Issues with Propagation Based Models for Graph-Level Outlier Detection

Lingxiao Zhao [1] **Leman Akoglu** [1]

Abstract

Graph-Level Outlier Detection (GLOD) is the task of identifying unusual graphs within a graph database, which received little attention compared to node-level detection in a single graph. As propagation based graph embedding by GNNs and graph kernels achieved promising results on another graph-level task, i.e. graph classification, we study applying those models to tackle GLOD. Instead of developing new models, *this paper identifies and delves into a fundamental and intriguing issue with applying propagation based models to GLOD*, with evaluation conducted on repurposed binary graph classification datasets where one class is down-sampled as outlier. We find that ROC-AUC performance of the models changes significantly ("flips" from high to low) depending on *which* class is down-sampled. Interestingly, ROC-AUCs on these two variants approximately sum to 1 and their performance gap is amplified with increasing propagations. We carefully study the graph embedding space produced by propagation based models and find two driving factors: (1) disparity between within-class densities which is amplified by propagation, and (2) overlapping support (mixing of embeddings) across classes. Our study sheds light onto the effects of using graph propagation based models and classification datasets for outlier detection for the first time.

1. Introduction

Given a graph database (i.e., a large collection) of node-attributed/labeled graphs, how can we effectively identify the anomalous ones within the set? Graphs of such nature are widespread in finance, health care and insurance, cybersecurity, fault monitoring, etc. where the outlier detection task finds a long list of applications such as identifying rare transaction graphs (Nguyen et al., 2020), command flow graphs (Manzoor et al., 2016), and human poses (Markovitz et al., 2020), fake news (Monti et al., 2019), traffic events (Harshaw et al., 2016), buggy software (Liu et al., 2005), money laundering (Weber et al., 2019), and so on.

In this paper we study the graph-level outlier detection (GLOD) problem. Despite its wide application and growing influence in real-world, surprisingly, limited work exists for specifically solving this problem. As propagation based methods like graph kernels and graph neural networks achieve great success in capturing graph representation, we specifically adapt several propagation based methods for solving GLOD including kernels and GNNs. Surprisingly, peculiar issues have been consistently observed for all propagation based methods over lots of datasets when evaluating on repurposed binary graph classification dataset (down-sample one class as ground-truth outlier class). As observed issues become an unavoidable obstacle for effectively solving the graph-level outlier detection problem, we aim to provide extensive and deep study to understand these issues and we believe it has broader impact on outlier detection and graph representation learning.

We summarize our contributions in the following:

- **Study of Deep Graph-level Outlier Detection:** We start with the design and evaluation of two different categories of models for outlier detection in graph databases; namely, (1) two-stage models—pipelining unsupervised graph-level representation learning with off-the-shelf outlier detectors, and (2) end-to-end models—learning representations simultaneously with optimizing an anomaly detection objective, such as one-class classification or reconstruction loss. (Sec. 2)
- **"Performance Flip" Issue:** We evaluated the aforementioned graph outlier models on repurposed binary graph classification dataset, by down-sampling one class of those datasets in *both* "directions" as outlier. Surprisingly, we find that most models, while achieving high detection performance on one variant, fail considerably on the other. That is, we identify the intriguing issue of what we call "performance flip" depending on which class has been down-sampled.
- **Driving Factors behind "Performance Flip":** We identify two key leading factors behind the observed "performance flip" issue, particularly (1) *density disparity*; where the density of graph embeddings differ considerably between two classes, and (2) *overlapping support*; where the distributions of graph embeddings from the two classes exhibit overlapping support in the representation space. We design quantitative metrics

to concretely measure those factors (Sec. 3.3).

- **Insights for Outlier Detection and Beyond:** Our findings have implications for the fair and systematic evaluation of outlier models. Moreover, we argue that issues we identify can extend beyond outlier detection, with possible implications on graph classification and clustering. It also questions current graph representation learning methods.

2. Outlier Detection Problem & Models

In this paper we focus on the graph-level outlier detection problem. The intriguing "performance flip" issue we observe arises from repurposing binary graph classification datasets for outlier detection evaluation. As far as we know, there is limited work studying the graph-level outlier detection problem, where the goal is to discover graphs with rare, unusual patterns which can be distinguished from the majority of graphs in a database. We call attention to the problem as it applies to many important real-world tasks from diverse domains such as drug discovery, money laundering, molecular synthesis, rare human pose detection (Markovitz et al., 2020), fake news detection (Monti et al., 2019), traffic events detection (Harshaw et al., 2016), and buggy software detection (Liu et al., 2005).

2.1. Graph-Level Outlier Detection

Let $G = (\mathcal{V}, \mathcal{E}, \mathbf{X})$ be an attributed or labeled graph with $\mathcal{V}$ and $\mathcal{E}$ depicting its vertex set and edge set, where each node $i \in \mathcal{V}$ is associated with a feature vector $\mathbf{x}_i \in \mathbb{R}^d$ and $\mathbf{X} = [\mathbf{x}_1, \ldots, \mathbf{x}_n]^T$ denotes the feature matrix, $n = |\mathcal{V}|$ being the total number of nodes. For labeled graphs, each node feature vector $\mathbf{x}_i \in \mathbb{R}^d$ is a one-hot encoded vector with d being the total number of unique (discrete) node labels.

Definition 2.1 (Graph-Level Outlier Detection Problem (GLOD))**.** *Given a graph database $\mathcal{G} = \{G_1, \ldots, G_N\}$ containing N labeled or attributed graphs, find the graphs that differ significantly from the majority of graphs in $\mathcal{G}$.*

The above problem is a general statement for graph-level outlier detection. In the real-world how one defines rareness or the degree of difference to the majority may be critical and may change depending on the application.

2.2. Graph-Level Outlier Detection Models

Although there is no specifically designed method existing for GLOD, several methods for solving graph classification can be easily modified for tackling the problem. In this paper we mainly focus on three **propagation** based methods that can be categorized as two types: two-stage versus end-to-end. For the purposes of this paper, we find these three methods to be sufficiently illustrative of the issues we discover. Notice that we focus on studying propagation based methods as all message-passing based GNNs belong to this category, and they share similar issues.

2.2.1. Two-Stage Graph Outlier Detection

Two-stage graph outlier detection approaches first transform graphs into graph embeddings or similarities between graphs by using unsupervised graph embedding methods (such as graph2vec (Narayanan et al., 2017) and FGSD (Verma & Zhang, 2017)) or graph kernels (such as Weisfeiler-Leman (WL) kernel (Shervashidze et al., 2011) and propagation kernel (Neumann et al., 2016)). Then traditional outlier detectors such as Local Outlier Factor (LOF (Breunig et al., 2000)), and one-class SVM (OCSVM) (Manevitz & Yousef, 2001) can be used to detect outliers in the embedding (vector) space. These approaches are easy to use and do not require much hyperparameter tuning, which makes them relatively stable for an unsupervised task like outlier detection. Nevertheless, two-stage methods may suffer from suboptimal solution as the feature extractor and outlier detector are independent.

To illustrate the performance flip issue, we focus on graph kernel based two-stage approaches with well-known outlier detectors LOF. A graph kernel defines a kernel function $\mathcal{K}$ that outputs a similarity between two graphs. Formally it can be written as $\mathcal{K}(G, G') = \langle \phi(G), \phi(G') \rangle_{\mathcal{H}}$, where $\mathcal{H}$ is a RKHS and $\langle \cdot, \cdot \rangle$ is the dot product in $\mathcal{H}$. The mapping $\phi(G)$ transforms graph G to an embedding vector in $\mathcal{H}$, which in our case contains counts of atomic subgraph patterns. Specifically we use the Weisfeiler-Leman subtree kernel and the propagation kernel, described as follows.

Weisfeiler-Leman Subtree Kernel. Inspired by Weisfeiler-Leman (WL) test of graph isomorphism (Weisfeiler & Leman, 1968) (a.k.a. the color-refinement algorithm), WL subtree kernel (Shervashidze et al., 2011) processes a labeled graph by iteratively re-labeling each vertex with a new label compressed from a multiset label consisting of the vertex's original label and the sorted labels of its neighbors. This procedure repeats for L iterations for all graphs and outputs L re-labeled graphs $\{G_1, ..., G_L\}$ for every graph G. One can easily show that each vertex in G_l at l iterations represents the subtree of the original vertex with depth l. WL subtree kernel compares two graphs by simply counting the number of co-occurrences of labels in both graphs at each iteration. The similarity score of two graphs is the summation of similarities across iterations.

Sparsification. Next we highlight a key issue of the WL subtree kernel that is closely related to the performance flip issue we discover. Substructure-based graph kernels consider each substructure as a separate feature to compare among graphs. The total number of distinct substructures grows exponentially in the diameter of the substructures,

which leads to the *sparsity problem* — that only a limited number of substructures would be shared among graphs. This property has also been referred to as *diagonal dominance* (Yanardag & Vishwanathan, 2015; Narayanan et al., 2016) — wherein each individual graph would mostly be similar only to itself but not much to any other graph. The sparsification property of WL kernel is visualized in Fig. 1, where the diagonal dominance and diminishing similarity among graphs are observed clearly.

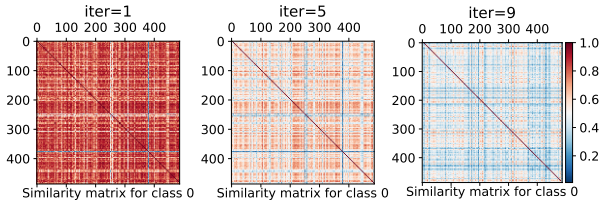


Figure 1: Sparsification in WL: Pairwise similarity of graphs (from DD dataset) decreases with increasing number of iterations.

Propagation Kernel. Propagation kernel (PK) (Neumann et al., 2016) is inspired from the idea of propagating label information among nodes. It shares similar structure as WL kernel. See Appendix.A.1 for detail introduction and its sparsification problem.

2.2.2. End-to-End Deep Graph Outlier Detection

Although several works have successfully applied GNNs to node-level outlier detection on a single graph, such as OCGNN (Wang et al., 2020) and DOMINANT (Ding et al., 2019), there is no deep model proposed for graph-level outlier detection. Here we present a GNN model adapted from graph classification, and leverage a one-class classification objective function to address graph-level outlier detection. Compared with the widely used Graph Convolution Network (GCN) (Kipf & Welling, 2017) model, Graph Isomorphism Network (GIN) (Xu et al., 2019) has been shown to be as powerful as the WL test of graph isomorphism, as such, we design a GIN based graph-level outlier detector. Note that an earlier GNN model called DGCNN (Zhang et al., 2018) has discussed its connection to WL subtree kernel and propagation kernel. GIN builds on the ideas of DGCNN, and as a result also shares connection to these graph kernels.

Let $h_v^{(l)}$ be the l-th layer representation of node v in the GIN model. GIN updates node representations at each layer by

$$h_v^{(l)} = \text{MLP}^{(l)}\Big((1+\epsilon^{(l)})\cdot h_v^{(l-1)} + \sum_{u\in\mathcal{N}(v)} h_u^{(l-1)}\Big) \quad (1)$$

where MLP denotes a multi-layer perceptron (we use 2 layer) and $\mathcal{N}(v)$ denotes the direct neighbors of node v. After L layers, GIN generates the graph-level representation (i.e. graph embedding) using a readout function as follows.

$$h_G = \text{CONCAT}\Big(\text{MEAN}(\{h_v^{(l)}|v\in G\}) \mid {}_{l=0}^{L}\Big) \quad (2)$$

To build one-class classification into the GIN model, we borrow the idea from DeepSVDD (Ruff et al., 2018). Specifically, we optimize the one-class deep SVDD objective at the output layer of the GIN model as

$$\min_W \quad \frac{1}{N}\sum_{i=1}^{N}\|\text{GIN}(G_i;W)-\mathbf{c}\|^2 + \frac{\lambda}{2}\sum_{l=1}^{L}\|W^l\|_F^2 \quad (3)$$

where W^l denotes the parameter of GIN at the l-th layer, $W=\{W^1,...,W^l\}$, and $\mathbf{c}$ is the center of the hypersphere in the representation space that is obtained as the average of all graph representations upon initializing GIN model. After training the model on all graphs, the distance to center is used as the outlier score for each graph.

3. Applying Propagation Based Models to GLOD: Issues

In this section we present in more detail the peculiar "performance flip" issue and related observations associated with applying propagation based models to GLOD. Interestingly, the strange issue happens to lots of GNN models (we have evaluated a number of GNN model variants for GLOD in developing new algorithms) on lots of datasets, although in this paper we only study 3 methods on 4 datasets. Thus, having a clear understanding of this issue becomes critical for developing new models for GLOD. In the following, we present the peculiar observations in detail (Sec. 3.1), state our hypothesis on the driving mechanisms behind these observations (Sec. 3.2), and introduce qualitative and quantitative measures for our empirical analysis (Sec. 3.3).

3.1. Peculiar Observations

Setup. We present our analysis on 4 widely used binary graph classification datasets: DD, PROTEINS, NCI1, and IMDB, which can be obtained from the TUDataset repository[1] (Morris et al., 2020). DD and PROTEINS are bioinformatics datasets, NCI1 is a molecular dataset and IMDB is a social network. Dataset statistics are summarized in Appendix.A.2. The performance flip and related issues are observed for the first 3 datasets, DD, PROTEINS, and NCI1, but not for IMDB, which is presented for comparison purposes.

We create two main variants of each dataset, by downsampling one class or the other (denoted class 0 or class 1), at varying rates. For each variant, we repeat the downsampling 10 times with different random seeds such that the

[1] https://chrsmrrs.github.io/datasets/

results are not an artifact of a specific data split. We report results using $L = 5$ for all models unless otherwise noted.

3.1.1. Peculiar Observation 1: Performance Flip.

The main observation we make in this work is that the detection performances of the outlier models appear to depend significantly on *which* class is down-sampled. Table 1 shows the ROC-AUC performance on all datasets, i.e. their two variants, at down-sampling rate 0.1 for all three methods, which supports the following observation:

Observation 1 (Performance Flip)**.** *(1) A large ROC-AUC gap is observed between the two different down-sampled variants of DD, PROTEINS, and NCI1, consistently across all models. The bad performance is surprisingly worse than random (ROC-AUC* < 0.5*). (2) Even more, the sum of the two ROC-AUC values on the two variants is approximately equal to 1.*

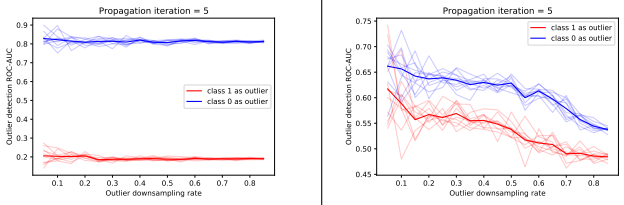
Table 1: Average ROC-AUC performance (and standard deviation) of 3 different graph embedding based methods for GLOD. Each dataset has 2 down-sampled variants, where outliers are created by down-sampling one of two classes (class 0 or class 1) with rate$=0.1$, averaged over 10 different down-samplings. Performance flip observed on DD, PROTEINS, and NCI1 for all outlier models, **where ROC-AUC is significantly larger on one variant than the other.** ROC-AUC values less than 0.5 are shown **in bold** as they indicate worse-than-random performance.

