometric.data.Data*) – Input graph.
- **label** (*torch.Tensor, optional*) – The optional outlier ground truth labels used for testing. Default: `None`.
- **h_loss_weight** (*float, optional*) – The weight of the neighborhood reconstruction loss term used in the weighted decision score. Default: `1.0`.
- **degree_loss_weight** (*float, optional*) – The weight of the node degree reconstruction loss term used in the weighted decision score. Default: `0.`.
- **feature_loss_weight** (*float, optional*) – The weight of the node feature reconstruction loss term used in the weighted decision score. Default: `2.5`.
- **loss_step** (*int, optional*) – The epoch interval to update the loss terms. Default: `20`.

**predict**(*data=None*, *label=None*, *return_pred=True*, *return_score=False*, *return_prob=False*, *prob_method='linear'*, *return_conf=False*, *return_emb=False*)

Prediction for testing data using the fitted detector. Return predicted labels by default.

**Parameters**

- **data** (*torch_geometric.data.Data, optional*) – The testing graph. If `None`, the training data is used. Default: `None`.
- **label** (*torch.Tensor, optional*) – The optional outlier ground truth labels used for testing. Default: `None`.
- **return_pred** (*bool, optional*) – Whether to return the predicted binary labels. The labels are determined by the outlier contamination on the raw outlier scores. Default: `True`.
- **return_score** (*bool, optional*) – Whether to return the raw outlier scores. Default: `False`.
- **return_prob** (*bool, optional*) – Whether to return the outlier probabilities. Default: `False`.
- **prob_method** (*str, optional*) – The method to convert the outlier scores to probabilities. Two approaches are possible:

  1. `'linear'`: simply use min-max conversion to linearly transform the outlier scores into the range of [0,1]. The model must be fitted first.

  2. `'unify'`: use unifying scores, see [KKSZ11].

  Default: `'linear'`.
- **return_conf** (*boolean, optional*) – Whether to return the model's confidence in making the same prediction under slightly different training sets. See [PVD20]. Default: `False`.
- **return_emb** (*bool, optional*) – Whether to return the learned node representations. Default: `False`.

**Returns**

- **pred** (*torch.Tensor*) – The predicted binary outlier labels of shape $N$. 0 stands for inliers and 1 for outliers. Only available when `return_label=True`.
- **score** (*torch.Tensor*) – The raw outlier scores of shape $N$. Only available when `return_score=True`.
- **prob** (*torch.Tensor*) – The outlier probabilities of shape $N$. Only available when `return_prob=True`.
- **conf** (*torch.Tensor*) – The prediction confidence of shape $N$. Only available when `return_conf=True`.

## 5.11 GAE

**class** pygod.detector.**GAE**(*hid_dim=64*, *num_layers=4*, *dropout=0.0*, *weight_decay=0.0*, *act=<function relu>*, *backbone=<class 'torch_geometric.nn.models.basic_gnn.GCN'>*, *recon_s=False*, *sigmoid_s=False*, *contamination=0.1*, *lr=0.004*, *epoch=100*, *gpu=-1*, *batch_size=0*, *num_neigh=-1*, *verbose=False*, *save_emb=False*, *compile_model=False*, ***kwargs*)

Bases: *DeepDetector*

Graph Autoencoder

See [KW16] for details.

**Parameters**

- **hid_dim** (*int, optional*) – Hidden dimension of model. Default: `64`.
- **num_layers** (*int, optional*) – Total number of layers in model. Default: `4`.
- **dropout** (*float, optional*) – Dropout rate. Default: `0.`.
- **weight_decay** (*float, optional*) – Weight decay (L2 penalty). Default: `0.`.
- **act** (*callable activation function or None, optional*) – Activation function if not None. Default: `torch.nn.functional.relu`.
- **backbone** (*torch.nn.Module, optional*) – The backbone of the deep detector implemented in PyG. Default: `torch_geometric.nn.GCN`.
- **recon_s** (*bool, optional*) – Reconstruct the structure instead of node feature . Default: `False`.
- **sigmoid_s** (*bool, optional*) – Whether to use sigmoid function to scale the reconstructed structure. Default: `False`.
- **contamination** (*float, optional*) – The amount of contamination of the dataset in (0., 0.5], i.e., the proportion of outliers in the dataset. Used when fitting to define the threshold on the decision function. Default: `0.1`.
- **lr** (*float, optional*) – Learning rate. Default: `0.004`.
- **epoch** (*int, optional*) – Maximum number of training epoch. Default: `100`.
- **gpu** (*int*) – GPU Index, -1 for using CPU. Default: `-1`.
- **batch_size** (*int, optional*) – Minibatch size, 0 for full batch training. Default: `0`.
- **num_neigh** (*int, optional*) – Number of neighbors in sampling, -1 for all neighbors. Default: `-1`.

- **verbose** (*int, optional*) – Verbosity mode. Range in [0, 3]. Larger value for printing out more log information. Default: `0`.
- **save_emb** (*bool, optional*) – Whether to save the embedding. Default: `False`.
- **compile_model** (*bool, optional*) – Whether to compile the model with `torch_geometric.compile`. Default: `False`.
- ****kwargs** (*optional*) – Other parameters for the backbone.

**decision_score_**

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

**Type**
torch.Tensor

**threshold_**

The threshold is based on `contamination`. It is the $N\times$ `contamination` most abnormal samples in `decision_score_`. The threshold is calculated for generating binary outlier labels.

**Type**
float

**label_**

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

**Type**
torch.Tensor

**emb**

The learned node hidden embeddings of shape $N\times$ `hid_dim`. Only available when `save_emb` is `True`. When the detector has not been fitted, `emb` is `None`. When the detector has multiple embeddings, `emb` is a tuple of torch.Tensor.

**Type**
torch.Tensor or tuple of torch.Tensor or None

**fit**(*data*, *label=None*)

Fit detector with training data.

**Parameters**

- **data** (*torch_geometric.data.Data*) – The training graph.
- **label** (*torch.Tensor, optional*) – The optional outlier ground truth labels used to monitor the training progress. They are not used to optimize the unsupervised model. Default: `None`.

**Returns**
**self** – Fitted detector.

**Return type**
object

**predict**(*data=None*, *label=None*, *return_pred=True*, *return_score=False*, *return_prob=False*, *prob_method='linear'*, *return_conf=False*, *return_emb=False*)

Prediction for testing data using the fitted detector. Return predicted labels by default.

**Parameters**

- **data** (*torch_geometric.data.Data, optional*) – The testing graph. If `None`, the training data is used. Default: `None`.
- **label** (*torch.Tensor, optional*) – The optional outlier ground truth labels used for testing. Default: `None`.
- **return_pred** (*bool, optional*) – Whether to return the predicted binary labels. The labels are determined by the outlier contamination on the raw outlier scores. Default: `True`.
- **return_score** (*bool, optional*) – Whether to return the raw outlier scores. Default: `False`.
- **return_prob** (*bool, optional*) – Whether to return the outlier probabilities. Default: `False`.
- **prob_method** (*str, optional*) – The method to convert the outlier scores to probabilities. Two approaches are possible:

  1. `'linear'`: simply use min-max conversion to linearly transform the outlier scores into the range of [0,1]. The model must be fitted first.

  2. `'unify'`: use unifying scores, see [KKSZ11].

  Default: `'linear'`.
- **return_conf** (*boolean, optional*) – Whether to return the model's confidence in making the same prediction under slightly different training sets. See [PVD20]. Default: `False`.
- **return_emb** (*bool, optional*) – Whether to return the learned node representations. Default: `False`.

**Returns**

- **pred** (*torch.Tensor*) – The predicted binary outlier labels of shape $N$. 0 stands for inliers and 1 for outliers. Only available when `return_label=True`.
- **score** (*torch.Tensor*) – The raw outlier scores of shape $N$. Only available when `return_score=True`.
- **prob** (*torch.Tensor*) – The outlier probabilities of shape $N$. Only available when `return_prob=True`.
- **conf** (*torch.Tensor*) – The prediction confidence of shape $N$. Only available when `return_conf=True`.

## 5.12 GUIDE

**class** pygod.detector.**GUIDE**(*hid_a=64*, *hid_s=4*, *num_layers=4*, *dropout=0.0*, *weight_decay=0.0*, *act=<function relu>*, *backbone=None*, *alpha=0.5*, *contamination=0.1*, *lr=0.004*, *epoch=100*, *gpu=-1*, *batch_size=0*, *num_neigh=-1*, *graphlet_size=4*, *selected_motif=True*, *cache_dir=None*, *verbose=0*, *save_emb=False*, *compile_model=False*, ***kwargs*)

Bases: *DeepDetector*

Higher-order Structure based Anomaly Detection on Attributed Networks

GUIDE is an anomaly detector consisting of an attribute graph convolutional autoencoder, and a structure graph attentive autoencoder (not the same as the graph attention networks). Instead of the adjacency matrix, node motif

degree is used as input of structure autoencoder. The reconstruction mean square error of the autoencoders are defined as structure anomaly score and attribute anomaly score, respectively.

Note: The calculation of node motif degree in preprocessing has high time complexity. It may take longer than you expect.

See [YZY+21] for details.

