GraphBoost: Adaptive Boosting Node Generation for Class-Imbalanced Graphs

Yuhe Gao*
ygao32@ncsu.edu
North Carolina State University
Raleigh, NC, USA

Sheng Zhang*
shengz0918@gmail.com
North Carolina State University
Raleigh, NC, USA

Rui Song†
songray@gmail.com
North Carolina State University
Raleigh, NC, USA

ABSTRACT

Classification in imbalanced data, where the majority class has a much larger representation than the minority class, has been a significant topic in recent decades. Two popular approaches for handling this issue are (1) rebalancing the sizes of classes through reweighting, resampling, or synthetic nodes generating, and (2) focusing on the data points that are hard to classify to enhance the classifier performance. In graphical data, several methods, such as GraphSMOTE [35], and GraphENS [23], from the first type have been developed recently for class-imbalanced node classification tasks, but few adaptations of the second approach have been proposed. In response to this gap , we present a novel multi-stage boosting framework inspired by the second approach. In particular, the framework proposed in this research paper jointly generates the topological structure and features of synthetic nodes by minimizing the distance of synthetic nodes and misclassified nodes from previous training stages. Our experiments on class-imbalanced graphs show that our novel framework outperforms standard graph neural networks. Furthermore, our framework can be combined with existing methods (such as GraphENS) resulting in further performance enhancements.

CCS CONCEPTS

• **Computing methodologies** → **Supervised learning by classification**; *Neural networks*.

KEYWORDS

Imbalanced Graph, Graph Neural Network, Generative Model, Node Classification

ACM Reference Format:
Yuhe Gao, Sheng Zhang, and Rui Song. 2023. GraphBoost: Adaptive Boosting Node Generation for Class-Imbalanced Graphs. In *Long Beach '23: ACM Symposium on Neural Gaze Detection, August 06–10, 2023, Long Beach, CA.* ACM, New York, NY, USA, 8 pages. https://doi.org/XXXXXXX.XXXXXXX

1 INTRODUCTION

In the traditional classification tasks with imbalanced data, two major branches of popular methods are applied to address the issues caused by imbalance. The first branch is the methods that rebalance the training set by reweighting different classes in the loss function, generating synthetic data from minority classes, or applying other sampling schemes. For instance, oversampling methods create more data from minority classes by sampling and interpolation [3]. Undersampling methods, on the other hand, downsize the majority class training data through some selection procedure [16]. Another branch of methods utilizes the subset of training data that is more easily to be misclassified and assigns more weights to such data points, such as focal loss [17], Adaboost [8] and ADASYN [12]. For instance, focal loss differs from classic cross-entropy loss by adding a shrinking factor to the logarithm terms to down-weight the well-classified data points, which significantly enhances the accuracy of object detection tasks.

Node classification is a classic task in large-scale graph analysis with wide applications. The objective of node classification is to label the nodes of a graph into different classes. To solve such tasks, various graph neural networks (GNNs) have been developed using the idea of convolutional neural networks (CNN) to implicitly propagate the information of the labeled nodes to unlabeled nodes through the linkage between nodes [9, 15, 32, 34]. These convolution-based graph neural networks have achieved superior performance on multiple benchmark datasets [33].

Despite the high performance of GNNs in standard node classification tasks, classical GNNs such as GCN[15], GAT [32], or GraphSage [9] cannot be well generalized into the situation of classification in class-imbalanced graphs. In real-world networks, imbalanced node classes have been observed in many cases, such as chemical compound graphs [22] and fake account detection graphs [19]. When applied to such datasets, GNN may suffer from bias towards the majority class in the training dataset. The complex topological structure of graphs adds further complication to such situations. The techniques of handling imbalanced data in traditional classifications have been introduced to graphs, where GraphSMOTE by [35] and GraphENS by [23] are variants of rebalancing methods that are tailored to class-imbalanced node classification problems. In particular, GraphENS is the state-of-the-art (SOTA) method handling node classification in imbalanced graphs so far. On the other hand, Boost-GNN [29] adapts the method of Adaboost (which belongs to the second branch of methods) to GNN by reweighting the misclassified nodes. However, Boost-GNN cannot be applied simultaneously with other rebalance-based methods such as GraphENS or oversampling.

In this work, we propose a multi-stage boosting framework in the graph (GraphBoost) by generating synthetic nodes similar to the misclassified nodes in the training set from the previous stage. Our method can be directly applied to vanilla GNN models or combined

*Both authors contributed equally to this research.
†Corresponding author.