Dataset	Outlier Cls.	OCGIN	PK+LOF	WL+LOF
DD	0	0.712 (0.041)	0.801 (0.026)	0.815 (0.020)
	1	**0.350 (0.043)**	**0.240 (0.035)**	**0.186 (0.024)**
PROTEINS	0	0.685 (0.059)	0.564 (0.043)	0.664 (0.024)
	1	**0.362 (0.040)**	**0.447 (0.071)**	**0.276 (0.021)**
NCI1	0	**0.465 (0.033)**	**0.403 (0.030)**	**0.349 (0.022)**
	1	0.656 (0.031)	0.662 (0.023)	0.730 (0.012)
IMDB	0	0.515 (0.033)	0.543 (0.042)	0.651 (0.022)
	1	0.651 (0.032)	0.610 (0.053)	0.603 (0.038)

Recall that ROC-AUC represents the probability of correctly ranking a random positive instance (i.e. outlier) above a random negative instance, the observation 1 suggests that the models always consider the graphs from one *fixed* class to be more outlier than those from the other, irrespective of which one is down-sampled. The ranking by the models is agnostic to this so-called "ground-truth" but rather has a pre-determined bias toward one (fixed) class.

3.1.2. Peculiar Observation 2: Invariance to Down-sampling Rate.

When down-sampling one class as outlier with a certain down-sampling rate, we would conjecture that a lower rate would make the outlier detection task easier as the density of outliers becomes lower. Fig. 2 shows the detection ROC-AUC of WL+LOF (with $L = 5$ iterations) for various down-sampling rates, from 0.05 to 0.85, on both variants of two datasets. The conjecture appears to hold only for IMDB – on which performance flip is not observed. In contrast, the performance is strikingly flat on DD, PROTEINS, and NCI1. Similar results hold for PK+LOF and OCGIN on these three datasets.



(a) DD (b) IMDB

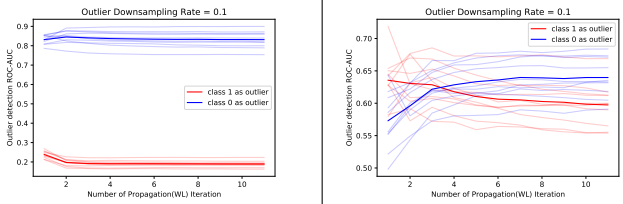
Figure 2: **Performance is invariant to downsampling rate** on DD, PROTEINS, and NCI1 for WL+LOF. Similar behavior is observed for the other two methods. See Appendix.A.5.

Observation 2 (Invariance to down-sampling rate)**.** *Performance flip issue is not an artifact of the down-sampling rate, in fact, ROC-AUC appears to be invariant to the rate.*

This observation is in agreement with Observation 1's (2) (AUCs-sum-approximately-to-1). The probabilistic interpretation is regarding *any two* random positive-negative instances, irrespective of the total number of instances from those groups.

3.1.3. Peculiar Observation 3: Growing Performance Gap with Propagation.

Another key property of the outlier models we employ in this work is the number of iterations (for WL and PK) or the number of layers (for GIN), earlier denoted with L (See Sec. 2.2), both of which correspond to propagations over the graph. Here we look at how the performance behaves under varying L. Fig. 3 shows the performance of WL+LOF on DD and IMDB datasets for two variants for L increased from 1 through 11. Results are qualitatively similar for PK+LOF, however OCGIN behaves differently (See Appendix.A.6).



(a) DD (b) IMDB

Figure 3: **Performance gap between two variants tends to grow with increasing number of propagations** (i.e. iterations) of WL (subsequently paired with LOF), significantly on DD, PROTEINS, and NCI1 (see Appendix.A.6). Similar behavior is observed for the PK-based model.

Observation 3 (Growing gap with propagation)**.** *The difference in ROC-AUC performances on two different down-*

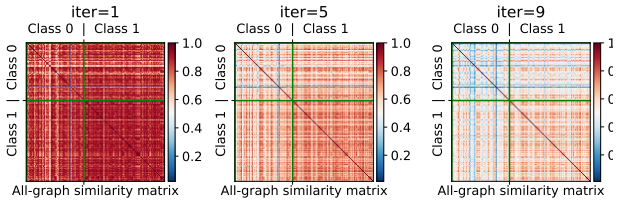


Figure 4: Pairwise similarities among all graphs in DD dataset (graphs grouped by class) based on WL subtree kernel over increasing iterations (left to right).

sampled variants of a classification dataset tends to grow with increasing number of graph propagations for WL- and PK-based outlier models. For OCGIN there exists no obvious growth.

3.2. Hypothesis on Driving Mechanisms

To better understand above peculiar observations, we focus on investigating pairwise similarities – either produced by the graph kernel or the dot product between graph embeddings – normalized in range $[0, 1]$. Figure 4 visualizes pairwise similarity of all graphs on DD dataset using WL kernel, with increasing number of graph propagations (i.e. iterations). The pairwise similarities are block-wise grouped based on the true class label. We route attention to two factors: (1) Diagonal (intra-class) similarities, where we observe that sparsification arises for both classes but the speed at which sparsification occurs is different for the two classes. This leads to growing **density disparity** among two classes. (2) Off-diagonal (inter-class) similarities, which shows that pairwise similarities among graphs within the sparser class (in Fig. 4, class 0 for DD) are *lower* than inter-class similarities on average. This suggests that the class-level distributions of graphs in the embedding space have **overlapping support**.

We conclude by forming the following hypothesis on the driving mechanisms behind "performance flip" and related observations. Provided graph embedding methods employed for outlier detection induce both density disparity as well as overlapping support, one down-sampling scenario creates a dense inlier distribution surrounded with dispersed outliers ('easy' task), whereas the other creates a sparse inlier distribution that has overlapping support with a small set of outliers with relatively higher density ('hard' task). Since most models assume the former scenario in their formalism, they 'do well' on the respective 'easy' task, and poorly on the other (Observation 1). What's more, the effect of propagation based methods on representation densities is way much larger than changing downsampling rate, which results in Observation 2 and observed AUCs-sum-approximately-to-1 behavior (Observation 1's (2)). The issue is exacerbated with more graph propagation as it leads to a growing disparity of densities and respective task difficulties (Observation 3).

3.3. Measures for Analysis

In the previous section we pointed out overlapping support and (growing) density disparity to be two key factors leading to the observed unusual behaviors. Here we introduce concrete measures to quantify these two factors.

1. **Qualitative visualization of both factors:** Multi-dimensional scaling (MDS) is used to map the graph embeddings into 2-dimensions, which can effectively visualize the degree of overlapping support and the speed of sparsification for two classes.
2. **Quantitative measure of density disparity:** The distance between any two graphs from the sparse class would be larger. Therefore, we use the so-called *NN-Radius* to quantify the degree of density.

   Definition 3.1. *NN-Radius is the distance (1-similarity as normalized in range $[0, 1]$) to the k-th[2] nearest neighbor (NN) of a graph in the embedding space.*

   A larger radius corresponds to lower local density. The distribution of the NN-Radius of the graphs from each class measures the density of the class. A more left-shifted distribution would imply a denser class.
3. **Quantitative measure of overlapping support:** In the absence of overlap, assuming graphs from different classes are linearly well-separated in the embedding space (forming two disjoint clusters), we would expect the nearest neighbors of each graph to be from the same class. Therefore, we use the so-called *NN-Disagreement%* as the degree of overlap, defined as follows.

   Definition 3.2. *NN-Disagreement% is the percentage of graphs from the opposite class within the NN-Radius of a graph.*

   We then study the distribution of NN-disagreement% of the graphs from each class. A left- or right-shifted distribution would respectively show whether a class is more likely to be surrounded by its own members (i.e. well-clustered, dense) or not (i.e. dispersed, sparse).

4. Empirical Analysis

In this section, we present further measurement and quantitative analysis to support our hypothesis presented in Sec. 3.2. The experimental setup has been mentioned it Sec. 3.1. The detailed model configuration is summarized in Appendix.A.3. All measures are computed on top of pairwise similarities among all graphs. The detail description of used similarity is presented in Appendix.A.4.

[2]In the paper we report results for $k = 20$ and note that the take-aways are not sensitive to this choice.

4.1. Analysis of Datasets with Performance Flip

As mentioned before, the performance flip and other strange behaviors are observed on 3 datasets: DD, PROTEINS, and NCI1. Because of space limitation, we present analysis on DD dataset and all other results can be found in Appendix.

4.1.1. Analysis on full data.

We start with analyzing the inherent differences between two classes regarding density on DD. Fig. 5 (top row) visualizes the all-pairs similarity matrix based on WL over increasing iterations where sparsification can be observed for both classes (left to right). In the second row, 2-d MDS embeddings of all the graphs based on the corresponding pairwise similarities are shown. Again sparsification can be visually confirmed based on the increasing spread of points (i.e. graphs) in each class. In addition, we notice that class 0 (green points) *sparsify faster* than class 1 (orange points), as the denser class 1 instances are surrounded by the dispersed class 0 instances. We quantify this difference via the *NN-Radius* measure, as in the third row, where the corresponding distributions of *NN-Radius* for all graphs are shown for each class. Over iterations both class distributions shift to the right (i.e. sparsify). The shift is more evident for class 0 (i.e. speed of sparsification is larger) as the (green) histogram spreads out while the other (orange) histogram remains relatively peaked.

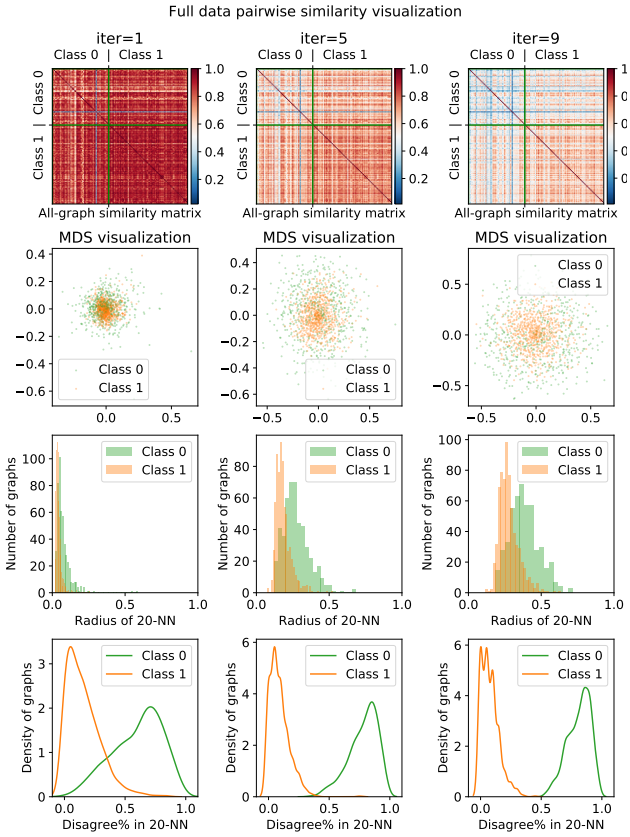


Figure 5: (Top row) Pairwise similarity matrix for all graphs in full DD dataset, based on WL subtree kernel over increasing iterations (left to right). (Second row) 2-d MDS visualization based on the similarity matrix. (Third row) Distribution of *NN-Radius* for all graphs in each class. (Last row) Distribution of *NN-Disagreement%* for all graphs in each class. Similar result is observed for PROTEIN and NCI datasets and other methods, see Appendix.A.7.

Next we analyze the overlapping support between two classes. The mixing of colors in the MDS visualizations is suggestive of overlap.To quantify overlap more concretely, we show the distributions of *NN-Disagreement%* for all graphs in each class in the last row of Fig. 5. The distributions for class 1 concentrate more on the left side (majority of neighbors are from the *same* class) whereas for class 0 they concentrate on the right (majority of neighbors are from the *opposite* class). Both density disparity and overlapping support are increasingly more evident with increasing number of iterations, which suggests that graph propagation amplifies the issue.

The conclusions are similar for PK on DD, as shown in Appendix.A.7. For OCGIN (shown in Fig. 6), while we continue to observe performance flip on DD variants, it is to a lesser degree. Specifically we find that the disparity between classes is not worsened with increasing graph propagation in this case. In fact, the difference in distributions seems to close especially at the last layer. Irrespectively, OCGIN performs considerably better when class 0 is down-sampled like the other models, meaning that it is not shielded from the performance flip issue that we identify. We believe this is due to the initial disparity (See Fig. 6(b) at layer 0, i.e. original input), which it cannot recover from, despite model training.

4.1.2. Analysis upon down-sampling.

Next we analyze the flip in performance upon down-sampling one class or the other via the contrast in *NN-*

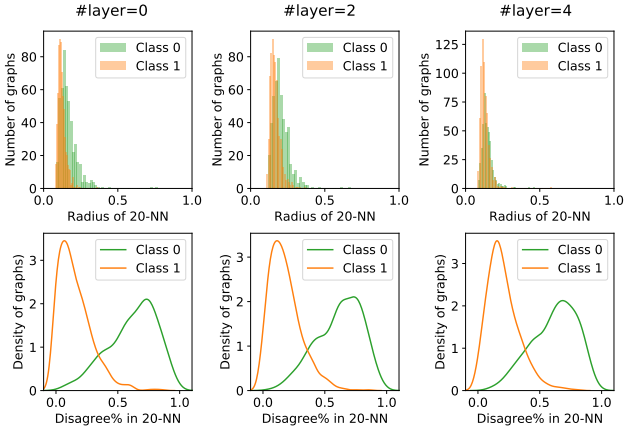


Figure 6: Quantitative measures of density disparity and overlapping support on DD for OCGIN for increasing graph propagation (left to right).