**Parameters**

- **hid_a** (*int, optional*) – Hidden dimension for attribute. Default: `64`.
- **hid_s** (*int, optional*) – Hidden dimension for structure. Default: `4`.
- **num_layers** (*int, optional*) – Total number of layers in model. Default: `4`.
- **dropout** (*float, optional*) – Dropout rate. Default: `0.`.
- **weight_decay** (*float, optional*) – Weight decay (L2 penalty). Default: `0.`.
- **act** (*callable activation function or None, optional*) – Activation function if not None. Default: `torch.nn.functional.relu`.
- **backbone** (*torch.nn.Module*) – The backbone of GUIDE is fixed. Changing of this parameter will not affect the model. Default: `None`.
- **alpha** (*float, optional*) – Weight between reconstruction of node feature and structure. Default: `0.5`.
- **contamination** (*float, optional*) – The amount of contamination of the dataset in (0., 0.5], i.e., the proportion of outliers in the dataset. Used when fitting to define the threshold on the decision function. Default: `0.1`.
- **lr** (*float, optional*) – Learning rate. Default: `0.004`.
- **epoch** (*int, optional*) – Maximum number of training epoch. Default: `100`.
- **gpu** (*int*) – GPU Index, -1 for using CPU. Default: `-1`.
- **batch_size** (*int, optional*) – Minibatch size, 0 for full batch training. Default: `0`.
- **num_neigh** (*int, optional*) – Number of neighbors in sampling, -1 for all neighbors. Default: `-1`.
- **graphlet_size** (*int, optional*) – The maximum size of graphlet. Default: `4`.
- **selected_motif** (*bool, optional*) – Whether to use selected motif in the paper. Default: `True`.
- **cache_dir** (*str, optional*) – The directory for the node motif degree caching. If `None`, ~/.pygod will be used. Default: `None`.
- **verbose** (*int, optional*) – Verbosity mode. Range in [0, 3]. Larger value for printing out more log information. Default: `0`.
- **save_emb** (*bool, optional*) – Whether to save the embedding. Default: `False`.
- **compile_model** (*bool, optional*) – Whether to compile the model with `torch_geometric.compile`. Default: `False`.
- ****kwargs** – Other parameters for the backbone.

`decision_score_`

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

**Type**
torch.Tensor

`threshold_`

The threshold is based on `contamination`. It is the $N\times$ `contamination` most abnormal samples in `decision_score_`. The threshold is calculated for generating binary outlier labels.

**Type**
float

`label_`

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

**Type**
torch.Tensor

`emb`

The learned node hidden embeddings of shape $N\times$ `hid_dim`. Only available when `save_emb` is `True`. When the detector has not been fitted, `emb` is `None`. When the detector has multiple embeddings, `emb` is a tuple of torch.Tensor.

**Type**
torch.Tensor or tuple of torch.Tensor or None

`fit`(*data*, *label=None*)

Fit detector with training data.

**Parameters**

- **data** (*torch_geometric.data.Data*) – The training graph.
- **label** (*torch.Tensor, optional*) – The optional outlier ground truth labels used to monitor the training progress. They are not used to optimize the unsupervised model. Default: `None`.

**Returns**
**self** – Fitted detector.

**Return type**
object

`predict`(*data=None*, *label=None*, *return_pred=True*, *return_score=False*, *return_prob=False*, *prob_method='linear'*, *return_conf=False*, *return_emb=False*)

Prediction for testing data using the fitted detector. Return predicted labels by default.

**Parameters**

- **data** (*torch_geometric.data.Data, optional*) – The testing graph. If `None`, the training data is used. Default: `None`.
- **label** (*torch.Tensor, optional*) – The optional outlier ground truth labels used for testing. Default: `None`.
- **return_pred** (*bool, optional*) – Whether to return the predicted binary labels. The labels are determined by the outlier contamination on the raw outlier scores. Default: `True`.
- **return_score** (*bool, optional*) – Whether to return the raw outlier scores. Default: `False`.
- **return_prob** (*bool, optional*) – Whether to return the outlier probabilities. Default: `False`.

- **prob_method** (*str, optional*) – The method to convert the outlier scores to probabilities. Two approaches are possible:

  1. `'linear'`: simply use min-max conversion to linearly transform the outlier scores into the range of [0,1]. The model must be fitted first.

  2. `'unify'`: use unifying scores, see [KKSZ11].

  Default: `'linear'`.

- **return_conf** (*boolean, optional*) – Whether to return the model's confidence in making the same prediction under slightly different training sets. See [PVD20]. Default: `False`.

- **return_emb** (*bool, optional*) – Whether to return the learned node representations. Default: `False`.

**Returns**

- **pred** (*torch.Tensor*) – The predicted binary outlier labels of shape $N$. 0 stands for inliers and 1 for outliers. Only available when `return_label=True`.

- **score** (*torch.Tensor*) – The raw outlier scores of shape $N$. Only available when `return_score=True`.

- **prob** (*torch.Tensor*) – The outlier probabilities of shape $N$. Only available when `return_prob=True`.

- **conf** (*torch.Tensor*) – The prediction confidence of shape $N$. Only available when `return_conf=True`.

## 5.13 OCGNN

**class** pygod.detector.**OCGNN**(*hid_dim=64*, *num_layers=2*, *dropout=0.0*, *weight_decay=0.0*, *act=<function relu>*, *backbone=<class 'torch_geometric.nn.models.basic_gnn.GCN'>*, *contamination=0.1*, *lr=0.004*, *epoch=100*, *gpu=-1*, *batch_size=0*, *num_neigh=-1*, *beta=0.5*, *warmup=2*, *eps=0.001*, *verbose=0*, *save_emb=False*, *compile_model=False*, ***kwargs*)

Bases: *DeepDetector*

One-Class Graph Neural Networks for Anomaly Detection in Attributed Networks

OCGNN is an anomaly detector that measures the distance of anomaly to the centroid, in a similar fashion to the support vector machine, but in the embedding space after feeding towards several layers of GCN.

See [WJD+21] for details.

**Parameters**

- **hid_dim** (*int, optional*) – Hidden dimension of model. Default: `64`.

- **num_layers** (*int, optional*) – Total number of layers in model. Default: `2`.

- **dropout** (*float, optional*) – Dropout rate. Default: `0.`.

- **weight_decay** (*float, optional*) – Weight decay (L2 penalty). Default: `0.`.

- **act** (*callable activation function or None, optional*) – Activation function if not None. Default: `torch.nn.functional.relu`.

- **backbone** (*torch.nn.Module*) – The backbone of the deep detector implemented in PyG. Default: `torch_geometric.nn.GCN`.

- **contamination** (*float, optional*) – The amount of contamination of the dataset in (0., 0.5], i.e., the proportion of outliers in the dataset. Used when fitting to define the threshold on the decision function. Default: `0.1`.
- **lr** (*float, optional*) – Learning rate. Default: `0.004`.
- **epoch** (*int, optional*) – Maximum number of training epoch. Default: `100`.
- **gpu** (*int*) – GPU Index, -1 for using CPU. Default: `-1`.
- **batch_size** (*int, optional*) – Minibatch size, 0 for full batch training. Default: `0`.
- **num_neigh** (*int, optional*) – Number of neighbors in sampling, -1 for all neighbors. Default: `-1`.
- **beta** (*float, optional*) – The weight between the reconstruction loss and radius. Default: `0.5`.
- **warmup** (*int, optional*) – The number of epochs for warm-up training. Default: `2`.
- **eps** (*float, optional*) – The slack variable. Default: `0.001`.
- **verbose** (*int, optional*) – Verbosity mode. Range in [0, 3]. Larger value for printing out more log information. Default: `0`.
- **save_emb** (*bool, optional*) – Whether to save the embedding. Default: `False`.
- **compile_model** (*bool, optional*) – Whether to compile the model with `torch_geometric.compile`. Default: `False`.
- ****kwargs** – Other parameters for the backbone model.

**decision_score_**

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

**Type**
torch.Tensor

**threshold_**

The threshold is based on `contamination`. It is the $N\times$ `contamination` most abnormal samples in `decision_score_`. The threshold is calculated for generating binary outlier labels.

**Type**
float

**label_**

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

**Type**
torch.Tensor

**emb**

The learned node hidden embeddings of shape $N\times$ `hid_dim`. Only available when `save_emb` is `True`. When the detector has not been fitted, `emb` is `None`. When the detector has multiple embeddings, `emb` is a tuple of torch.Tensor.

**Type**
torch.Tensor or tuple of torch.Tensor or None

**fit**(*data*, *label=None*)

Fit detector with training data.

**Parameters**

- **data** (*torch_geometric.data.Data*) – The training graph.
- **label** (*torch.Tensor, optional*) – The optional outlier ground truth labels used to monitor the training progress. They are not used to optimize the unsupervised model. Default: `None`.

**Returns**

**self** – Fitted detector.

**Return type**

object

**predict**(*data=None*, *label=None*, *return_pred=True*, *return_score=False*, *return_prob=False*, *prob_method='linear'*, *return_conf=False*, *return_emb=False*)

Prediction for testing data using the fitted detector. Return predicted labels by default.

**Parameters**

- **data** (*torch_geometric.data.Data, optional*) – The testing graph. If `None`, the training data is used. Default: `None`.
- **label** (*torch.Tensor, optional*) – The optional outlier ground truth labels used for testing. Default: `None`.
- **return_pred** (*bool, optional*) – Whether to return the predicted binary labels. The labels are determined by the outlier contamination on the raw outlier scores. Default: `True`.
- **return_score** (*bool, optional*) – Whether to return the raw outlier scores. Default: `False`.
- **return_prob** (*bool, optional*) – Whether to return the outlier probabilities. Default: `False`.
- **prob_method** (*str, optional*) – The method to convert the outlier scores to probabilities. Two approaches are possible:

  1. `'linear'`: simply use min-max conversion to linearly transform the outlier scores into the range of [0,1]. The model must be fitted first.

  2. `'unify'`: use unifying scores, see [KKSZ11].

  Default: `'linear'`.
- **return_conf** (*boolean, optional*) – Whether to return the model's confidence in making the same prediction under slightly different training sets. See [PVD20]. Default: `False`.
- **return_emb** (*bool, optional*) – Whether to return the learned node representations. Default: `False`.

**Returns**

- **pred** (*torch.Tensor*) – The predicted binary outlier labels of shape $N$. 0 stands for inliers and 1 for outliers. Only available when `return_label=True`.
- **score** (*torch.Tensor*) – The raw outlier scores of shape $N$. Only available when `return_score=True`.

- **prob** (*torch.Tensor*) – The outlier probabilities of shape $N$. Only available when `return_prob=True`.
- **conf** (*torch.Tensor*) – The prediction confidence of shape $N$. Only available when `return_conf=True`.

## 5.14 ONE

**class** pygod.detector.**ONE**(*hid_a=36*, *hid_s=36*, *alpha=1.0*, *beta=1.0*, *gamma=1.0*, *weight_decay=0.0*, *contamination=0.1*, *lr=0.004*, *epoch=5*, *gpu=-1*, *verbose=0*)

Bases: *Detector*

Outlier Aware Network Embedding for Attributed Networks

**Note:** This detector is transductive only. Using `predict` with unseen data will train the detector from scratch.

See [BLM19] for details.