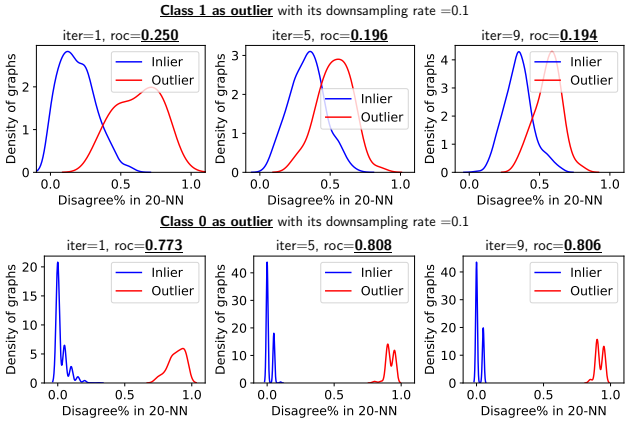


Figure 7: *NN-Disagreement%* distribution of outliers (top row: class 1 is down-sampled and bottom row: class 0 is down-sampled, in red) and inliers (vice versa, in blue) over WL iterations (left to right) on DD.

Disagreement% distributions. Fig. 7 (top) and (bottom) respectively show those distributions when class 1 is down-sampled (denoted Outlier in red) and when class 0 is down-sampled (now denoted Outlier in red) for WL on DD. We notice the stark difference: At the (bottom), the inliers form a dense distribution with negligible mixing with the outliers. The outliers are dispersed, far from one another, as their NN-neighborhood mainly contains inliers. This is the 'easy' detection task that well aligns with the underlying assumption of most outlier detectors – hence the high peformance of WL+LOF.

In contrast, the (top) figures suggest that the inliers and outliers are intermixed, which worsens with propagation as the distributions become more and more indistinguishable. (Note that down-sampled class inherently has a right-shifted distribution as down-sampling induces sparsity.) This corresponds to the 'hard' detection task where it is difficult to distinguish inliers from outliers – hence the poor, in fact worse-than-random performance.

The conclusions are similar for WL+LOF on PROTEINS and NCI1, respectively. In fact, performance appears to be inversely correlated with the amount of overlap between the *NN-Disagreement%* distributions of inliers and outliers. We refer to the Appendix.A.8 for similar results on these three datasets for PK+LOF and OCGIN.

4.2. Analysis of Dataset without Performance Flip

Although the performance flip issue occured on a considerable number of datasets we have experimented with, it is not always observed. In this section, we present a similar analysis for IMDB-BINARY for comparison purposes based on WL, analysis for other methods can be found in Appendix.A.9.

As shown in Fig. 8, sparsification arises rather fast on

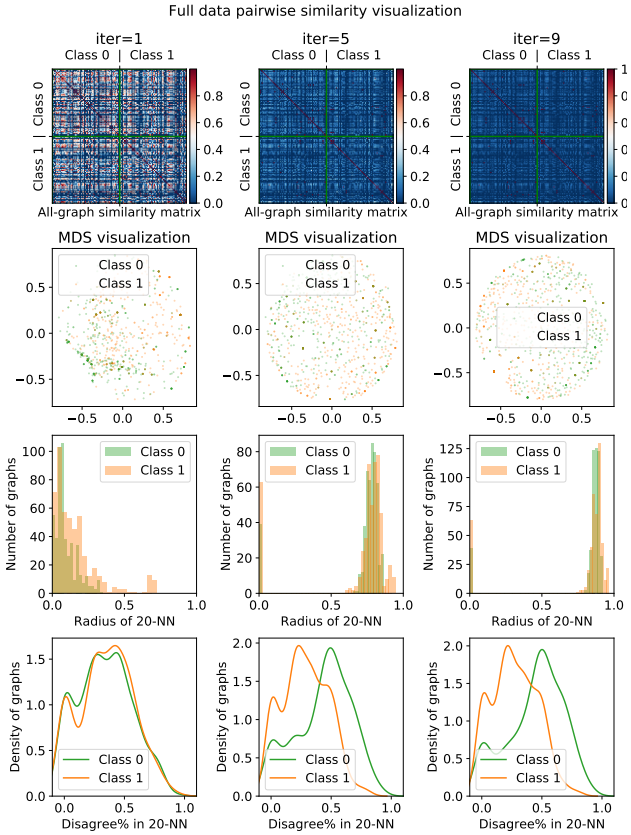


Figure 8: (Top row) Pairwise similarity matrix for all graphs in full IMDB-BINARY dataset, based on WL subtree kernel over increasing iterations (left to right). (Second row) 2-d MDS visualization based on the similarity matrix. (Third row) Distribution of *NN-Radius* for all graphs in each class. (Last row) Distribution of *NN-Disagreement%* for all graphs in each class.

IMDB-BINARY with increasing WL iterations – notice the drastic shift of the *NN-Radius* distributions from left-most to right-most (third row). Interestingly, the rate (or speed) of sparsification appears to be similar among the two classes. As such, density disparity does not seem to arise – notice the similar mixing among the colored points in 2-d MDS visualization (second row). On the other hand, we continue to observe the overlapping support (i.e. mixing) of graph embeddings to a large extent (last row).

These suggest that down-sampling any one of the classes as outlier would induce two detection tasks with similar difficulty. Fig. 9 confirms this hypothesis, where the distribution of inliers and outliers in the embedding space look similar between the two down-sampled variants of IMDB-BINARY. Moreover, the outliers are sparser and more dispersed as compared to the inliers, which consequently leads to better-than-random performance on both tasks. The ROC-AUC values are not too high due to the mixing of the embeddings that makes the detection task harder. In Sec. 3.2 we argued that density disparity alone is not a sufficient condition for performance flip. This result on IMDB-BINARY shows that overlapping support (i.e. mixing) alone is also not a suffi-

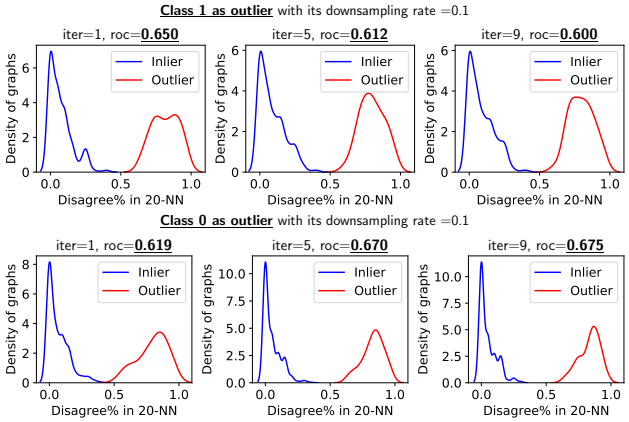


Figure 9: *NN-Disagreement%* distribution of outliers (top row: class 1 is down-sampled and bottom row: class 0 is down-sampled, in red) and inliers (vice versa, in blue) over WL iterations (left to right) on IMDB-BINARY.

cient condition for the performance flip. Both conditions together give rise to the issue.

5. Discussion of the Findings

The findings in our paper are studied in the context of graph-level outlier detection, which stands at the intersection of graph representation learning and outlier detection. On one hand, performance flip becomes an obstacle for effectively solving the graph-level outlier detection problem. On the other hand, it also raises a 'red flag' regarding potentially broader problems in these two widely studied fields: outlier detection and graph representation learning. We mainly talk about graph representation learning, and put discussion of outlier detection in Appendix.A.10.

To the best of our knowledge, our findings have not been reported in any prior work. Arguably, we believe that the issues we have identified have broader implications for graph classification and graph-level clustering using graph neural networks, and the potential to open new directions of future work in both fields. The authors of the Graph Isomorphism Network (GIN) (Xu et al., 2019) have claimed GIN to be "the most expressive [architecture] among the class of GNNs" because it can be as discriminative as the WL graph isomorphism test. We challenge this viewpoint: the discriminating ability of the WL test may not be suitable to be built into graph representation learning when used for tasks that expect well-clustered data in the feature space. In particular, (graph-level) clustering would work well for well-clustered data with clear cluster boundaries – which would be adversely affected by the mixing (i.e. overlap) and sparsification issues we have identified. In addition, graph classification would work well provided distinguishable class manifolds that are densely sampled from (to be able to learn discriminative class boundaries) – for which mixing and sparsity would be deteriorating, where a small amount of sparsely sampled training data from a class would not be representative of the support region of the class and would lead to poor generalization. We believe the conflict in viewpoints arises as we draw attention to graph-level tasks whereas a vast body of prior work on representation learning, including GIN, focused on *node-level* tasks (i.e. node embedding). More research is called for the expressive power of various *graph* embedding techniques.

6. Conclusion

In this work we identified the (what we call) *performance flip* issue with propagation based model, including graph kernel and graph neural network based models, on GLOD problem evaluated with repurposing binary graph classification datasets. While down-sampling one class to constitute the outliers yields relatively high ROC-AUC performance for the models, down-sampling the other yields significantly worse (in fact, worse than random) performance, irrespective of the down-sampling rate. The performances on these two variants approximately sum to 1, implying one of the classes is always deemed more outlying irrespective of *which* one is down-sampled.

Through careful analysis, we find that the driving factors behind the issue are two: (1) disparity between class-level densities (or within-class sample similarity), and (2) overlapping support of the density distributions. We also find specifically for kernel based models that graph propagation, and the associated sparsification property, is a contributing factor that amplifies the disparity and ultimately the performance gap.

Our findings have implications for the fair and effective evaluation of outlier models. Our analysis provides tools for better understanding the datasets used for benchmark evaluation. We also shed light onto the effect of graph propagation on the distribution of graph embeddings, which we argued to likely have further implications for graph classification and clustering tasks. To facilitate future research on these and related issues, we will release a deep graph-level anomaly detection library in future.

References

Breunig, M. M., Kriegel, H.-P., Ng, R. T., and Sander, J. Lof: identifying density-based local outliers. In *Proceedings of the 2000 ACM SIGMOD international conference on Management of data*, pp. 93–104, 2000.

Chalapathy, R. and Chawla, S. Deep learning for anomaly detection: A survey, 2019. URL http://arxiv.org/abs/1901.03407. cite arxiv:1901.03407.

Ding, K., Li, J., Bhanushali, R., and Liu, H. Deep anomaly detection on attributed networks. In *Proceedings of the*

2019 SIAM International Conference on Data Mining, pp. 594–602. SIAM, 2019.

Gionis, A., Indyk, P., and Motwani, R. Similarity search in high dimensions via hashing. In *The VLDB Journal*, pp. 518–529, 1999. URL http://citeseer.ist.psu.edu/203242.html.

Harshaw, C. R., Bridges, R. A., Iannacone, M. D., Reed, J. W., and Goodall, J. R. Graphprints: Towards a graph analytic method for network anomaly detection. In *Proceedings of the 11th Annual Cyber and Information Security Research Conference*, pp. 1–4, 2016.

He, J. *Analysis of Rare Categories*. Cognitive Technologies. Springer, 2012. ISBN 978-3-642-22812-4. doi: 10.1007/978-3-642-22813-1. URL https://doi.org/10.1007/978-3-642-22813-1.

Kingma, D. P. and Ba, J. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

Kipf, T. N. and Welling, M. Semi-supervised classification with graph convolutional networks. In *International Conference on Learning Representations (ICLR)*, 2017.

Liu, C., Yan, X., Yu, H., Han, J., and Yu, P. S. Mining behavior graphs for "backtrace" of noncrashing bugs. In *Proceedings of the 2005 SIAM International Conference on Data Mining*, pp. 286–297. SIAM, 2005.

Manevitz, L. M. and Yousef, M. One-class svms for document classification. *Journal of machine Learning research*, 2(Dec):139–154, 2001.

Manzoor, E. A., Milajerdi, S. M., and Akoglu, L. Fast memory-efficient anomaly detection in streaming heterogeneous graphs. In *KDD*, pp. 1035–1044. ACM, 2016. URL http://dblp.uni-trier.de/db/conf/kdd/kdd2016.html#ManzoorMA16.

Markovitz, A., Sharir, G., Friedman, I., Zelnik-Manor, L., and Avidan, S. Graph embedded pose clustering for anomaly detection. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 10539–10547, 2020.

Monti, F., Frasca, F., Eynard, D., Mannion, D., and Bronstein, M. M. Fake news detection on social media using geometric deep learning. *arXiv preprint arXiv:1902.06673*, 2019.

Morris, C., Kriege, N. M., Bause, F., Kersting, K., Mutzel, P., and Neumann, M. TUdataset: A collection of benchmark datasets for learning with graphs. *arXiv preprint arXiv:2007.08663*, 2020.

Narayanan, A., Chandramohan, M., Chen, L., Liu, Y., and Saminathan, S. subgraph2vec: Learning distributed representations of rooted sub-graphs from large graphs. *arXiv preprint arXiv:1606.08928*, 2016.

Narayanan, A., Chandramohan, M., Venkatesan, R., Chen, L., Liu, Y., and Jaiswal, S. graph2vec: Learning distributed representations of graphs. *arXiv preprint arXiv:1707.05005*, 2017.

Neumann, M., Garnett, R., Bauckhage, C., and Kersting, K. Propagation kernels: efficient graph kernels from propagated information. *Machine Learning*, 102(2):209–245, 2016.

Nguyen, H. T., Liang, P. J., and Akoglu, L. Anomaly detection in large labeled multi-graph databases, 2020.

Pang, G., Shen, C., Cao, L., and van den Hengel, A. Deep learning for anomaly detection: A review. *CoRR*, abs/2007.02500, 2020. URL http://dblp.uni-trier.de/db/journals/corr/corr2007.html#abs-2007-02500.

Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., Blondel, M., Prettenhofer, P., Weiss, R., Dubourg, V., Vanderplas, J., Passos, A., Cournapeau, D., Brucher, M., Perrot, M., and Duchesnay, E. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.

Ruff, L., Vandermeulen, R., Goernitz, N., Deecke, L., Siddiqui, S. A., Binder, A., Müller, E., and Kloft, M. Deep one-class classification. In *International conference on machine learning*, pp. 4393–4402, 2018.

Ruff, L., Kauffmann, J. R., Vandermeulen, R. A., Montavon, G., Samek, W., Kloft, M., Dietterich, T. G., and Müller, K.-R. A unifying review of deep and shallow anomaly detection. *CoRR*, abs/2009.11732, 2020. URL http://dblp.uni-trier.de/db/journals/corr/corr2009.html#abs-2009-11732.

Shervashidze, N., Schweitzer, P., Van Leeuwen, E. J., Mehlhorn, K., and Borgwardt, K. M. Weisfeiler-lehman graph kernels. *Journal of Machine Learning Research*, 12(9), 2011.

Verma, S. and Zhang, Z.-L. Hunt for the unique, stable, sparse and fast feature learning on graphs. In *Advances in Neural Information Processing Systems*, pp. 88–98, 2017.

Wang, X., Du, Y., Cui, P., and Yang, Y. Ocgnn: One-class classification with graph neural networks. *arXiv preprint arXiv:2002.09594*, 2020.