**Parameters**

- **hid_a** (*int, optional*) – Hidden dimension for the attribute. Default: `36`.
- **hid_s** (*int, optional*) – Hidden dimension for the structure. Default: `36`.
- **alpha** (*float, optional*) – Weight for the attribute loss. Default: `1.`.
- **beta** (*float, optional*) – Weight for the structural loss. Default: `1.`.
- **gamma** (*float, optional*) – Weight for the combined loss. Default: `1.`.
- **weight_decay** (*float, optional*) – Weight decay (L2 penalty). Default: `0.`.
- **contamination** (*float, optional*) – Valid in (0., 0.5). The proportion of outliers in the data set. Used when fitting to define the threshold on the decision function. Default: `0.1`.
- **lr** (*float, optional*) – Learning rate. Default: `0.004`.
- **epoch** (*int, optional*) – Maximum number of training epoch. Default: `5`.
- **gpu** (*int*) – GPU Index, -1 for using CPU. Default: `-1`.
- **verbose** (*int, optional*) – Verbosity mode. Range in [0, 3]. Larger value for printing out more log information. Default: `0`.

**fit**(*data*, *label=None*)

Fit detector with training data.

**Parameters**

- **data** (*torch_geometric.data.Data*) – The training graph.
- **label** (*torch.Tensor, optional*) – The optional outlier ground truth labels used to monitor the training progress. They are not used to optimize the unsupervised model. Default: `None`.

**Returns**

**self** – Fitted detector.

**Return type**

object

**predict**(*data=None*, *label=None*, *return_pred=True*, *return_score=False*, *return_prob=False*, *prob_method='linear'*, *return_conf=False*)

Prediction for testing data using the fitted detector. Return predicted labels by default.

**Parameters**

- **data** (*torch_geometric.data.Data, optional*) – The testing graph. If `None`, the training data is used. Default: `None`.
- **label** (*torch.Tensor, optional*) – The optional outlier ground truth labels used for testing. Default: `None`.
- **return_pred** (*bool, optional*) – Whether to return the predicted binary labels. The labels are determined by the outlier contamination on the raw outlier scores. Default: `True`.
- **return_score** (*bool, optional*) – Whether to return the raw outlier scores. Default: `False`.
- **return_prob** (*bool, optional*) – Whether to return the outlier probabilities. Default: `False`.
- **prob_method** (*str, optional*) – The method to convert the outlier scores to probabilities. Two approaches are possible:

  1. `'linear'`: simply use min-max conversion to linearly transform the outlier scores into the range of [0,1]. The model must be fitted first.

  2. `'unify'`: use unifying scores, see [KKSZ11].

  Default: `'linear'`.
- **return_conf**(*boolean, optional*) – Whether to return the model's confidence in making the same prediction under slightly different training sets. See [PVD20]. Default: `False`.

**Returns**

- **pred** (*torch.Tensor*) – The predicted binary outlier labels of shape $N$. 0 stands for inliers and 1 for outliers. Only available when `return_label=True`.
- **score** (*torch.Tensor*) – The raw outlier scores of shape $N$. Only available when `return_score=True`.
- **prob** (*torch.Tensor*) – The outlier probabilities of shape $N$. Only available when `return_prob=True`.
- **conf** (*torch.Tensor*) – The prediction confidence of shape $N$. Only available when `return_conf=True`.

## 5.15 Radar

**class** pygod.detector.**Radar**(*gamma=1.0*, *weight_decay=0.0*, *lr=0.004*, *epoch=100*, *gpu=-1*, *contamination=0.1*, *verbose=0*)

Bases: *Detector*

Residual Analysis for Anomaly Detection in Attributed Networks

Radar is an anomaly detector with residual analysis.

**Note:** This detector is transductive only. Using `predict` with unseen data will train the detector from scratch.

See [LDHL17] for details.

**Parameters**

- **gamma** (*float, optional*) – Weight of node feature w.r.t. structure. Default: `1`.
- **weight_decay** (*float, optional*) – Weight decay (L2 penalty). Default: `0.`.
- **lr** (*float, optional*) – Learning rate. Default: `0.004`.
- **epoch** (*int, optional*) – Maximum number of training epoch. Default: `100`.
- **gpu** (*int*) – GPU Index, -1 for using CPU. Default: `-1`.
- **contamination** (*float, optional*) – Valid in (0., 0.5). The proportion of outliers in the data set. Used when fitting to define the threshold on the decision function. Default: `0.1`.
- **verbose** (*int, optional*) – Verbosity mode. Range in [0, 3]. Larger value for printing out more log information. Default: `0`.

**fit**(*data*, *label=None*)

Fit detector with training data.

**Parameters**

- **data** (*torch_geometric.data.Data*) – The training graph.
- **label** (*torch.Tensor, optional*) – The optional outlier ground truth labels used to monitor the training progress. They are not used to optimize the unsupervised model. Default: `None`.

**Returns**

**self** – Fitted detector.

**Return type**

object

**predict**(*data=None*, *label=None*, *return_pred=True*, *return_score=False*, *return_prob=False*, *prob_method='linear'*, *return_conf=False*)

Prediction for testing data using the fitted detector. Return predicted labels by default.

**Parameters**

- **data** (*torch_geometric.data.Data, optional*) – The testing graph. If `None`, the training data is used. Default: `None`.
- **label** (*torch.Tensor, optional*) – The optional outlier ground truth labels used for testing. Default: `None`.
- **return_pred** (*bool, optional*) – Whether to return the predicted binary labels. The labels are determined by the outlier contamination on the raw outlier scores. Default: `True`.
- **return_score** (*bool, optional*) – Whether to return the raw outlier scores. Default: `False`.
- **return_prob** (*bool, optional*) – Whether to return the outlier probabilities. Default: `False`.
- **prob_method** (*str, optional*) – The method to convert the outlier scores to probabilities. Two approaches are possible:

1. 'linear': simply use min-max conversion to linearly transform the outlier scores into the range of [0,1]. The model must be fitted first.

2. 'unify': use unifying scores, see [KKSZ11].

Default: 'linear'.

- **return_conf** (*boolean, optional*) – Whether to return the model's confidence in making the same prediction under slightly different training sets. See [PVD20]. Default: False.

**Returns**

- **pred** (*torch.Tensor*) – The predicted binary outlier labels of shape $N$. 0 stands for inliers and 1 for outliers. Only available when return_label=True.
- **score** (*torch.Tensor*) – The raw outlier scores of shape $N$. Only available when return_score=True.
- **prob** (*torch.Tensor*) – The outlier probabilities of shape $N$. Only available when return_prob=True.
- **conf** (*torch.Tensor*) – The prediction confidence of shape $N$. Only available when return_conf=True.

## 5.16 SCAN

**class** pygod.detector.**SCAN**(*eps=0.5*, *mu=2*, *contamination=0.1*, *verbose=0*)

Bases: *Detector*

Structural Clustering Algorithm for Networks

SCAN is a clustering algorithm, which only takes the graph structure without the node features as the input. Note: This model will output detected clusters instead of "outliers" descibed in the original paper.

---

**Note:** This detector is transductive only. Using predict with unseen data will train the detector from scratch.

---

See [XYFS07] for details.

**Parameters**

- **eps** (*float, optional*) – Neighborhood threshold. Default: .5.
- **mu** (*int, optional*) – Minimal size of clusters. Default: 2.
- **contamination** (*float, optional*) – Valid in (0., 0.5). The proportion of outliers in the data set. Used when fitting to define the threshold on the decision function. Default: 0.1.
- **verbose** (*int, optional*) – Verbosity mode. Range in [0, 3]. Larger value for printing out more log information. Default: 0.

**decision_score_**

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**
torch.Tensor

**threshold_**

The threshold is based on `contamination`. It is the $N\times$ `contamination` most abnormal samples in `decision_score_`. The threshold is calculated for generating binary outlier labels.

**Type**
float

**label_**

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

**Type**
torch.Tensor

**hub_score_**

The binary hub scores of each node.

**Type**
torch.Tensor

**scatter_score_**

The binary scatter scores of each node, i.e., the "outlier" scores in the original paper.

**Type**
torch.Tensor

**fit**(*data*, *label=None*)

Fit detector with training data.

**Parameters**

- **data** (*torch_geometric.data.Data*) – The training graph.
- **label** (*torch.Tensor, optional*) – The optional outlier ground truth labels used to monitor the training progress. They are not used to optimize the unsupervised model. Default: `None`.

**Returns**
**self** – Fitted detector.

**Return type**
object

**predict**(*data=None*, *label=None*, *return_pred=True*, *return_score=False*, *return_prob=False*, *prob_method='linear'*, *return_conf=False*)

Prediction for testing data using the fitted detector. Return predicted labels by default.

**Parameters**

- **data** (*torch_geometric.data.Data, optional*) – The testing graph. If `None`, the training data is used. Default: `None`.
- **label** (*torch.Tensor, optional*) – The optional outlier ground truth labels used for testing. Default: `None`.
- **return_pred** (*bool, optional*) – Whether to return the predicted binary labels. The labels are determined by the outlier contamination on the raw outlier scores. Default: `True`.
- **return_score** (*bool, optional*) – Whether to return the raw outlier scores. Default: `False`.
- **return_prob** (*bool, optional*) – Whether to return the outlier probabilities. Default: `False`.

- **prob_method** (*str, optional*) – The method to convert the outlier scores to probabilities. Two approaches are possible:

  1. `'linear'`: simply use min-max conversion to linearly transform the outlier scores into the range of [0,1]. The model must be fitted first.

  2. `'unify'`: use unifying scores, see [KKSZ11].

  Default: `'linear'`.

- **return_conf** (*boolean, optional*) – Whether to return the model's confidence in making the same prediction under slightly different training sets. See [PVD20]. Default: `False`.

**Returns**

- **pred** (*torch.Tensor*) – The predicted binary outlier labels of shape $N$. 0 stands for inliers and 1 for outliers. Only available when `return_label=True`.
- **score** (*torch.Tensor*) – The raw outlier scores of shape $N$. Only available when `return_score=True`.
- **prob** (*torch.Tensor*) – The outlier probabilities of shape $N$. Only available when `return_prob=True`.
- **conf** (*torch.Tensor*) – The prediction confidence of shape $N$. Only available when `return_conf=True`.