Weber, M., Domeniconi, G., Chen, J., Weidele, D. K. I., Bellei, C., Robinson, T., and Leiserson, C. E. Anti-money

laundering in bitcoin: Experimenting with graph convolutional networks for financial forensics, 2019.

Weisfeiler, B. and Leman, A. The reduction of a graph to canonical form and the algebgra which appears therein. *NTI, Series*, 2, 1968.

Xu, K., Hu, W., Leskovec, J., and Jegelka, S. How powerful are graph neural networks? In *ICLR*. OpenReview.net, 2019. URL http://dblp.uni-trier.de/db/conf/iclr/iclr2019.html#XuHLJ19.

Yanardag, P. and Vishwanathan, S. Deep graph kernels. In *Proceedings of the 21th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pp. 1365–1374, 2015.

Zhang, M., Cui, Z., Neumann, M., and Chen, Y. An end-to-end deep learning architecture for graph classification. In *AAAI*, volume 18, pp. 4438–4445, 2018.

Zhu, X. and Ghahramani, Z. Learning from labeled and unlabeled data with label propagation. 2002.

A. Supplementary Material

A.1. Propagation Kernel and Sparsification Issue

Propagation kernel (PK) (Neumann et al., 2016) is inspired from the idea of propagating label information among nodes over the graph structure such as label propagation algorithm (Zhu & Ghahramani, 2002) for semi-supervised node classification and can be used for both attributed graphs and one-hot encoded labeled graphs. For each graph $G = (\mathcal{V}, \mathcal{E}, \mathbf{X})$, let $X_0 = \mathbf{X}$ denote the original feature matrix. Then PK generates a new feature matrix at each iteration by propagating the feature matrix using the transition matrix $T = D^{-1}A$ (where A is the adjacency matrix and D is the diagonal degree matrix) of the graph. Formally, $\mathbf{X}_{l+1} = T\mathbf{X}_l$. Similar to WL subtree kernel, PK compares two graphs at each iteration. The similarity between two graphs is measured based on propagated features through binning. Formally we can write the kernel as

$$\mathcal{K}_{PK}(G, G') = \sum_{l=0}^{L} \langle \phi_{PK}(\mathbf{X}_l^G), \phi_{PK}(\mathbf{X}_l^{G'}) \rangle \tag{4}$$

where $\phi_{PK}(\cdot)$ denotes the hash function that maps a given set of (feature) vectors into bins. To preserve locality and keep efficiency, locally sensitive hashing (LSH) (Gionis et al., 1999) is used for the binning.

Sparsification. The propagation kernel also exhibits the aforementioned sparsity problem, increasingly for larger number of iterations. Compared to WL subtree kernel that generates new features via re-labeling (a hard transformation), propagation kernel generates new features via multiplying by the transition matrix (a soft transformation). As such, the feature space grows much slower for PK. As illustrated in Fig. 10, the diagonal dominance still holds while the sparsification occurs at a lower rate than WL subtree kernel (cf. Fig. 1).

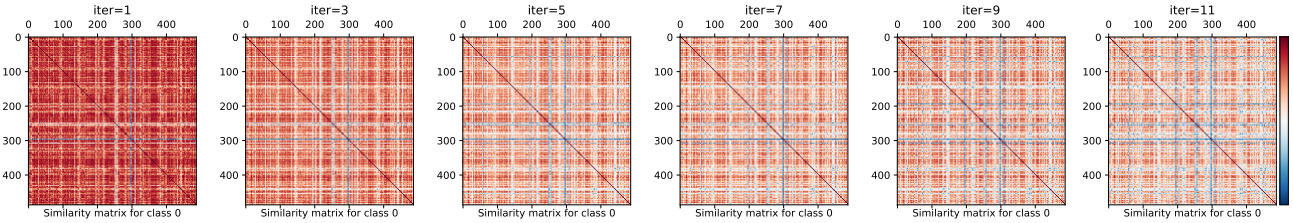


Figure 10: Sparsification in PK: Pairwise similarity of graphs (from DD dataset) decreases with increasing number of iterations (left to right).

A.2. Data Statistics

All datasets used in the experiments are repurposed from binary graph classification datasets, their statistics are summarized in Table 2.

Table 2: Dataset summary statistics.

Dataset	GraphType	Class	#Graphs	#NodeLabels	Avg. #Nodes	Avg. #Edges	Avg. Degree
DD	Labeled	0	691	89	355.2	1806.6	5.04
		1	487	89	183.7	898.8	4.88
PROTEINS	Labeled	0	663	3	50	188.1	3.79
		1	450	3	22.9	83.1	3.64
NCI1	Labeled	0	2053	37	25.65	55.3	2.15
		1	2057	37	34.07	73.9	2.17
IMDB	Degree-Labeled	0	500	136	20.1	193.5	9.1
		1	500	136	19.4	192.5	8.6

A.3. Model Configuration

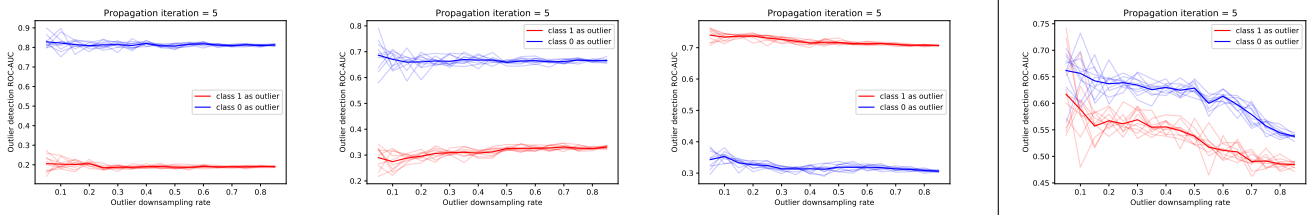
Our experiments are based on two-stage models WL+LOF and PK+LOF, as well as end-to-end deep-one-class model OCGIN. We study the behavior of these models under different number of propagations; specifically WL and PK iterations range from 1 to 11, and OCGIN embeddings are extracted from layers 0 (i.e. input node vectors) through 5. PK has a

bin-width hyperparameter for hash function ϕ_{PK} (See Eq. 4), which is set to 0.1. Note that smaller bin-width leads to faster sparsification with a more severe performance flip. The LOF outlier detector is setup with default parameters (k =20 number of neighbors, leaf size 30) from scikit-learn (Pedregosa et al., 2011). For OCGIN we use the default GIN implementation from (Xu et al., 2019), where we remove bias terms at all layers to prevent feature collapse (Ruff et al., 2018). A graph's representation is produced from the summation of all previous layers' hidden representations, with a mean pooling over all nodes in the graph. Note that we train OCGIN only on down-sampled variants of a dataset, as it is trained end-to-end assuming outliers to be minority. For figures utilizing full data, we simply feed-forward all the graphs in the database over the *trained* model. We set number of layers to $L = 5$ and number of hidden units to 128 for all datasets. We use the Adam optimizer (Kingma & Ba, 2014) to train OCGIN with a $5 \cdot 10^{-4}$ L_2 penalty on weights. The model is trained for 25 epochs. All other hyperparameters are picked from typical/default values, since our goal is to illustrate the performance flip and related issues instead of achieving best performance. Hyperparameter selection for unsupervised deep outlier detection is an important problem which is outside the scope of this paper.

A.4. Pairwise Similarity

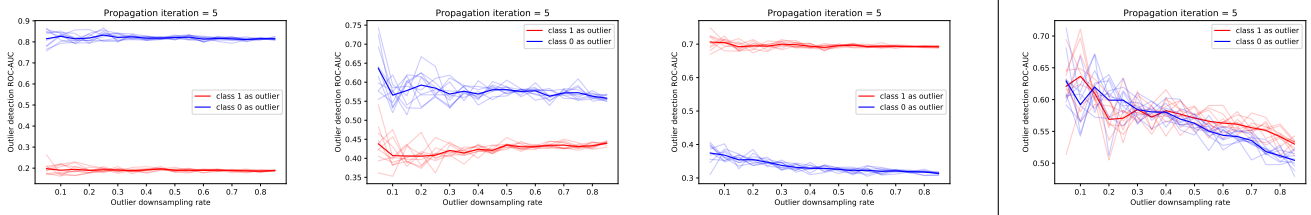
Our proposed measures in Sec. 3.3 are computed on top of pairwise similarities among all graphs. For PK and WL kernels, the normalized kernel matrix is investigated. For OCGIN, where we only have access to graph embeddings, we calculate pairwise similarity as (1 - normalized pairwise distance) between two graphs, using Euclidean distance (i.e. L_2 norm). Similarity matrix is normalized to range $[0, 1]$ via dividing it by the largest element in the matrix.

A.5. Additional Results for Observation 2



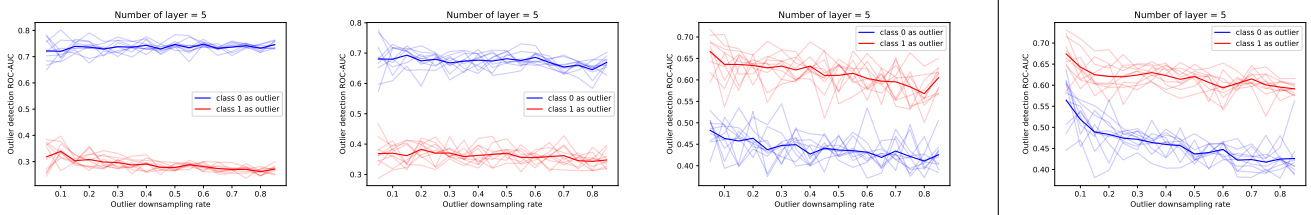
(a) DD (b) PROTEINS (c) NCI1 (d) IMDB

Figure 11: Full version of Figure 2, performance is invariant to downsampling rate on DD, PROTEINS, and NCI1 for **WL+LOF**.



(a) DD (b) PROTEINS (c) NCI1 (d) IMDB

Figure 12: Performance is invariant to downsampling rate on DD, PROTEINS, and NCI1 for **PK+LOF**.

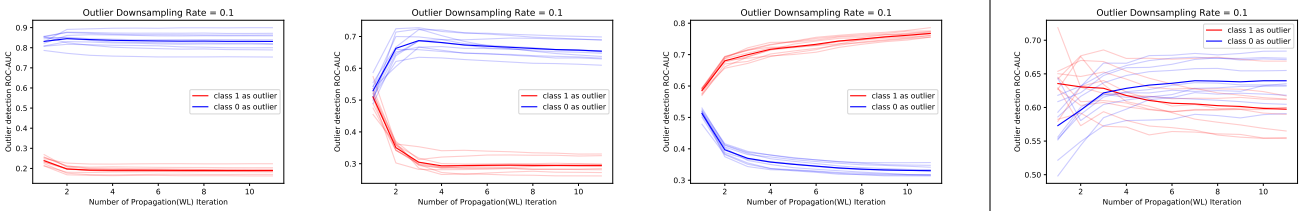


(a) DD (b) PROTEINS (c) NCI1 (d) IMDB

Figure 13: Performance is invariant to downsampling rate on DD, PROTEINS, and NCI1 for **OCGIN**.

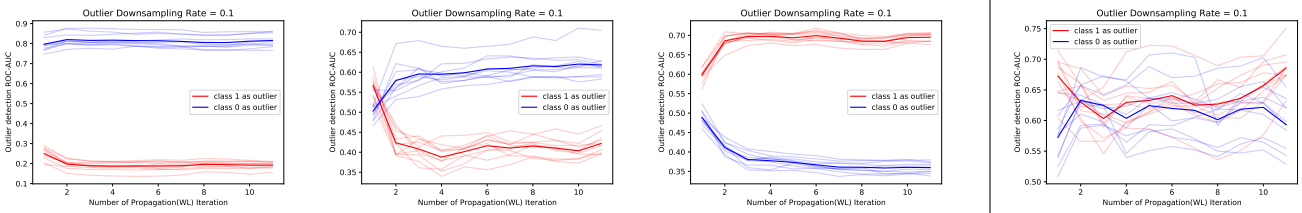
A.6. Additional Results for Observation 3

We have observed that performance gap between two variants of repurposed graph classification datasets tends to grow with increasing number of propagations on WL (Figure 14) and PK (Figure 15) in DD, PROTEINS, and NCI1, while not for IMDB. However this observation does not hold for OCGIN, which could because of the training procedure (shown in Figure 16).



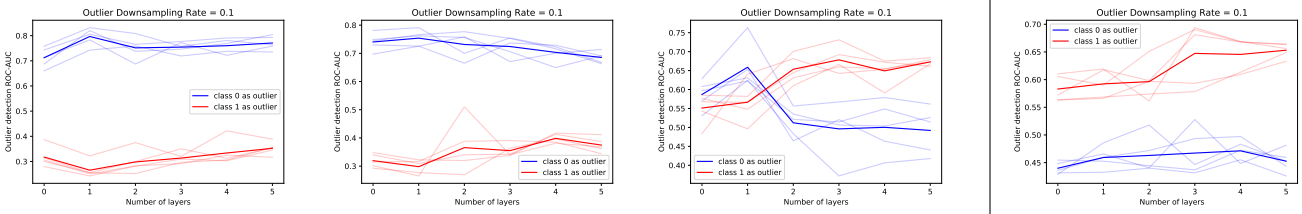
(a) DD (b) PROTEINS (c) NCI1 (d) IMDB

Figure 14: Full version of Figure 3, performance gap between two variants tends to grow with increasing number of propagations (i.e. iterations) of **WL** (subsequently paired with LOF), significantly on DD, PROTEINS, and NCI1.



(a) DD (b) PROTEINS (c) NCI1 (d) IMDB

Figure 15: Performance gap between two variants tends to grow with increasing number of propagations (i.e. iterations) of **PK** (subsequently paired with LOF), significantly on DD, PROTEINS, and NCI1.



(a) DD (b) PROTEINS (c) NCI1 (d) IMDB

Figure 16: Performance gap between two variants of datasets for **OGCIN**

A.7. Additional Measures on DD, PROTEIN and NCI

A.7.1. DD

We show the designed measures for pairwise similarity of all graphs in DD dataset, using WL kernel (Figure 17), PK kernel (Figure 18), and OCGIN (Figure 19).