CHAPTER

# SIX

# PYGOD.GENERATOR

pygod.generator.**gen_contextual_outlier**(*data*, *n*, *k*, *seed=None*)

Generating contextual outliers according to paper [DLBL19]. We randomly select `n` nodes as the attribute perturbation candidates. For each selected node $i$, we randomly pick another `k` nodes from the data and select the node $j$ whose attributes $x_j$ deviate the most from node $i$'s attribute $x_i$ among `k` nodes by maximizing the Euclidean distance $\|x_i x_j\|$. Afterwards, we then substitute the attributes $x_i$ of node $i$ to $x_j$.

**Parameters**

- **data** (*torch_geometric.data.Data*) – The input data.
- **n** (*int*) – Number of nodes converting to outliers.
- **k** (*int*) – Number of candidate nodes for each outlier node.
- **seed** (*int, optional*) – The seed to control the randomness, Default: `None`.

**Returns**

- **data** (*torch_geometric.data.Data*) – The contextual outlier graph with modified node attributes.
- **y_outlier** (*torch.Tensor*) – The outlier label tensor where 1 represents outliers and 0 represents normal nodes.

pygod.generator.**gen_structural_outlier**(*data*, *m*, *n*, *p=0*, *directed=False*, *seed=None*)

Generating structural outliers according to paper : cite:*ding2019deep*. We randomly select `m` nodes from the network and then make those nodes fully connected, and then all the `m` nodes in the clique are regarded as outliers. We iteratively repeat this process until a number of `n` cliques are generated and the total number of structural outliers is `m * n`.

**Parameters**

- **data** (*torch_geometric.data.Data*) – The input data.
- **m** (*int*) – Number nodes in the outlier cliques.
- **n** (*int*) – Number of outlier cliques.
- **p** (*int, optional*) – Probability of edge drop in cliques. Default: `0`.
- **directed** (*bool, optional*) – Whether the edges added are directed. Default: `False`.
- **seed** (*int, optional*) – The seed to control the randomness, Default: `None`.

**Returns**

- **data** (*torch_geometric.data.Data*) – The structural outlier graph with injected edges.
- **y_outlier** (*torch.Tensor*) – The outlier label tensor where 1 represents outliers and 0 represents normal nodes.

CHAPTER

# SEVEN

# PYGOD.METRIC

pygod.metric.**eval_average_precision**(*label*, *score*)

Average precision score for binary classification.

**Parameters**

- **label** (*torch.Tensor*) – Labels in shape of `(N, )`, where 1 represents outliers, 0 represents normal instances.
- **score** (*torch.Tensor*) – Outlier scores in shape of `(N, )`.

**Returns**

**ap** – Average precision score.

**Return type**

float

pygod.metric.**eval_f1**(*label*, *pred*)

F1 score for binary classification.

**Parameters**

- **label** (*torch.Tensor*) – Labels in shape of `(N, )`, where 1 represents outliers, 0 represents normal instances.
- **pred** (*torch.Tensor*) – Outlier prediction in shape of `(N, )`.

**Returns**

**f1** – F1 score.

**Return type**

float

pygod.metric.**eval_precision_at_k**(*label*, *score*, *k=None*)

Precision score for top k instances with the highest outlier scores.

**Parameters**

- **label** (*torch.Tensor*) – Labels in shape of `(N, )`, where 1 represents outliers, 0 represents normal instances.
- **score** (*torch.Tensor*) – Outlier scores in shape of `(N, )`.
- **k** (*int, optional*) – The number of instances to evaluate. `None` for precision. Default: `None`.

**Returns**

**precision_at_k** – Precision for top k instances with the highest outlier scores.

**Return type**
float

pygod.metric.**eval_recall_at_k**(*label*, *score*, *k=None*)

Recall score for top k instances with the highest outlier scores.

**Parameters**

- **label** (*torch.Tensor*) – Labels in shape of `(N, )`, where 1 represents outliers, 0 represents normal instances.
- **score** (*torch.Tensor*) – Outlier scores in shape of `(N, )`.
- **k** (*int, optional*) – The number of instances to evaluate. `None` for recall. Default: `None`.

**Returns**
**recall_at_k** – Recall for top k instances with the highest outlier scores.

**Return type**
float

pygod.metric.**eval_roc_auc**(*label*, *score*)

ROC-AUC score for binary classification.

**Parameters**

- **label** (*torch.Tensor*) – Labels in shape of `(N, )`, where 1 represents outliers, 0 represents normal instances.
- **score** (*torch.Tensor*) – Outlier scores in shape of `(N, )`.

**Returns**
**roc_auc** – Average ROC-AUC score across different labels.

**Return type**
float

CHAPTER

# EIGHT

# PYGOD.NN

| | |
|---|---|
| *AdONEBase* | Adversarial Outlier Aware Attributed Network Embedding |
| *AnomalyDAEBase* | Dual Autoencoder for Anomaly Detection on Attributed Networks |
| *CoLABase* | Anomaly Detection on Attributed Networks via Contrastive Self-Supervised Learning |
| *DMGDBase* | Deep Multiclass Graph Description |
| *DOMINANTBase* | Deep Anomaly Detection on Attributed Networks |
| *DONEBase* | Deep Outlier Aware Attributed Network Embedding |
| *GAANBase* | Generative Adversarial Attributed Network Anomaly Detection |
| *GADNRBase* | Graph Anomaly Detection via Neighborhood Reconstruction |
| *GAEBase* | Graph Autoencoder |
| *GUIDEBase* | Higher-order Structure based Anomaly Detection on Attributed Networks |
| *OCGNNBase* | One-Class Graph Neural Networks for Anomaly Detection in Attributed Networks |

## 8.1 AdONEBase

**class** pygod.nn.**AdONEBase**(*x_dim*, *s_dim*, *hid_dim=64*, *num_layers=4*, *dropout=0.0*, *act=<function relu>*, *w1=0.2*, *w2=0.2*, *w3=0.2*, *w4=0.2*, *w5=0.2*, ***kwargs*)

Bases: `Module`

Adversarial Outlier Aware Attributed Network Embedding

AdONE consists of an attribute autoencoder and a structure autoencoder. It estimates five loss to optimize the model, including an attribute proximity loss, an attribute homophily loss, a structure proximity loss, a structure homophily loss, and an alignment loss. It calculates three outlier scores, and averages them as an overall score. This model is transductive only.

See [BVM20] for details.

**Parameters**

- **x_dim** (*int*) – Input dimension of attribute.
- **s_dim** (*int*) – Input dimension of structure.
- **hid_dim** (*int, optional*) – Hidden dimension of model. Default: `64`.

- **num_layers** (*int, optional*) – Total number of layers in model. A half (floor) of the layers are for the encoder, the other half (ceil) of the layers are for decoders. Default: `4`.
- **dropout** (*float, optional*) – Dropout rate. Default: `0.`.
- **weight_decay** (*float, optional*) – Weight decay (L2 penalty). Default: `0.`.
- **act** (*callable activation function or None, optional*) – Activation function if not None. Default: `torch.nn.functional.relu`.
- **w1** (*float, optional*) – Weight of structure proximity loss. Default: `0.2`.
- **w2** (*float, optional*) – Weight of structure homophily loss. Default: `0.2`.
- **w3** (*float, optional*) – Weight of attribute proximity loss. Default: `0.2`.
- **w4** (*float, optional*) – Weight of attribute homophily loss. Default: `0.2`.
- **w5** (*float, optional*) – Weight of alignment loss. Default: `0.2`.
- ****kwargs** – Other parameters for the backbone.

**forward**(*x*, *s*, *edge_index*)

Forward computation.

**Parameters**

- **x** (*torch.Tensor*) – Input attribute embeddings.
- **s** (*torch.Tensor*) – Input structure embeddings.
- **edge_index** (*torch.Tensor*) – Edge index.

**Returns**

- **x_** (*torch.Tensor*) – Reconstructed attribute embeddings.
- **s_** (*torch.Tensor*) – Reconstructed structure embeddings.
- **h_a** (*torch.Tensor*) – Attribute hidden embeddings.
- **h_s** (*torch.Tensor*) – Structure hidden embeddings.
- **dna** (*torch.Tensor*) – Attribute neighbor distance.
- **dns** (*torch.Tensor*) – Structure neighbor distance.
- **dis_a** (*torch.Tensor*) – Attribute discriminator score.
- **dis_s** (*torch.Tensor*) – Structure discriminator score.

**static process_graph**(*data*)

Obtain the dense adjacency matrix of the graph.

**Parameters**

**data** (*torch_geometric.data.Data*) – Input graph.

## 8.2 AnomalyDAEBase

**class** pygod.nn.**AnomalyDAEBase**(*in_dim*, *num_nodes*, *emb_dim=64*, *hid_dim=64*, *dropout=0.0*, *act=<function relu>*, ***kwargs*)

Bases: Module

Dual Autoencoder for Anomaly Detection on Attributed Networks

AnomalyDAE is an anomaly detector that consists of a structure autoencoder and an attribute autoencoder to learn both node embedding and attribute embedding jointly in latent space. The structural autoencoder uses Graph Attention layers. The reconstruction mean square error of the decoders are defined as structure anomaly score and attribute anomaly score, respectively, with two additional penalties on the reconstructed adj matrix and node attributes (force entries to be nonzero).

See [FZL20] for details.

**Parameters**

- **in_dim** (*int*) – Input dimension of model.
- **num_nodes** (*int*) – Number of input nodes or batch size in minibatch training.
- **emb_dim:** – int: Embedding dimension of model. Default: 64.
- **hid_dim** (*int*) – Hidden dimension of model. Default: 64.
- **dropout** (*float, optional*) – Dropout rate. Default: 0..
- **act** (*callable activation function or None, optional*) – Activation function if not None. Default: torch.nn.functional.relu.
- ****kwargs** (*optional*) – Other parameters of torch_geometric.nn.GATConv.

**forward**(*x*, *edge_index*, *batch_size*)

Forward computation.

**Parameters**

- **x** (*torch.Tensor*) – Input attribute embeddings.
- **edge_index** (*torch.Tensor*) – Edge index.
- **batch_size** (*int*) – Batch size.

**Returns**

- **x_** (*torch.Tensor*) – Reconstructed attribute embeddings.
- **s_** (*torch.Tensor*) – Reconstructed adjacency matrix.

**static process_graph**(*data*)

Obtain the dense adjacency matrix of the graph.

**Parameters**

**data** (*torch_geometric.data.Data*) – Input graph.

## 8.3 CoLABase

**class** pygod.nn.**CoLABase**(*in_dim*, *hid_dim=64*, *num_layers=4*, *dropout=0.0*, *act=<function relu>*, *backbone=<class 'torch_geometric.nn.models.basic_gnn.GCN'>*, ***kwargs*)

Bases: Module

Anomaly Detection on Attributed Networks via Contrastive Self-Supervised Learning

CoLA is a contrastive self-supervised learning based method for graph anomaly detection. This implementation is base on random neighbor sampling instead of random walk sampling in the original paper.

See [LLP+21] for details.

**Parameters**

- **in_dim** (*int*) – Input dimension of model.
- **hid_dim** (*int, optional*) – Hidden dimension of model. Default: 64.
- **num_layers** (*int, optional*) – Total number of layers in model. Default: 4.
- **dropout** (*float, optional*) – Dropout rate. Default: 0..
- **act** (*callable activation function or None, optional*) – Activation function if not None. Default: torch.nn.functional.relu.
- **backbone** (*torch.nn.Module*) – The backbone of the deep detector implemented in PyG. Default: torch_geometric.nn.GCN.
- ****kwargs** – Other parameters for the backbone.

**forward**(*x*, *edge_index*)

Forward computation.

**Parameters**

- **x** (*torch.Tensor*) – Input attribute embeddings.
- **edge_index** (*torch.Tensor*) – Edge index.

**Returns**

- **logits** (*torch.Tensor*) – Discriminator logits of positive examples.
- **neg_logits** (*torch.Tensor*) – Discriminator logits of negative examples.

## 8.4 DMGDBase

**class** pygod.nn.**DMGDBase**(*in_dim*, *hid_dim=64*, *num_layers=2*, *dropout=0.0*, *act=<function relu>*, *backbone=<class 'torch_geometric.nn.models.mlp.MLP'>*, *alpha=1*, *beta=1*, *gamma=1*, *warmup=2*, *k=2*, ***kwargs*)

Bases: Module

Deep Multiclass Graph Description

DMGD is a support vector based multiclass outlier detector. Its backbone is an autoencoder that reconstructs the adjacency matrix of the graph with MSE loss and homophily loss. It applies k-means to cluster the nodes embedding and then uses support vector to detect outliers.

See [BVVM20] for details.

**Parameters**

- **in_dim** (*int*) – Input dimension.
- **hid_dim** (*int, optional*) – Hidden dimension of model. Default: `64`.
- **num_layers** (*int, optional*) – Total number of layers in model. A half (floor) of the layers are for the encoder, the other half (ceil) of the layers are for decoders. Default: `4`.
- **dropout** (*float, optional*) – Dropout rate. Default: `0.`.
- **weight_decay** (*float, optional*) – Weight decay (L2 penalty). Default: `0.`.
- **act** (*callable activation function or None, optional*) – Activation function if not None. Default: `torch.nn.functional.relu`.
- **backbone** (*torch.nn.Module, optional*) – The backbone of the deep detector implemented in PyG. Default: `torch_geometric.nn.MLP`.
- **alpha** (*float, optional*) – Weight of the radius loss. Default: `1`.
- **beta** (*float, optional*) – Weight of the reconstruction loss. Default: `1`.
- **gamma** (*float, optional*) – Weight of the homophily loss. Default: `1`.
- **k** (*int, optional*) – The number of clusters. Default: `2`.
- ****kwargs** – Other parameters for the backbone.

**forward**(*x*, *edge_index*)

Forward computation.

**Parameters**

- **x** (*torch.Tensor*) – Input attribute embeddings.
- **edge_index** (*torch.Tensor*) – Edge index.

**Returns**

- **x_** (*torch.Tensor*) – Reconstructed attribute embeddings.
- **nd** (*torch.Tensor*) – Neighbor distance.

**loss_func**(*x*, *x_*, *nd*, *emb*)

Loss function for DMGD.

**Parameters**

- **x** (*torch.Tensor*) – Input attribute embeddings.
- **x** – Reconstructed attribute embeddings.
- **nd** (*torch.Tensor*) – Neighbor distance.
- **emb** (*torch.Tensor*) – Embeddings.

**Returns**

**loss** – Loss value.

**Return type**

torch.Tensor

**static process_graph**(*data*)

Obtain the dense adjacency matrix of the graph.

**Parameters**

**data** (*torch_geometric.data.Data*) – Input graph.

## 8.5 DOMINANTBase

class pygod.nn.**DOMINANTBase**(*in_dim*, *hid_dim=64*, *num_layers=4*, *dropout=0.0*, *act=<function relu>*, *sigmoid_s=False*, *backbone=<class 'torch_geometric.nn.models.basic_gnn.GCN'>*, ***kwargs*)

Bases: Module

Deep Anomaly Detection on Attributed Networks

DOMINANT is an anomaly detector consisting of a shared graph convolutional encoder, a structure reconstruction decoder, and an attribute reconstruction decoder. The reconstruction mean squared error of the decoders are defined as structure anomaly score and attribute anomaly score, respectively.

See [DLBL19] for details.

**Parameters**

- **in_dim** (*int*) – Input dimension of model.
- **hid_dim** (*int*) – Hidden dimension of model. Default: 64.
- **num_layers** (*int, optional*) – Total number of layers in model. A half (floor) of the layers are for the encoder, the other half (ceil) of the layers are for decoders. Default: 4.
- **dropout** (*float, optional*) – Dropout rate. Default: 0..
- **act** (*callable activation function or None, optional*) – Activation function if not None. Default: torch.nn.functional.relu.
- **sigmoid_s** (*bool, optional*) – Whether to apply sigmoid to the structure reconstruction. Default: False.
- **backbone** (*torch.nn.Module, optional*) – The backbone of the deep detector implemented in PyG. Default: torch_geometric.nn.GCN.
- ****kwargs** (*optional*) – Additional arguments for the backbone.

**forward**(*x*, *edge_index*)

Forward computation.

**Parameters**

- **x** (*torch.Tensor*) – Input attribute embeddings.
- **edge_index** (*torch.Tensor*) – Edge index.

**Returns**

- **x_** (*torch.Tensor*) – Reconstructed attribute embeddings.
- **s_** (*torch.Tensor*) – Reconstructed adjacency matrix.

**static process_graph**(*data*)

Obtain the dense adjacency matrix of the graph.

**Parameters**

**data** (*torch_geometric.data.Data*) – Input graph.

## 8.6 DONEBase

**class** pygod.nn.**DONEBase**(*x_dim*, *s_dim*, *hid_dim=64*, *num_layers=4*, *dropout=0.0*, *act=<function relu>*, *w1=0.2*, *w2=0.2*, *w3=0.2*, *w4=0.2*, *w5=0.2*, ***kwargs*)

Bases: Module

Deep Outlier Aware Attributed Network Embedding

DONE consists of an attribute autoencoder and a structure autoencoder. It estimates five losses to optimize the model, including an attribute proximity loss, an attribute homophily loss, a structure proximity loss, a structure homophily loss, and a combination loss. It calculates three outlier scores, and averages them as an overall scores. This model is transductive only.

See [BVM20] for details.

**Parameters**

- **x_dim** (*int*) – Input dimension of attribute.
- **s_dim** (*int*) – Input dimension of structure.
- **hid_dim** (*int, optional*) – Hidden dimension of model. Default: 64.
- **num_layers** (*int, optional*) – Total number of layers in model. A half (floor) of the layers are for the encoder, the other half (ceil) of the layers are for decoders. Default: 4.
- **dropout** (*float, optional*) – Dropout rate. Default: 0..
- **weight_decay** (*float, optional*) – Weight decay (L2 penalty). Default: 0..
- **act** (*callable activation function or None, optional*) – Activation function if not None. Default: torch.nn.functional.relu.
- **w1** (*float, optional*) – Weight of structure proximity loss. Default: 0.2.
- **w2** (*float, optional*) – Weight of structure homophily loss. Default: 0.2.
- **w3** (*float, optional*) – Weight of attribute proximity loss. Default: 0.2.
- **w4** (*float, optional*) – Weight of attribute homophily loss. Default: 0.2.
- **w5** (*float, optional*) – Weight of combination loss. Default: 0.2.
- ****kwargs** – Other parameters for the backbone.

**forward**(*x*, *s*, *edge_index*)

Forward computation.

**Parameters**

- **x** (*torch.Tensor*) – Input attribute embeddings.
- **s** (*torch.Tensor*) – Input structure embeddings.
- **edge_index** (*torch.Tensor*) – Edge index.

**Returns**

- **x_** (*torch.Tensor*) – Reconstructed attribute embeddings.
- **s_** (*torch.Tensor*) – Reconstructed structure embeddings.
- **h_a** (*torch.Tensor*) – Attribute hidden embeddings.
- **h_s** (*torch.Tensor*) – Structure hidden embeddings.
- **dna** (*torch.Tensor*) – Attribute neighbor distance.

- **dns** (*torch.Tensor*) – Structure neighbor distance.

**loss_func**(*x*, *x_*, *s*, *s_*, *h_a*, *h_s*, *dna*, *dns*)

Loss function for DONE.

**Parameters**

- **x** (*torch.Tensor*) – Input attribute embeddings.
- **x** – Reconstructed attribute embeddings.
- **s** (*torch.Tensor*) – Input structure embeddings.
- **s** – Reconstructed structure embeddings.
- **h_a** (*torch.Tensor*) – Attribute hidden embeddings.
- **h_s** (*torch.Tensor*) – Structure hidden embeddings.
- **dna** (*torch.Tensor*) – Attribute neighbor distance.
- **dns** (*torch.Tensor*) – Structure neighbor distance.