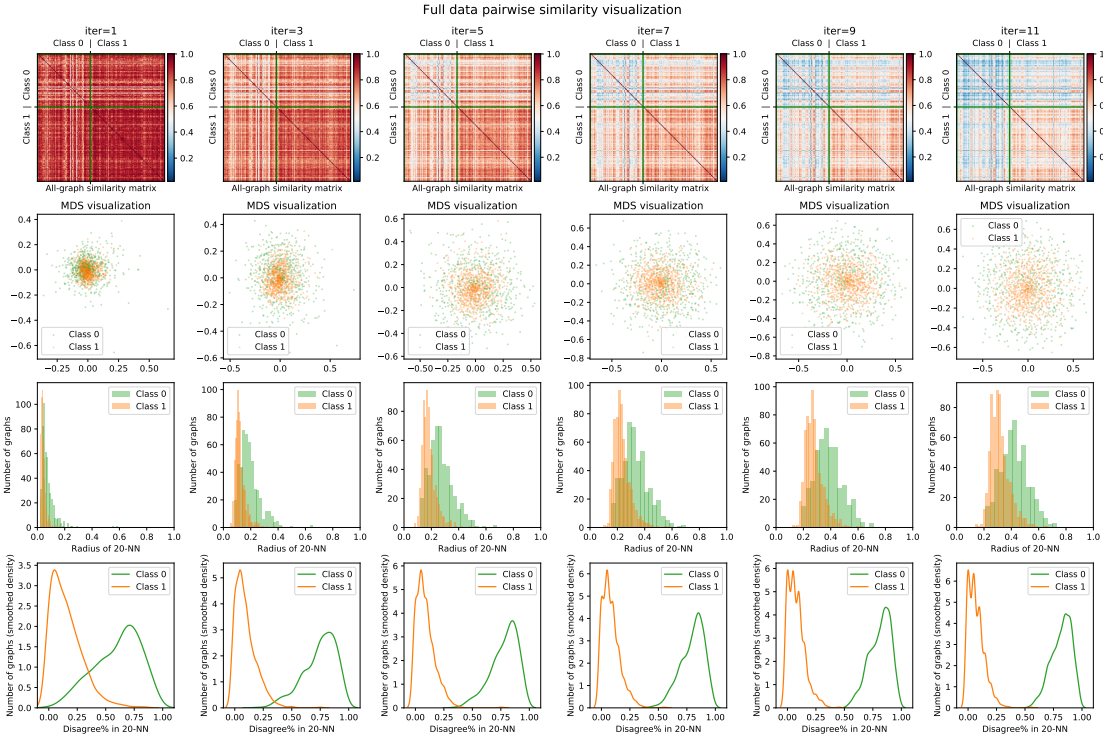


Figure 17: Measures for full pairwise similarity of all graphs on DD dataset with **WL+LOF**

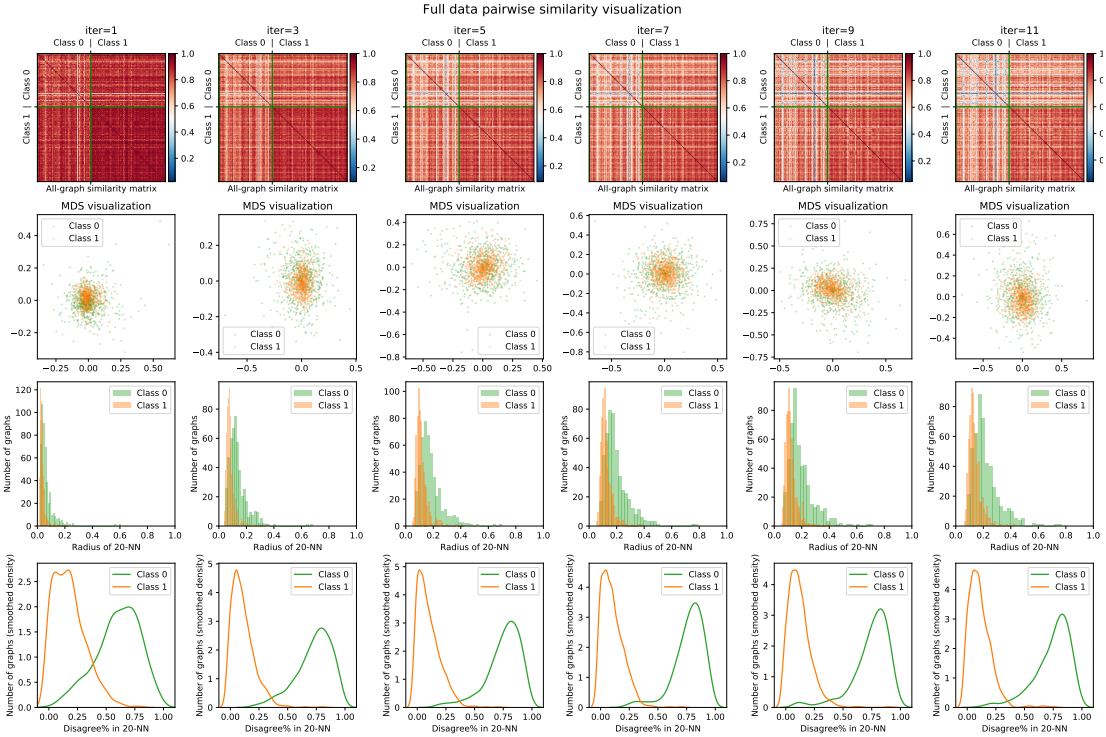


Figure 18: Measures for full pairwise similarity of all graphs on DD dataset with **PK+LOF**

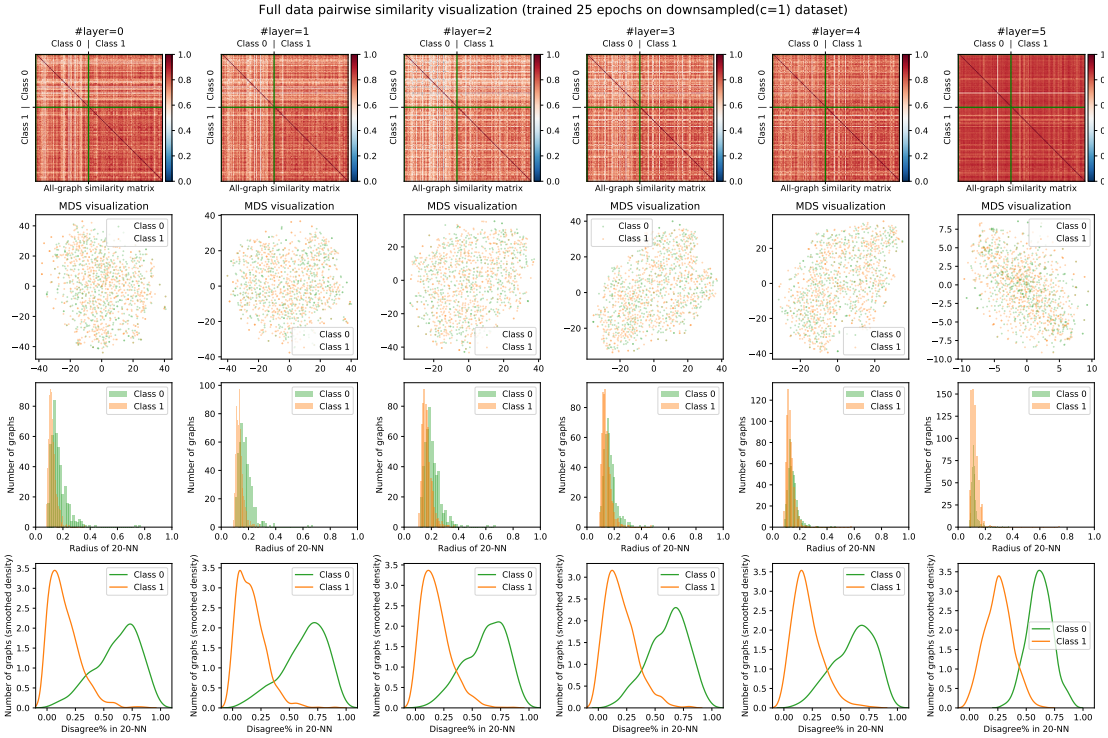


Figure 19: Measures for full pairwise similarity of all graphs on DD dataset with **OCGIN** (trained for 25 epochs)

A.7.2. PROTEINS

We show the designed measures for pairwise similarity of all graphs in PROTEINS dataset, using WL kernel (Figure 20), PK kernel (Figure 21), and OCGIN (Figure 22).

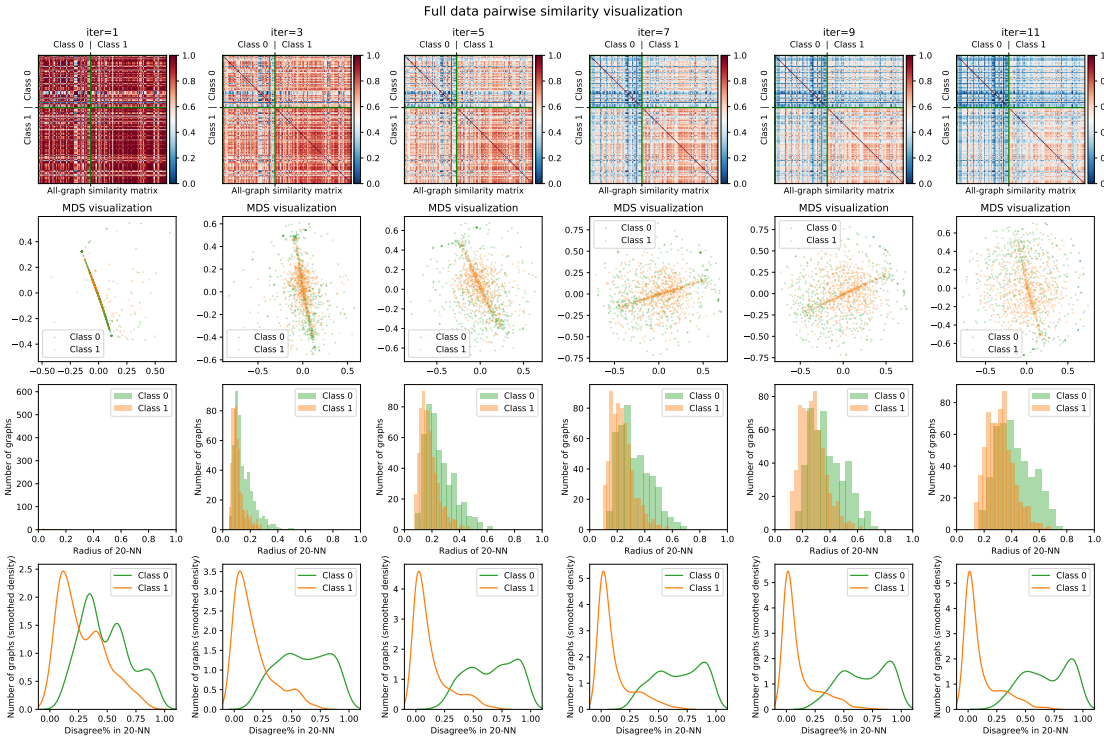


Figure 20: Measures for full pairwise similarity of all graphs on PROTEINS dataset with **WL+LOF**

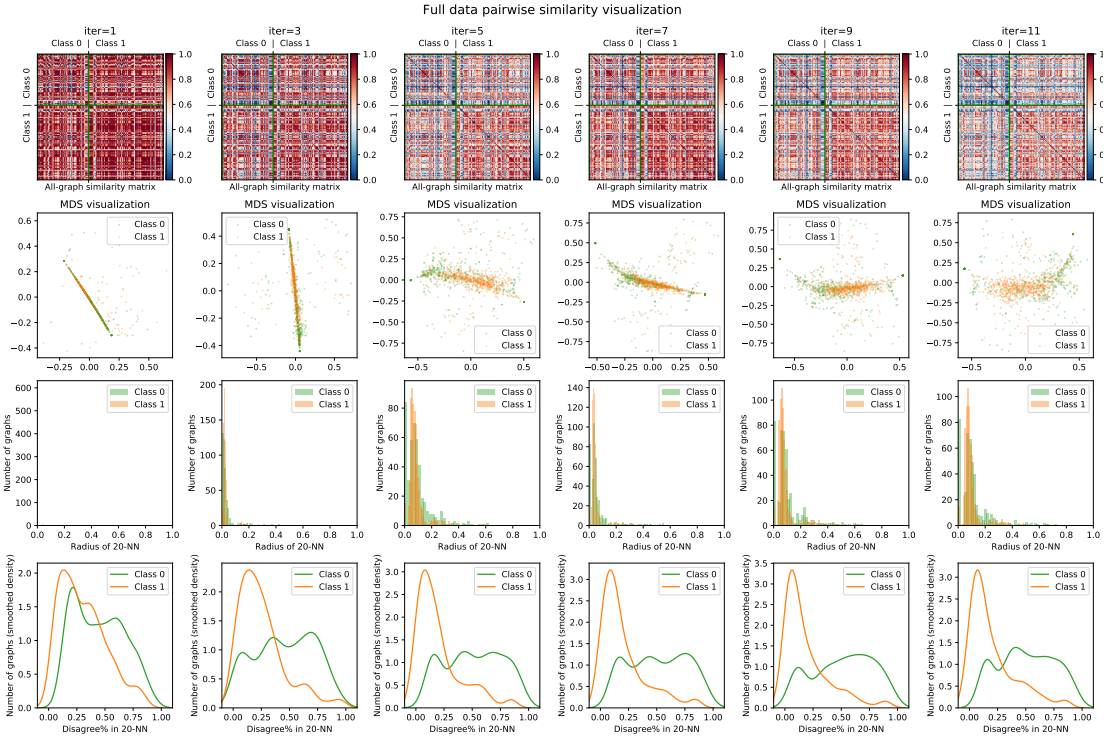


Figure 21: Measures for full pairwise similarity of all graphs on PROTEINS dataset with **PK+LOF**

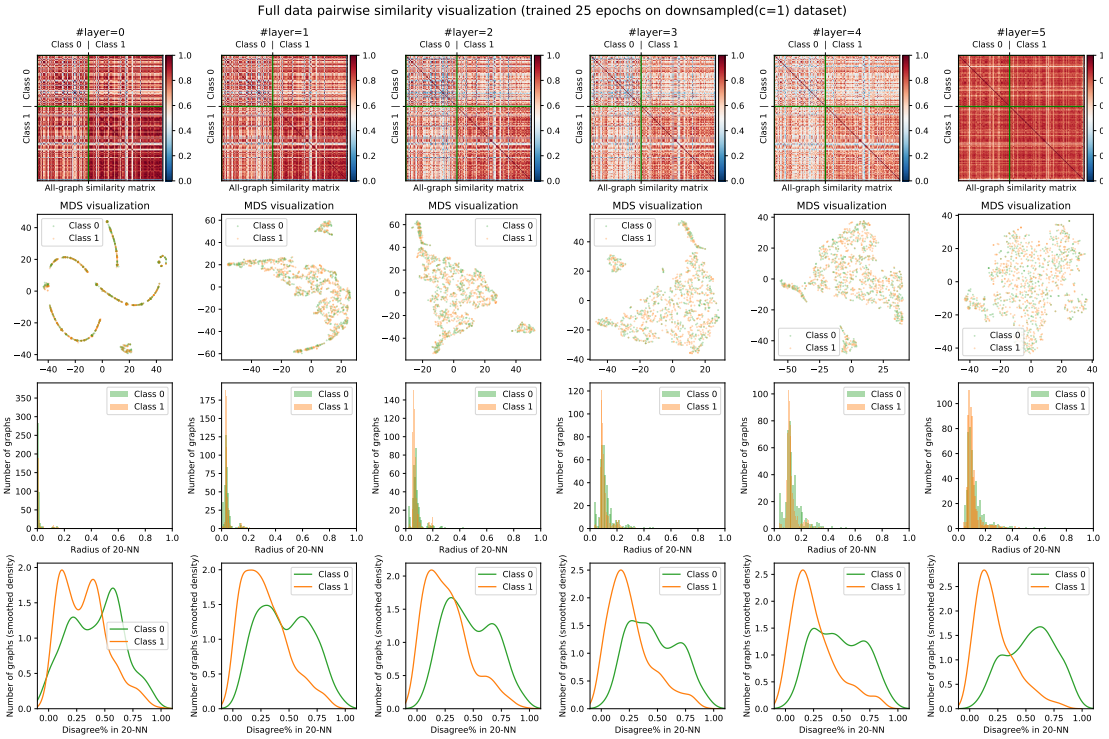


Figure 22: Measures for full pairwise similarity of all graphs on PROTEINS dataset with **OCGIN** (trained for 25 epochs)

A.7.3. NCI1

We show the designed measures for pairwise similarity of all graphs in NCI1 dataset, using WL kernel (Figure 23), PK kernel (Figure 24), and OCGIN (Figure 25).