**Returns**

- **loss** (*torch.Tensor*) – Loss value.
- **oa** (*torch.Tensor*) – Attribute outlier scores.
- **os** (*torch.Tensor*) – Structure outlier scores.
- **oc** (*torch.Tensor*) – Combined outlier scores.

**static process_graph**(*data*)

Obtain the dense adjacency matrix of the graph.

**Parameters**

**data** (*torch_geometric.data.Data*) – Input graph.

## 8.7 GAANBase

**class** pygod.nn.**GAANBase**(*in_dim*, *noise_dim*, *hid_dim=64*, *num_layers=4*, *dropout=0.0*, *act=<function relu>*, ***kwargs*)

Bases: Module

Generative Adversarial Attributed Network Anomaly Detection

GAAN is a generative adversarial attribute network anomaly detection framework, including a generator module, an encoder module, a discriminator module, and uses anomaly evaluation measures that consider sample reconstruction error and real sample recognition confidence to make predictions. This model is transductive only.

See [CLW+20] for details.

**Parameters**

- **in_dim** (*int*) – Input dimension of the node features.
- **noise_dim** (*int, optional*) – Input dimension of the Gaussian random noise. Defaults: 16.
- **hid_dim** (*int, optional*) – Hidden dimension of model. Default: 64.

- **num_layers** (*int, optional*) – Total number of layers in model. A half (floor) of the layers are for the generator, the other half (ceil) of the layers are for encoder. Default: 4.
- **dropout** (*float, optional*) – Dropout rate. Default: 0..
- **act** (*callable activation function or None, optional*) – Activation function if not None. Default: torch.nn.functional.relu.
- ****kwargs** – Other parameters for the backbone.

**forward**(*x*, *noise*)

Forward computation.

**Parameters**

- **x** (*torch.Tensor*) – Input attribute embeddings.
- **noise** (*torch.Tensor*) – Input noise.

**Returns**

- **x_** (*torch.Tensor*) – Reconstructed node features.
- **a** (*torch.Tensor*) – Reconstructed adjacency matrix from real samples.
- **a_** (*torch.Tensor*) – Reconstructed adjacency matrix from fake samples.

**static process_graph**(*data*)

Obtain the dense adjacency matrix of the graph.

**Parameters**

**data** (*torch_geometric.data.Data*) – Input graph.

## 8.8 GADNRBase

**class** pygod.nn.**GADNRBase**(*in_dim*, *hid_dim=64*, *encoder_layers=1*, *deg_dec_layers=4*, *fea_dec_layers=3*, *sample_size=2*, *sample_time=3*, *neighbor_num_list=None*, *neigh_loss='KL'*, *lambda_loss1=0.01*, *lambda_loss2=0.001*, *lambda_loss3=0.0001*, *full_batch=True*, *dropout=0.0*, *act=<function relu>*, *backbone=<class 'torch_geometric.nn.models.basic_gnn.GCN'>*, *device='cpu'*, ***kwargs*)

Bases: Module

Graph Anomaly Detection via Neighborhood Reconstruction

GAD-NR is a new type of GAE based on neighborhood reconstruction for graph anomaly detection. GAD-NR aims to reconstruct the entire neighborhood (including local structure, self attributes, and neighbors attributes) around a node based on the corresponding node representation.

See [RSL+24] for details.

**Parameters**

- **in_dim** (*int*) – Input dimension of model.
- **hid_dim** (*int*) – Hidden dimension of model. Default: 64.
- **encoder_layers** (*int, optional*) – The number of layers for the graph encoder. Default: 1.
- **deg_dec_layers** (*int, optional*) – The number of layers for the node degree decoder. Default: 4.

- **fea_dec_layers** (*int, optional*) – The number of layers for the node feature decoder. Default: 3.
- **sample_size** (*int, optional*) – The number of samples for the neighborhood distribution. Default: 2.
- **sample_time** (*int, optional*) – The number sample times to remove the noise during node feature and neighborhood distribution reconstruction. Default: 3.
- **neighbor_num_list** (*torch.Tensor*) – The node degree tensor used by the PNAConv model.
- **neigh_loss** (*str, optional*) – The neighbor reconstruction loss. KL represents the KL divergence loss, W2 represents the W2 loss. Default: KL.
- **lambda_loss1** (*float, optional*) – The weight of the neighborhood reconstruction loss term. Default: 1e-2.
- **lambda_loss2** (*float, optional*) – The weight of the node feature reconstruction loss term. Default: 1e-3.
- **lambda_loss3** (*float, optional*) – The weight of the node degree reconstruction loss term. Default: 1e-4.
- **full_batch** (*bool, optional*) – Whether in the full batch or the mini-batch training/inference mode. Default: True.
- **dropout** (*float, optional*) – Dropout rate. Default: 0..
- **act** (*callable activation function or None, optional*) – Activation function if not None. Default: torch.nn.functional.relu.
- **backbone** (*torch.nn.Module, optional*) – The backbone of the deep detector implemented in PyG. Default: torch_geometric.nn.GCN.
- **device** (*string, optional*) – The device used by the model. Default: cpu.
- ****kwargs** (*optional*) – Additional arguments for the backbone.

**forward**(*x*, *edge_index*, *input_id=None*, *neighbor_dict=None*, *id_mapping=None*)

Forward computation.

**Parameters**

- **x** (*torch.Tensor*) – Input attribute embeddings.
- **edge_index** (*torch.Tensor*) – Edge index.
- **input_id** (*List*) – List of center node ids in the current batch. If input_id is not None, the input data is a sampled mini_batch. If input_id is None, the input data is a full batch. Default: None.
- **neighbor_dict** (*Dict*) – Dictionary where nodes in the current batch as keys and their neighbor list as corresponding values. If neighbor_dict is not None, the input data is a sampled mini_batch. If neighbor_dict is None, the input data is a full batch. Default: None.
- **id_mapping** (*Dict*) – Dictionary where nodes in the current batch as keys and their feature matrix id as the values. If id_mapping is not None, the input data is a sampled mini_batch. If id_mapping is None, the input data is a full batch. Default: None.

**Returns**

- **h0** (*torch.Tensor*) – Node feature initial embeddings.

- **degree_logits** (*torch.Tensor*) – Reconstructed node degree logits.
- **feat_recon_list** (*List[torch.Tensor]*) – Reconstructed node features.
- **neigh_recon_list** (*List[torch.Tensor]*) – Reconstructed neighbor distributions.

`loss_func`(*h0*, *degree_logits*, *feat_recon_list*, *neigh_recon_list*, *ground_truth_degree_matrix*)

The loss function proposed in the GAD-NR paper.

**Parameters**

- **h0** (*torch.Tensor*) – Node feature initial embeddings.
- **degree_logits** (*torch.Tensor*) – Reconstructed node degree logits.
- **feat_recon_list** (*List[torch.Tensor]*) – Reconstructed node features.
- **neigh_recon_list** (*List[torch.Tensor]*) – Reconstructed neighbor distributions.
- **ground_truth_degree_matrix** (*torch.Tensor*) – The ground truth degree of the input nodes.

**Returns**

- **loss** (*torch.Tensor*) – The total loss value used to backpropagate and update the model parameters.
- **loss_per_node** (*torch.Tensor*) – The original loss value per node used to compute the decision score (outlier score) of the node.
- **h_loss_per_node** (*torch.Tensor*) – The neigborhood reconstruction loss value per node used to compute the adaptive decision score (outlier score) of the node.
- **degree_loss_per_node** (*torch.Tensor*) – The node degree reconstruction loss value per node used to compute the adaptive decision score (outlier score) of the node.
- **feature_loss_per_node** (*torch.Tensor*) – The node feature reconstruction loss value per node used to compute the adaptive decision score (outlier score) of the node.

`static process_graph`(*data*, *input_id=None*)

Preprocess the input graph and obtain the required data for future use.

**Parameters**

- **data** (*torch_geometric.data.Data*) – Input graph.
- **input_id** (*List*) – List of center node ids in the current batch. If `input_id` is not `None`, the input data is a sampled mini_batch. If `input_id` is `None`, the input data is a full batch. Default: `None`.

**Returns**

- **data** (*torch_geometric.data.Data*) – Preprocessed input graph.
- **neighbor_dict** (*Dict*) – Dictionary where nodes in the input_id list as keys and their neighbor list as corresponding values.
- **neighbor_num_list** (*torch.Tensor*) – A n*1 tensor where its value represents the corresponding node degree for the nodes in input_id list.
- **id_mapping** (*Dict*) – Dictionary where nodes in the input_id list as keys and their feature matrix id as the values.

## 8.9 GAEBase

class pygod.nn.**GAEBase**(*in_dim*, *hid_dim=64*, *num_layers=4*, *dropout=0.0*, *act=<function relu>*, *backbone=<class 'torch_geometric.nn.models.basic_gnn.GCN'>*, *recon_s=False*, *sigmoid_s=False*, ***kwargs*)

Bases: Module

Graph Autoencoder

See [KW16] for details.

**Parameters**

- **in_dim** (*int*) – Input dimension of model.
- **hid_dim** (*int*) – Hidden dimension of model. Default: 64.
- **num_layers** (*int, optional*) – Total number of layers in model. Default: 4.
- **dropout** (*float, optional*) – Dropout rate. Default: 0..
- **act** (*callable activation function or None, optional*) – Activation function if not None. Default: torch.nn.functional.relu.
- **backbone** (*torch.nn.Module, optional*) – The backbone of the deep detector implemented in PyG. Default: torch_geometric.nn.GCN.
- **recon_s** (*bool, optional*) – Reconstruct the structure instead of node feature . Default: False.
- **sigmoid_s** (*bool, optional*) – Whether to use sigmoid function to scale the reconstructed structure. Default: False.
- ****kwargs** (*optional*) – Other parameters for the backbone.

**forward**(*x*, *edge_index*)

Forward computation.

**Parameters**

- **x** (*torch.Tensor*) – Input attribute embeddings.
- **edge_index** (*torch.Tensor*) – Edge index.

**Returns**

**x_** – Reconstructed embeddings.

**Return type**

torch.Tensor

static **process_graph**(*data*, *recon_s=False*)

Obtain the dense adjacency matrix of the graph.

**Parameters**

- **data** (*torch_geometric.data.Data*) – Input graph.
- **recon_s** (*bool, optional*) – Reconstruct the structure instead of node feature .

## 8.10 GUIDEBase

**class** pygod.nn.**GUIDEBase**(*dim_a*, *dim_s*, *hid_a=64*, *hid_s=4*, *num_layers=4*, *dropout=0.0*, *act=<function relu>*, ***kwargs*)

Bases: Module

Higher-order Structure based Anomaly Detection on Attributed Networks

GUIDE is an anomaly detector consisting of an attribute graph convolutional autoencoder, and a structure graph attentive autoencoder (not the same as the graph attention networks). Instead of the adjacency matrix, node motif degree is used as input of structure autoencoder. The reconstruction mean square error of the autoencoders are defined as structure anomaly score and attribute anomaly score, respectively.

Note: The calculation of node motif degree in preprocessing has high time complexity. It may take longer than you expect.

See [YZY+21] for details.

**Parameters**

- **dim_a** (*int*) – Input dimension for attribute.
- **dim_s** (*int*) – Input dimension for structure.
- **hid_a** (*int, optional*) – Hidden dimension for attribute. Default: 64.
- **hid_s** (*int, optional*) – Hidden dimension for structure. Default: 4.
- **num_layers** (*int, optional*) – Total number of layers in model. Default: 4.
- **dropout** (*float, optional*) – Dropout rate. Default: 0..
- **act** (*callable activation function or None, optional*) – Activation function if not None. Default: torch.nn.functional.relu.
- ****kwargs** – Other parameters for GCN.

**forward**(*x*, *s*, *edge_index*)

Forward computation of GUIDE.

**Parameters**

- **x** (*torch.Tensor*) – Input attribute embeddings.
- **s** (*torch.Tensor*) – Input structure embeddings.
- **edge_index** (*torch.Tensor*) – Edge index.

**Returns**

- **x_** (*torch.Tensor*) – Reconstructed attribute embeddings.
- **s_** (*torch.Tensor*) – Reconstructed structure embeddings.

## 8.11 OCGNNBase

**class** pygod.nn.**OCGNNBase**(*in_dim*, *hid_dim*, *num_layers=2*, *dropout=0.0*, *act=<function relu>*, *backbone=<class 'torch_geometric.nn.models.basic_gnn.GCN'>*, *beta=0.5*, *warmup=2*, *eps=0.001*, ***kwargs*)

Bases: Module

One-Class Graph Neural Networks for Anomaly Detection in Attributed Networks

OCGNN is an anomaly detector that measures the distance of anomaly to the centroid, in a similar fashion to the support vector machine, but in the embedding space after feeding towards several layers of GCN.

See [WJD+21] for details.

**Parameters**

- **in_dim** (*int*) – Input dimension of model.
- **hid_dim** (*int, optional*) – Hidden dimension of model. Default: 64.
- **num_layers** (*int, optional*) – Total number of layers in model. Default: 2.
- **dropout** (*float, optional*) – Dropout rate. Default: 0..
- **act** (*callable activation function or None, optional*) – Activation function if not None. Default: torch.nn.functional.relu.
- **backbone** (*torch.nn.Module*) – The backbone of the deep detector implemented in PyG. Default: torch_geometric.nn.GCN.
- **beta** (*float, optional*) – The weight between the reconstruction loss and radius. Default: 0.5.
- **warmup** (*int, optional*) – The number of epochs for warm-up training. Default: 2.
- **eps** (*float, optional*) – The slack variable. Default: 0.001.
- ****kwargs** – Other parameters for the backbone model.

**forward**(*x*, *edge_index*)

Forward computation.

**Parameters**

- **x** (*torch.Tensor*) – Input attribute embeddings.
- **edge_index** (*torch.Tensor*) – Edge index.

**Returns**

**emb** – Output embeddings.

**Return type**

torch.Tensor

**loss_func**(*emb*)

Loss function for OCGNN

**Parameters**

**emb** (*torch.Tensor*) – Embeddings.

**Returns**

- **loss** (*torch.Tensor*) – Loss value.
- **score** (*torch.Tensor*) – Outlier scores of shape $N$ with gradients.

CHAPTER

# NINE

# PYGOD.NN.CONV

Convolutional Layers for Graph Neural Networks.

**class** pygod.nn.conv.**GNAConv**(*in_channels*, *out_channels*)

Bases: MessagePassing

Graph Node Attention Network (GNA) layer. See [YZY+21] for more details.

**forward**(*s*, *edge_index*)

Forward computation.

**Parameters**

- **s** (*torch.Tensor*) – Input node embeddings.
- **edge_index** (*torch.Tensor*) – Edge index.

**Returns**
**s** – Updated node embeddings.

**Return type**
torch.Tensor

**message**(*s_i*, *s_j*, *edge_index*)

Constructs messages from node $j$ to node $i$ in analogy to $\phi_{\Theta}$ for each edge in `edge_index`. This function can take any argument as input which was initially passed to `propagate()`. Furthermore, tensors passed to `propagate()` can be mapped to the respective nodes $i$ and $j$ by appending `_i` or `_j` to the variable name, *.e.g.* `x_i` and `x_j`.

**class** pygod.nn.conv.**NeighDiff**

Bases: MessagePassing

Calculate the Euclidean distance between the node features of the central node and its neighbors, reducing by mean.

**forward**(*h*, *edge_index*)

Forward computation.

**Parameters**

- **h** (*torch.Tensor*) – Input node embeddings.
- **edge_index** (*torch.Tensor*) – Edge index.

**Returns**
**h** – Updated node embeddings.

**Return type**
torch.Tensor

**message**(*h_i*, *h_j*, *edge_index*)

Constructs messages from node $j$ to node $i$ in analogy to $\phi_{\Theta}$ for each edge in `edge_index`. This function can take any argument as input which was initially passed to `propagate()`. Furthermore, tensors passed to `propagate()` can be mapped to the respective nodes $i$ and $j$ by appending `_i` or `_j` to the variable name, *.e.g.* `x_i` and `x_j`.

CHAPTER

# TEN

# PYGOD.NN.ENCODER

Graph Neural Networks Encoders

**class** pygod.nn.encoder.**GNA**(*in_channels*, *hidden_channels*, *num_layers*, *out_channels*, *dropout*, *act*)

Graph Node Attention Network (GNA). See [YZY+21] for more details.

**forward**(*s*, *edge_index*)

Forward computation.

**Parameters**

- **s** (*torch.Tensor*) – Input node embeddings.
- **edge_index** (*torch.Tensor*) – Edge index.

**Returns**

**s** – Updated node embeddings.

**Return type**

torch.Tensor

CHAPTER

# ELEVEN

# PYGOD.NN.DECODER

class pygod.nn.decoder.**DotProductDecoder**(*in_dim*, *hid_dim=64*, *num_layers=1*, *dropout=0.0*, *act=<function relu>*, *sigmoid_s=False*, *backbone=<class 'torch_geometric.nn.models.basic_gnn.GCN'>*, ***kwargs*)

Dot product decoder for the structure reconstruction, which is defined as $A' = \sigma(ZZ^\intercal)$, where $\sigma$ is the optional sigmoid function, $Z$ is the input hidden embedding, and the $A'$ is the reconstructed adjacency matrix.

**Parameters**

- **in_dim** (*int*) – Input dimension of node features.
- **hid_dim** (*int, optional*) – Hidden dimension of model. Default: `64`.
- **num_layers** (*int, optional*) – Number of layers in the decoder. Default: `1`.
- **dropout** (*float, optional*) – Dropout rate. Default: `0.`.
- **act** (*callable activation function or None, optional*) – Activation function if not None. Default: `torch.nn.functional.relu`.
- **sigmoid_s** (*bool, optional*) – Whether to apply sigmoid to the structure reconstruction. Default: `False`.
- **backbone** (*torch.nn.Module, optional*) – The backbone of the deep decoder implemented in PyG. Default: `torch_geometric.nn.GCN`.
- ****kwargs** (*optional*) – Additional arguments for the backbone.

**forward**(*x*, *edge_index*)

Forward computation.

**Parameters**

- **x** (*torch.Tensor*) – Input node embeddings.
- **edge_index** (*torch.Tensor*) – Edge index.

**Returns**

**s_** – Reconstructed adjacency matrix.

**Return type**

torch.Tensor

CHAPTER

# TWELVE

# PYGOD.NN.FUNCTIONAL

Functional Interface for PyGOD

pygod.nn.functional.**KL_neighbor_loss**(*predictions*, *targets*, *mask_len*, *device*)

The local neighor distribution KL divergence loss used in GAD-NR. Source: https://github.com/Graph-COM/GAD-NR/blob/master/GAD-NR_inj_cora.ipynb

pygod.nn.functional.**W2_neighbor_loss**(*predictions*, *targets*, *mask_len*, *device*)

The local neighor distribution W2 loss used in GAD-NR. Source: https://github.com/Graph-COM/GAD-NR/blob/master/GAD-NR_inj_cora.ipynb

pygod.nn.functional.**double_recon_loss**(*x*, *x_*, *s*, *s_*, *weight=0.5*, *pos_weight_a=0.5*, *pos_weight_s=0.5*, *bce_s=False*)

Double reconstruction loss function for feature and structure. The loss function is defined as $\alpha E_a + (1-\alpha)E_s$, where $\alpha$ is the weight between 0 and 1 inclusive, and $E_a$ and $E_s$ are the reconstruction loss for feature and structure, respectively. The first dimension is kept for outlier scores of each node.

For feature reconstruction, we use mean squared error loss: $E_a = \|X - X'\| \odot H$, where $H = \begin{cases} 1-\eta & \text{if } x_{ij} = 0 \\ \eta & \text{if } x_{ij} > 0 \end{cases}$, and $\eta$ is the positive weight for feature.

For structure reconstruction, we use mean squared error loss by default: $E_s = \|S - S'\| \odot \Theta$, where $\Theta = \begin{cases} 1-\theta & \text{if } s_{ij} = 0 \\ \theta & \text{if } s_{ij} > 0 \end{cases}$, and $\theta$ is the positive weight for structure. Alternatively, we can use binary cross entropy loss for structure reconstruction: $E_s = \text{BCE}(S, S' \odot \Theta)$.