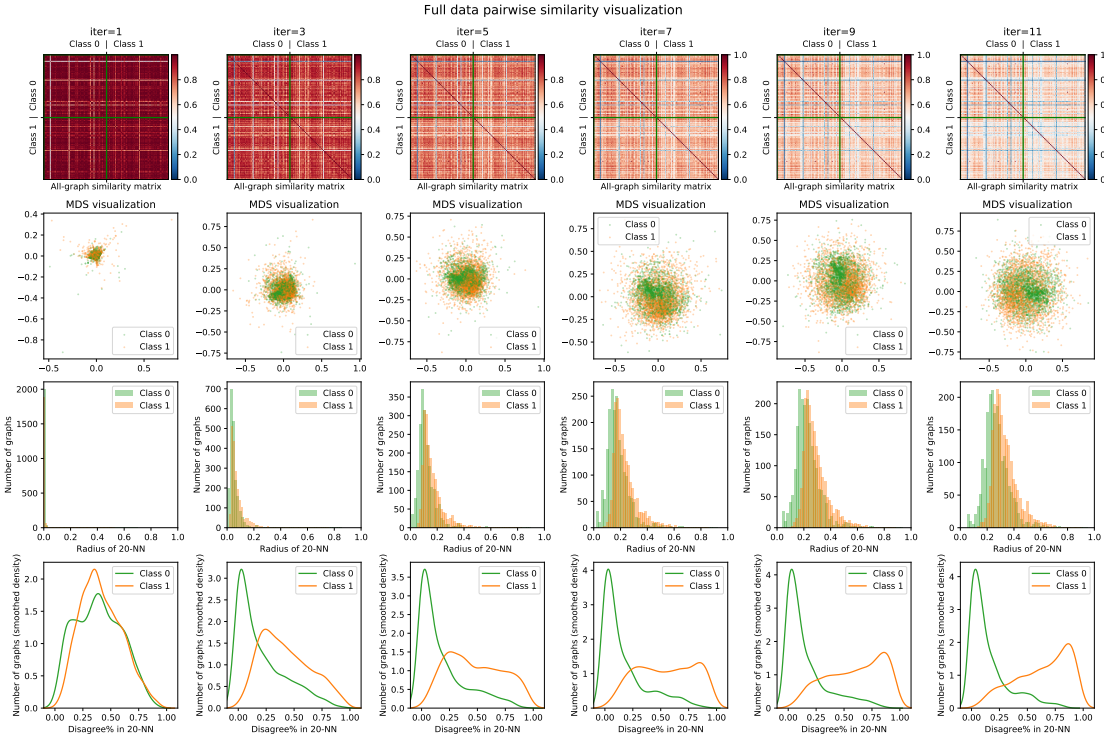


Figure 23: Measures for full pairwise similarity of all graphs on NCI1 dataset with **WL+LOF**

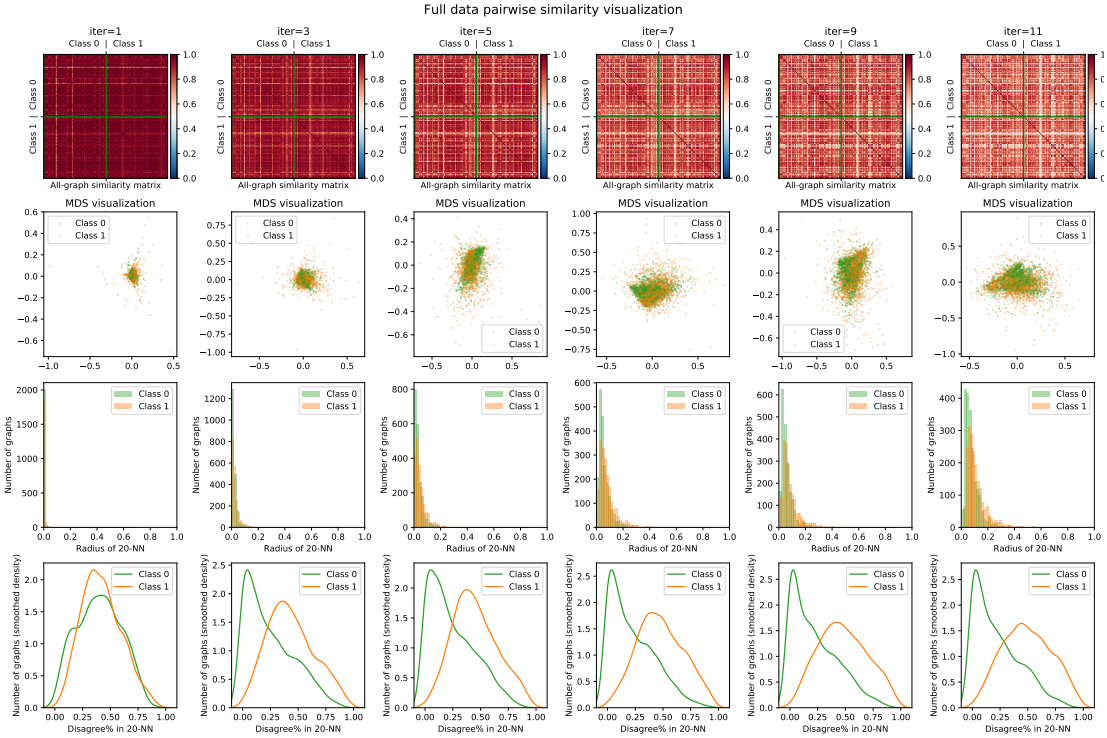


Figure 24: Measures for full pairwise similarity of all graphs on NCI1 dataset with **PK+LOF**

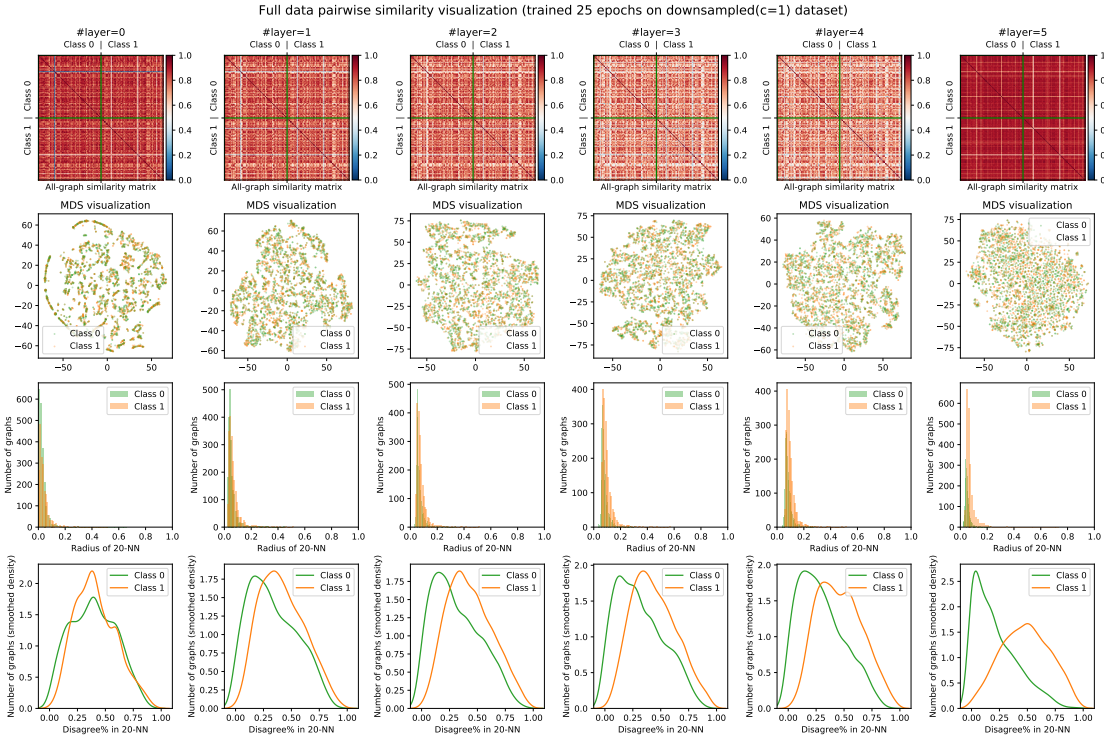


Figure 25: Measures for full pairwise similarity of all graphs on NCI1 dataset with **OCGIN** (trained for 25 epochs)

A.8. Additional Measures on Downsampled DD, PROTEIN and NCI

A.8.1. DD

We analyze the pairwise similarity of graphs on two variants of down-sampled DD dataset, by showing the distribution of *NN-Disagreement%* (measures shown in sec.3.3) and corresponding ROC-AUC. The analysis is conducted for three methods: WL+LOF (Figure 26), PK+LOF (Figure 27), and OCGIN (Figure 28).

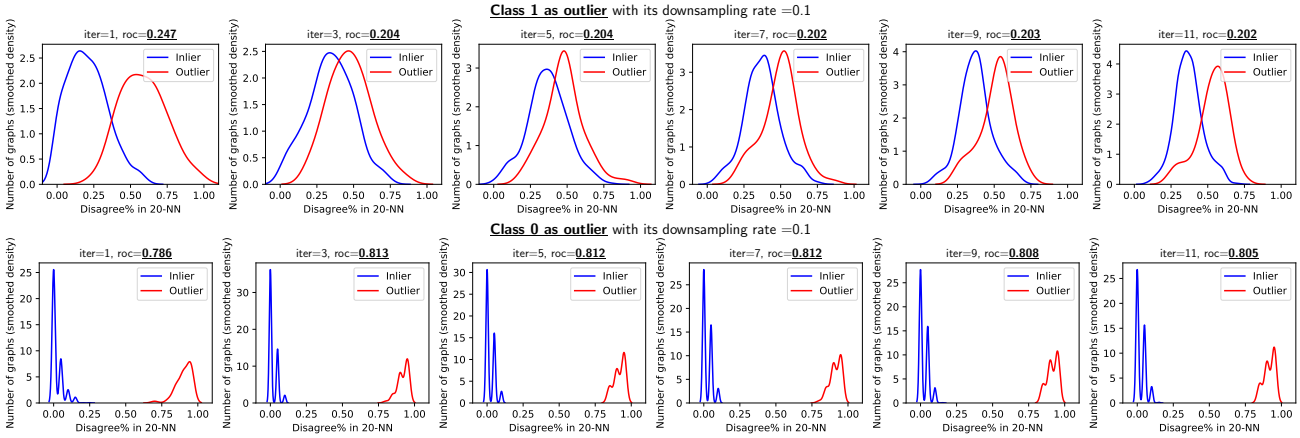


Figure 26: *NN-Disagreement%* distribution of outliers (top row: class 1 is down-sampled and bottom row: class 0 is down-sampled, in red) and inliers (vice versa, in blue) over iterations (left to right) on DD with **WL+LOF**.

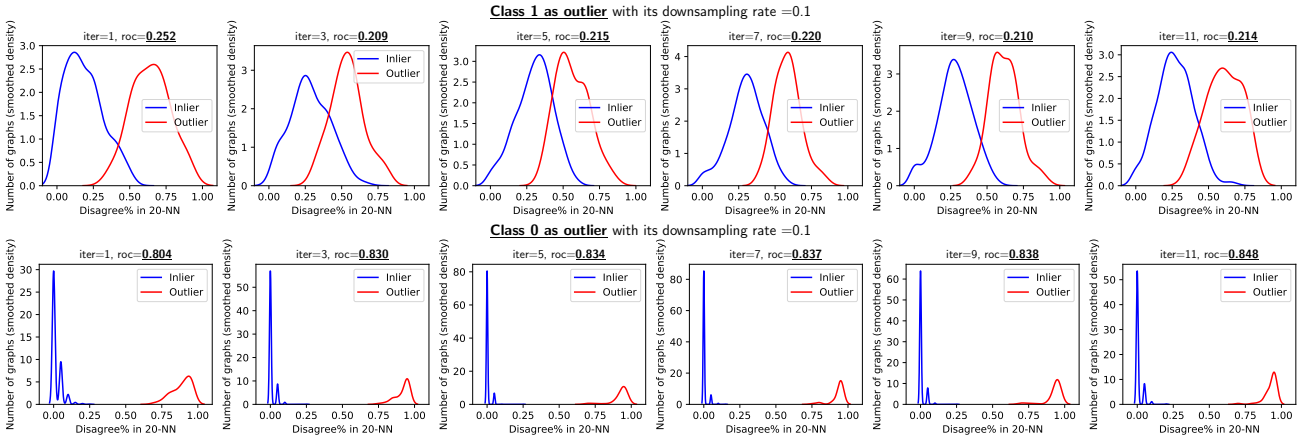


Figure 27: *NN-Disagreement%* distribution of outliers (top row: class 1 is down-sampled and bottom row: class 0 is down-sampled, in red) and inliers (vice versa, in blue) over iterations (left to right) on DD with **PK+LOF**.

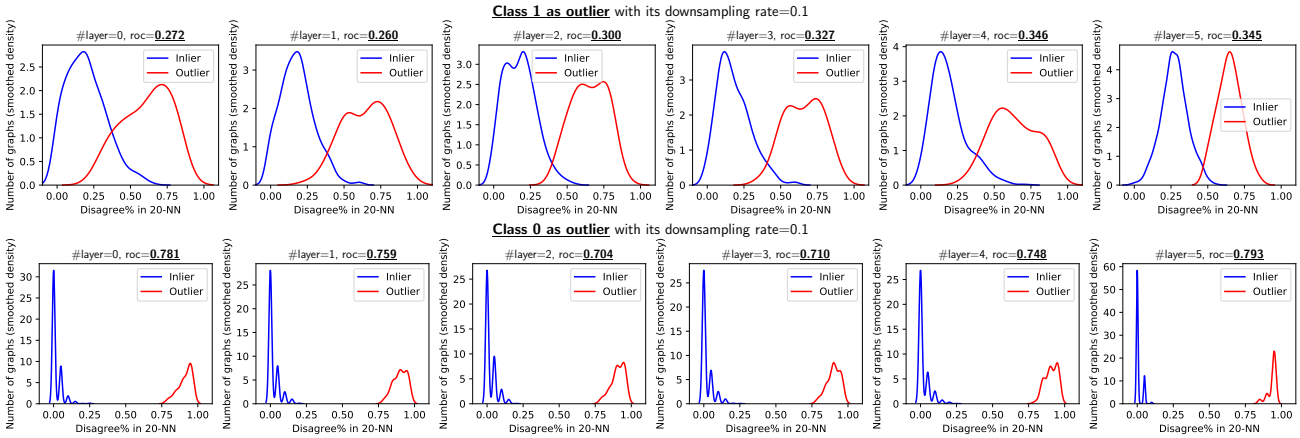


Figure 28: *NN-Disagreement%* distribution of outliers (top row: class 1 is down-sampled and bottom row: class 0 is down-sampled, in red) and inliers (vice versa, in blue) over iterations (left to right) on DD with **OCGIN**.

A.8.2. PROTEINS

We analyze the pairwise similarity of graphs on two variants of down-sampled PROTEINS dataset, by showing the distribution of *NN-Disagreement%* (measures shown in sec.3.3) and corresponding ROC-AUC. The analysis is conducted for three methods: WL+LOF (Figure 29), PK+LOF (Figure 30), and OCGIN (Figure 31).

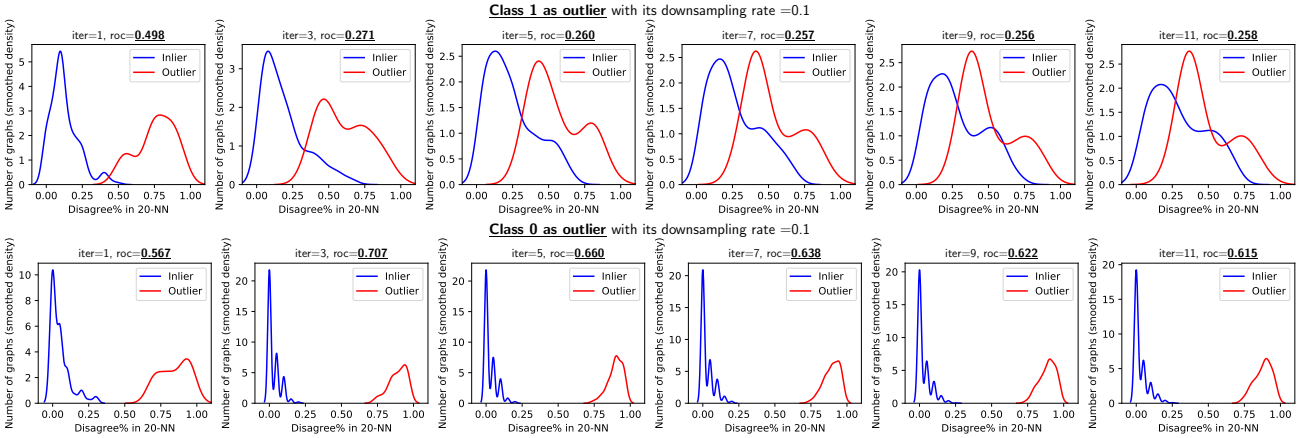


Figure 29: *NN-Disagreement%* distribution of outliers (top row: class 1 is down-sampled and bottom row: class 0 is down-sampled, in red) and inliers (vice versa, in blue) over iterations (left to right) on PROTEINS with **WL+LOF**.

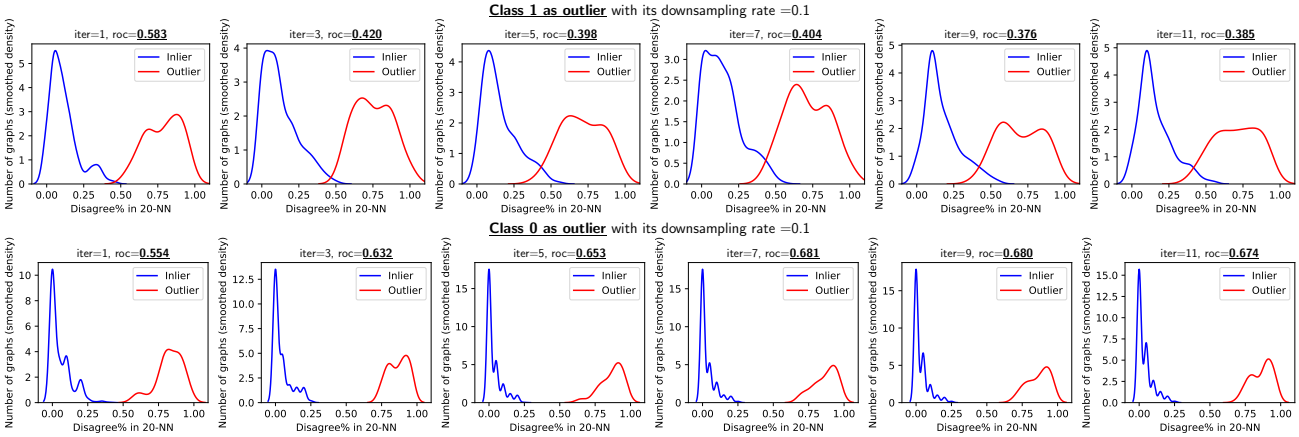


Figure 30: *NN-Disagreement%* distribution of outliers (top row: class 1 is down-sampled and bottom row: class 0 is down-sampled, in red) and inliers (vice versa, in blue) over iterations (left to right) on PROTEINS with **PK+LOF**.

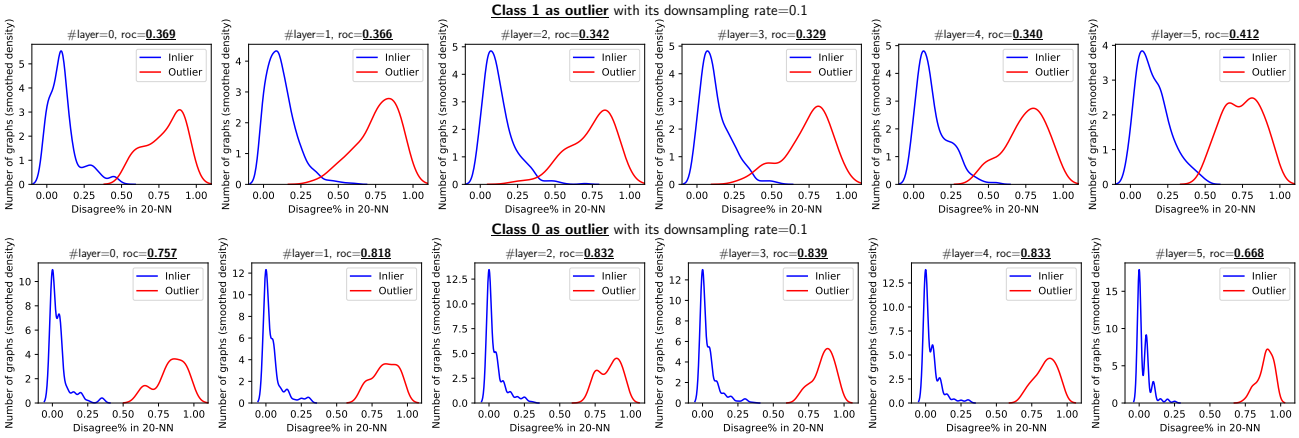


Figure 31: *NN-Disagreement%* distribution of outliers (top row: class 1 is down-sampled and bottom row: class 0 is down-sampled, in red) and inliers (vice versa, in blue) over iterations (left to right) on PROTEINS with **OCGIN**.

A.8.3. NCI1

We analyze the pairwise similarity of graphs on two variants of down-sampled NCI1 dataset, by showing the distribution of *NN-Disagreement%* (measures shown in sec.3.3) and corresponding ROC-AUC. The analysis is conducted for three methods: WL+LOF (Figure 32), PK+LOF (Figure 33), and OCGIN (Figure 34).

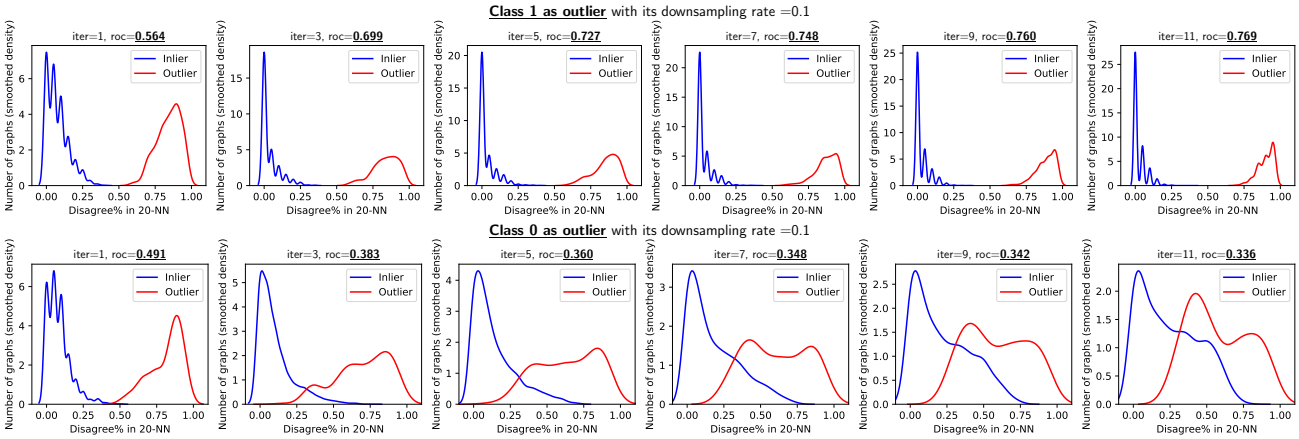


Figure 32: *NN-Disagreement%* distribution of outliers (top row: class 1 is down-sampled and bottom row: class 0 is down-sampled, in red) and inliers (vice versa, in blue) over iterations (left to right) on NCI1 with **WL+LOF**.

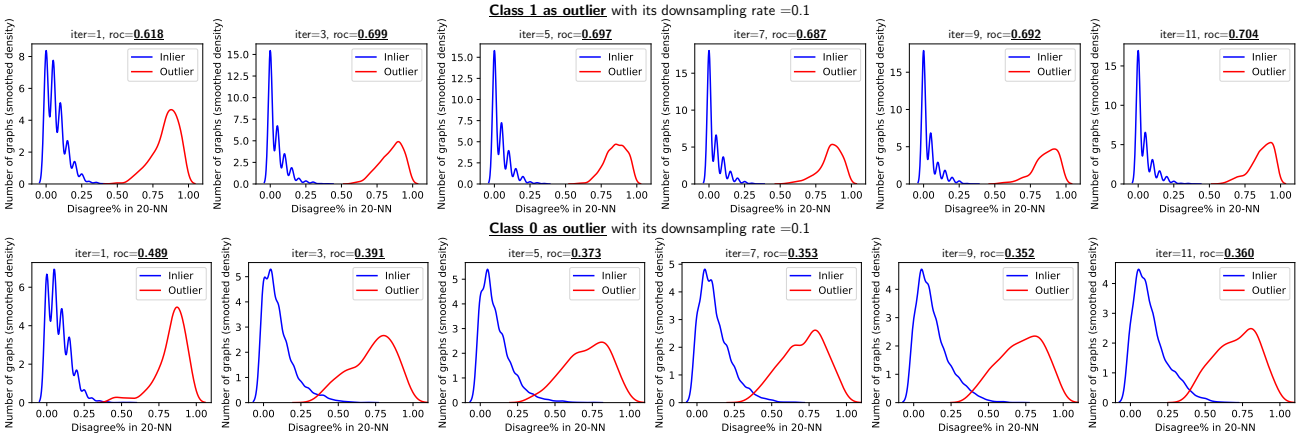


Figure 33: *NN-Disagreement%* distribution of outliers (top row: class 1 is down-sampled and bottom row: class 0 is down-sampled, in red) and inliers (vice versa, in blue) over iterations (left to right) on NCI1 with **PK+LOF**.

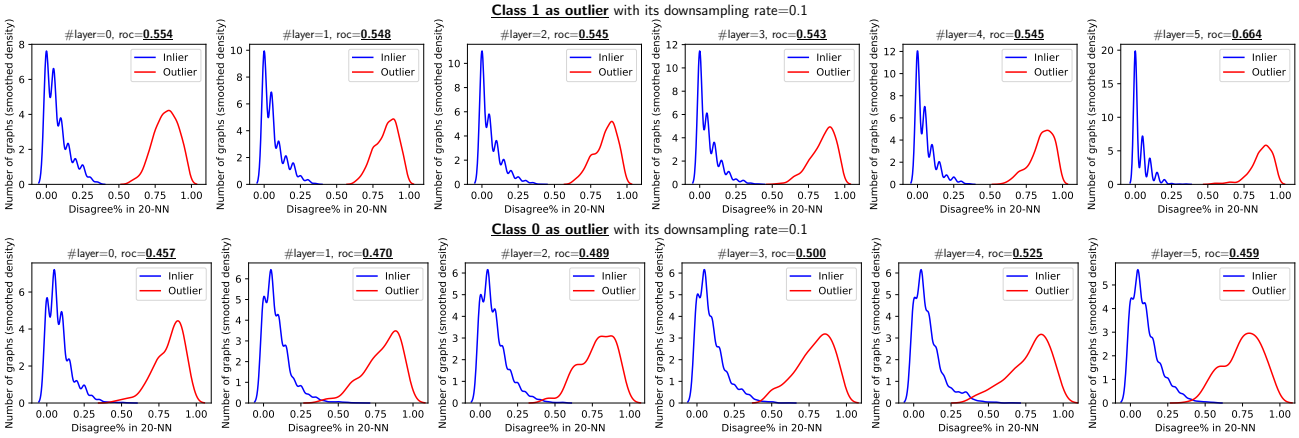


Figure 34: *NN-Disagreement%* distribution of outliers (top row: class 1 is down-sampled and bottom row: class 0 is down-sampled, in red) and inliers (vice versa, in blue) over iterations (left to right) on NCI1 with **OCGIN**.