**Parameters**

- **x** (*torch.Tensor*) – Ground truth node feature
- **x** – Reconstructed node feature
- **s** (*torch.Tensor*) – Ground truth node structure
- **s** – Reconstructed node structure
- **weight** (*float, optional*) – Balancing weight $\alpha$ between 0 and 1 inclusive between node feature and graph structure. Default: `0.5`.
- **pos_weight_a** (*float, optional*) – Positive weight for feature $\eta$. Default: `0.5`.
- **pos_weight_s** (*float, optional*) – Positive weight for structure $\theta$. Default: `0.5`.
- **bce_s** (*bool, optional*) – Use binary cross entropy for structure reconstruction loss.

**Returns**

**score** – Outlier scores of shape $N$ with gradients.

**Return type**
torch.tensor

CHAPTER

# THIRTEEN

# PYGOD.UTILS

## 13.1 Data Loading

**class** pygod.utils.**load_data**(*name*, *cache_dir=None*)

Data loading function. See data repository for supported datasets. For injected/generated datasets, the labels meanings are as follows.

- 0: inlier
- 1: contextual outlier only
- 2: structural outlier only
- 3: both contextual outlier and structural outlier

**Parameters**

- **name** (*str*) – The name of the dataset.
- **cache_dir** (*str, optional*) – The directory for dataset caching. Default: None.

**Returns**

**data** – The outlier dataset.

**Return type**

torch_geometric.data.Data

### Examples

```
>>> from pygod.utils import load_data
>>> data = load_data(name='weibo') # in PyG format
>>> y = data.y.bool()    # binary labels (inlier/outlier)
>>> yc = data.y >> 0 & 1 # contextual outliers
>>> ys = data.y >> 1 & 1 # structural outliers
```

## 13.2 Score Conversion

pygod.utils.**to_edge_score**(*score*, *edge_index*)

Convert outlier node score to outlier edge score by averaging the scores of two nodes connected by an edge.

**Parameters**

- **score** (*torch.Tensor*) – The node score.
- **edge_index** (*torch.Tensor*) – The edge index.

**Returns**

**score** – The edge score.

**Return type**

torch.Tensor

pygod.utils.**to_graph_score**(*score*)

Convert outlier node score to outlier graph score by averaging the scores of all nodes in a graph.

**Parameters**

**score** (*torch.Tensor*) – The node score.

**Returns**

**score** – The graph score.

**Return type**

torch.Tensor

CHAPTER

# FOURTEEN

# CITE

Our software paper and benchmark paper are publicly available. If you use PyGOD or BOND in a scientific publication, we would appreciate citations to the following papers:

```
@article{liu2022pygod,
  title={PyGOD: A Python Library for Graph Outlier Detection},
  author={Liu, Kay and Dou, Yingtong and Zhao, Yue and Ding, Xueying and Hu, Xiyang and␣
↪Zhang, Ruitong and Ding, Kaize and Chen, Canyu and Peng, Hao and Shu, Kai and Chen,␣
↪George H. and Jia, Zhihao and Yu, Philip S.},
  journal={arXiv preprint arXiv:2204.12095},
  year={2022}
}
@article{liu2022bond,
  title={Bond: Benchmarking unsupervised outlier node detection on static attributed␣
↪graphs},
  author={Liu, Kay and Dou, Yingtong and Zhao, Yue and Ding, Xueying and Hu, Xiyang and␣
↪Zhang, Ruitong and Ding, Kaize and Chen, Canyu and Peng, Hao and Shu, Kai and Sun,␣
↪Lichao and Li, Jundong and Chen, George H. and Jia, Zhihao and Yu, Philip S.},
  journal={Advances in Neural Information Processing Systems},
  volume={35},
  pages={27021--27035},
  year={2022}
}
```

or:

```
Liu, K., Dou, Y., Zhao, Y., Ding, X., Hu, X., Zhang, R., Ding, K., Chen, C., Peng, H.,␣
↪Shu, K. and Chen, G.H., Jia, Z., and Yu, P.S. 2022. PyGOD: A Python Library for Graph␣
↪Outlier Detection. arXiv preprint arXiv:2204.12095.
Liu, K., Dou, Y., Zhao, Y., Ding, X., Hu, X., Zhang, R., Ding, K., Chen, C., Peng, H.,␣
↪Shu, K. and Sun, L., Li, J., Chen, G.H., Jia, Z., and Yu, P.S. 2022. Bond:␣
↪Benchmarking unsupervised outlier node detection on static attributed graphs. Advances␣
↪in Neural Information Processing Systems, 35, pp.27021-27035.
```

CHAPTER

FIFTEEN

# CORE TEAM

PyGOD is a great team effort by researchers from UIC, IIT, BUAA, ASU, and CMU. Our core team members include:

Kay Liu (UIC)

Yingtong Dou (UIC)

Yue Zhao (CMU)

Xueying Ding (CMU)

Xiyang Hu (CMU)

Ruitong Zhang (BUAA)

Kaize Ding (ASU)

Canyu Chen (IIT)

CHAPTER

SIXTEEN

# REFERENCE

# BIBLIOGRAPHY


[BVVM20] S Bandyopadhyay, S Vishal Vivek, and MN Murty. Integrating network embedding and community outlier detection via multiclass graph description. *Frontiers in Artificial Intelligence and Applications*, 325:976–983, 2020.

[BLM19] Sambaran Bandyopadhyay, N Lokesh, and M Narasimha Murty. Outlier aware network embedding for attributed networks. In *Proceedings of the AAAI conference on artificial intelligence*, volume 33, 12–19. 2019.

[BVM20] Sambaran Bandyopadhyay, Saley Vishal Vivek, and MN Murty. Outlier resistant unsupervised deep architectures for attributed network embedding. In *Proceedings of the 13th International Conference on Web Search and Data Mining*, 25–33. 2020.

[CCL+21] Lei Cai, Zhengzhang Chen, Chen Luo, Jiaping Gui, Jingchao Ni, Ding Li, and Haifeng Chen. Structural temporal graph neural networks for anomaly detection in dynamic graphs. In *Proceedings of the 30th ACM International Conference on Information & Knowledge Management*, 3747–3756. 2021.

[CLW+20] Zhenxing Chen, Bo Liu, Meiqing Wang, Peng Dai, Jun Lv, and Liefeng Bo. Generative adversarial attributed network anomaly detection. In *Proceedings of the 29th ACM International Conference on Information & Knowledge Management*, 1989–1992. 2020.

[DLBL19] Kaize Ding, Jundong Li, Rohit Bhanushali, and Huan Liu. Deep anomaly detection on attributed networks. In *Proceedings of the 2019 SIAM International Conference on Data Mining*, 594–602. SIAM, 2019.

[DLS+20] Yingtong Dou, Zhiwei Liu, Li Sun, Yutong Deng, Hao Peng, and Philip S Yu. Enhancing graph neural network-based fraud detectors against camouflaged fraudsters. In *Proceedings of the 29th ACM International Conference on Information & Knowledge Management*, 315–324. 2020.

[FZL20] Haoyi Fan, Fengbin Zhang, and Zuoyong Li. Anomalydae: dual autoencoder for anomaly detection on attributed networks. In *ICASSP 2020 - 2020 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, 5685–5689. 2020. doi:10.1109/ICASSP40776.2020.9053387.

[KW16] Thomas N Kipf and Max Welling. Variational graph auto-encoders. *arXiv preprint arXiv:1611.07308*, 2016.

[KKSZ11] Hans-Peter Kriegel, Peer Kroger, Erich Schubert, and Arthur Zimek. Interpreting and unifying outlier scores. In *Proceedings of the 2011 SIAM International Conference on Data Mining*, 13–24. SIAM, 2011.

[LDHL17] Jundong Li, Harsh Dani, Xia Hu, and Huan Liu. Radar: residual analysis for anomaly detection in attributed networks. In *IJCAI*, 2152–2158. 2017.

[LLP+21] Yixin Liu, Zhao Li, Shirui Pan, Chen Gong, Chuan Zhou, and George Karypis. Anomaly detection on attributed networks via contrastive self-supervised learning. *IEEE transactions on neural networks and learning systems*, 2021.

[PLL+18] Zhen Peng, Minnan Luo, Jundong Li, Huan Liu, and Qinghua Zheng. Anomalous: a joint modeling approach for anomaly detection on attributed networks. In *IJCAI*, 3513–3519. 2018.

[PVD20] Lorenzo Perini, Vincent Vercruyssen, and Jesse Davis. Quantifying the confidence of anomaly detectors in their example-wise predictions. In *Joint European Conference on Machine Learning and Knowledge Discovery in Databases*, 227–243. Springer, 2020.

[RSL+24] Amit Roy, Juan Shu, Jia Li, Carl Yang, Olivier Elshocht, Jeroen Smeets, and Pan Li. Gad-nr : graph anomaly detection via neighborhood reconstruction. In *Proceedings of the 17th ACM International Conference on Web Search and Data Mining*. 2024.

[WJD+21] Xuhong Wang, Baihong Jin, Ying Du, Ping Cui, Yingshui Tan, and Yupu Yang. One-class graph neural networks for anomaly detection in attributed networks. *Neural computing and applications*, 33(18):12073–12085, 2021.

[XYFS07] Xiaowei Xu, Nurcan Yuruk, Zhidan Feng, and Thomas AJ Schweiger. Scan: a structural clustering algorithm for networks. In *Proceedings of the 13th ACM SIGKDD international conference on Knowledge discovery and data mining*, 824–833. 2007.

[XHZ+22] Zhiming Xu, Xiao Huang, Yue Zhao, Yushun Dong, and Jundong Li. Contrastive attributed network anomaly detection with data augmentation. In *Pacific-Asian Conference on Knowledge Discovery and Data Mining (PAKDD)*. 2022.

[YZY+21] Xu Yuan, Na Zhou, Shuo Yu, Huafei Huang, Zhikui Chen, and Feng Xia. Higher-order structure based anomaly detection on attributed networks. In *2021 IEEE International Conference on Big Data (Big Data)*, 2691–2700. IEEE, 2021.

# PYTHON MODULE INDEX

## p

# INDEX

## G



## H



## I



## K



## L



## M



## N



## O



## P

## R



## S



## T



## W