A.9. Additional Measures on IMDB

Different from DD, PROTEINS, and NCI1, IMDB is the only dataset we haven't observe all peculiar observations (mentioned in sec.3.1), and we have shown analysis for WL+LOF on this dataset in sec.4.2. Now we present similar measures and analysis for PK+LOF (Figure 35 and Figure 36) and OCGIN (Figure 37 and Figure 38).

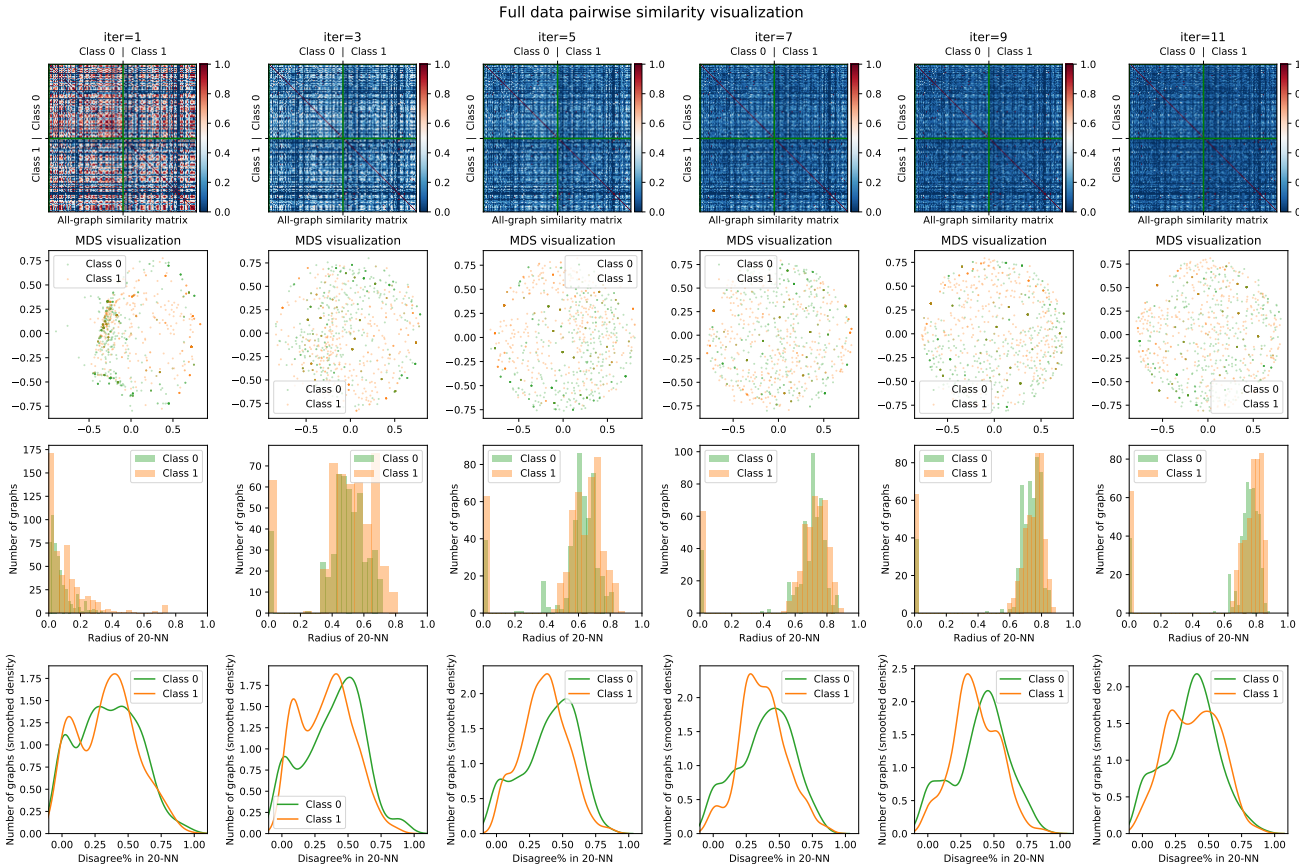


Figure 35: (Top row) Pairwise similarity matrix for all graphs in full IMDB-BINARY dataset, based on **PK** kernel over increasing iterations (left to right). (Second row) 2-d MDS visualization based on the similarity matrix. (Third row) Distribution of *NN-Radius* for all graphs in each class. (Last row) Distribution of *NN-Disagreement%* for all graphs in each class.

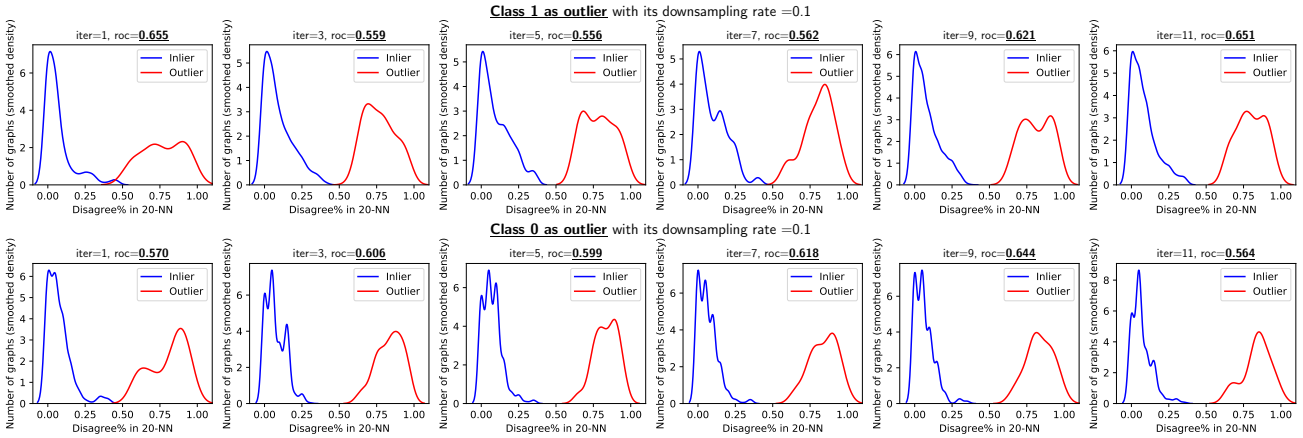


Figure 36: *NN-Disagreement%* distribution of outliers (top row: class 1 is down-sampled and bottom row: class 0 is down-sampled, in red) and inliers (vice versa, in blue) over iterations (left to right) on IMDB-BINARY with **PK+LOF**.

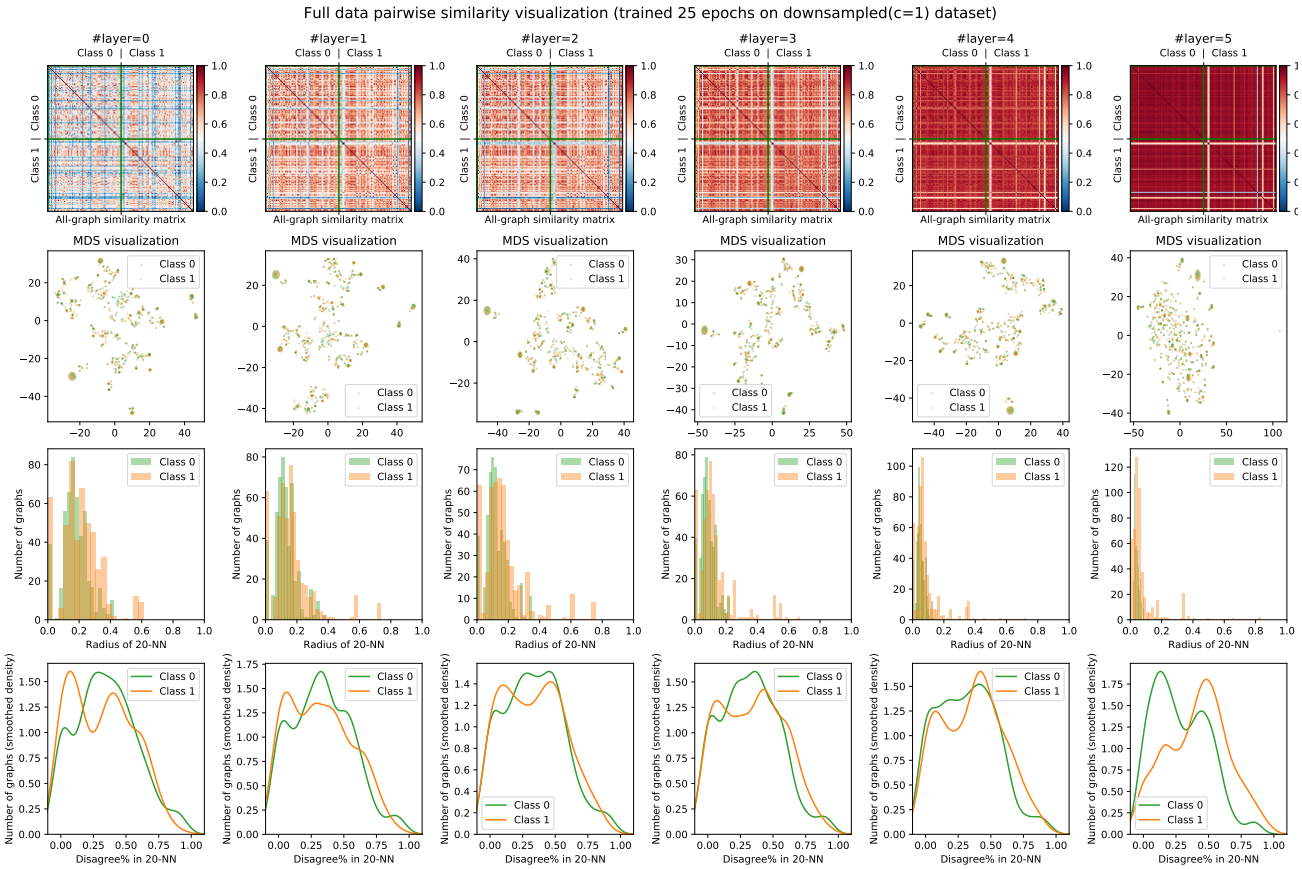


Figure 37: (Top row) Pairwise similarity matrix for all graphs in full IMDB-BINARY dataset, based on **OCGIN** over increasing iterations (left to right). (Second row) 2-d MDS visualization based on the similarity matrix. (Third row) Distribution of *NN-Radius* for all graphs in each class. (Last row) Distribution of *NN-Disagreement%* for all graphs in each class.

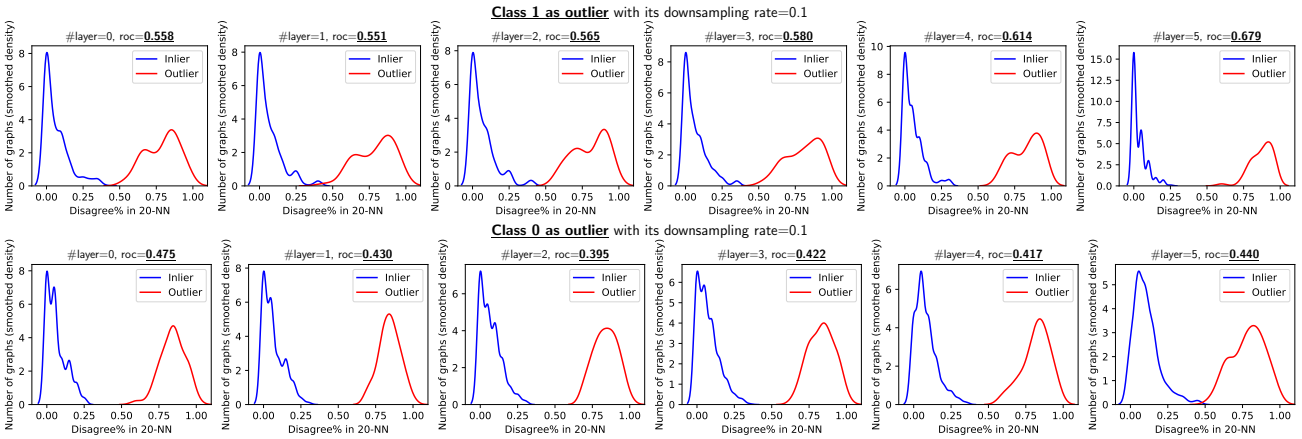


Figure 38: *NN-Disagreement%* distribution of outliers (top row: class 1 is down-sampled and bottom row: class 0 is down-sampled, in red) and inliers (vice versa, in blue) over iterations (left to right) on IMDB-BINARY with **OCGIN**.

A.10. Discussion of Impact on Outlier Detection

Most traditional outlier detectors are designed for low-dimensional point-cloud (i.e. vector) datasets based on specific assumptions of the behavior in input space. With the emergence of large-scale high-dimensional and highly structured data such as images, text, time series, and so on (e.g. in medical records), deep anomaly detection has emerged as a promising area of research (Chalapathy & Chawla, 2019; Ruff et al., 2020; Pang et al., 2020). Irrespectively, most detection models assume outliers to 'live' in low-density regions of the space. In contrast, as our findings reveal, repurposing classification datasets for outlier model evaluation often creates a setting in which this assumption is violated. In fact the setting in which the so-called outlier samples form a higher density region than the inliers is studied under a different area called rare category discovery (He, 2012).

Moreover, our findings have a close connection with graph representations as generated by propagation based models. This motivates the design of better graph representation learning techniques for outlier detection, particularly those that do not exhibit similar pitfalls.

Our study focused on binary graph classification datasets, however the issues would arguably extend to multi-class scenarios where two or more classes are subsampled to construct the outliers. Although the possibility of all down-sampled classes to violate the sparsity assumption would be reduced, the model errors could still be biased (i.e. skewed) toward the ones that exhibit the issues we have reported. Finally, more research is called for the transferability of our findings to non-graph settings.