Conference acronym 'XX, August 06–10, 2023, Long Beach, CA
© 2023 Association for Computing Machinery.
This is the author's version of the work. It is posted here for your personal use. Not for redistribution. The definitive Version of Record was published in *Long Beach '23: ACM Symposium on Neural Gaze Detection, August 06–10, 2023, Long Beach, CA*, https://doi.org/XXXXXXX.XXXXXXX.

with existing synthetic node generation methods on imbalanced data such as GraphENS. To our best knowledge, our work is the first to leverage the misclassified nodes and synthetic node generation in imbalanced node classification tasks.

There are two challenges in constructing a general framework: first, the attributed graph data are non-Euclidean whose distribution contains information on graph topology structure as well as the attributes of nodes. Hence, it is non-trivial to construct the generator to model this distribution. Second, even if the generator can model the distribution of feature and topological structure in a graph, the generator should be further trained properly to boost the performance of the classifier. A poor-quality generator would introduce noises to the existing graph and adversely affect the classifier performance. To address the first challenge, the topological structure is captured by graph embedding methods that can restore the adjacency matrix of the original graph. The graph embedding vectors of each node are then concatenated with node attributes, where a generator will generate the joint distribution of them. Details of graph embedding methods and generating adjacency information of synthetic nodes can be found in Section 3.3 and Section 4.1 respectively. To handle the second challenge, we adapt the semi-GAN framework [26], where we convert the M-class classification task into solving a $(M+1)$-class problem where the synthetic $(M+1)$-class is generated by the generator. To address the training convergence issue in generative adversarial network (GAN), we use the variational auto-encoder (VAE) by [14] as the generator. [6] provides a theoretical insight that the generated data in semi-GAN are able to boost the performance of the classifier in a general classification setting if the generated data fall on the complementary part of the existing classes. In our framework, the generated data mimic the distribution of nodes that are difficult to be classified, which can be regarded as complementary parts. An illustration of our generated nodes is in Figure 1.

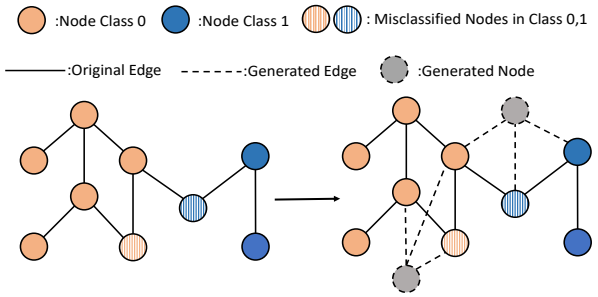

Figure 1: Illustration of the proposed framework GraphBoost. Two classes in the graph, orange nodes denote the majority class 0 and blue nodes represent the minority class 1. In the previous stage, two nodes are misclassified. In the following stage, synthetic nodes are generated by mimicing the features and topological structure of misclassified nodes and labeled as class 2. GNN classifier is updated by classifying all classes.

Our contribution can then be summarized into three folds. First, we propose a novel framework (GraphBoost) motivated by Adaboost and focal loss to generate synthetic nodes similar to nodes that are easy to be misclassified to improve baseline classifier accuracy on imbalanced node classifications. Second, we propose to use an adversarial learning framework to generate synthetic nodes. In particular, the generative model in our framework generates nodes labeled as $M+1$-th class in the data, where the feature part and link part of these synthetic nodes are generated jointly. This further extends the application of generative models to synthetic nodes generation in graphs. Third, in the setting of class imbalanced node classification, we show that using our boosting framework can obtain improvement over the performance of vanilla GNN classifiers. Furthermore, our framework can be applied simultaneously with existing SOTA method (such as GraphENS) to achieve further performance enhancement.

2 RELATED WORK

2.1 Methods for Classification on Imbalanced Data

Many methods have been proposed to handle classification tasks in class-imbalanced setting. One popular branch of methods is the approaches that rebalance the training datasets. Some of the methods directly rebalance the training data through sampling or synthetic data generation, while some other methods [5, 13, 31] achieve the same goal by reweighting the loss function. Resampling methods include oversampling, undersampling and combination of both schemes [16, 18, 30]. Synthetic data generation methods include SMOTE [3],SMOTEBoost [4], Borderline-SMOTE[11], etc. Another branch of methods focus on using data points that are difficult to be classified to enhance classification performance in imbalanced data. For instance, Adaboost [8] and its variants [7, see, e.g.] increase the weight of misclassified data points in a multi-stage procedure and obtain an voted ensemble of classifiers from all the stages. Focal loss [17], on the hand, alters the classical cross-entropy loss by reducing the weight of easy-to-classify data points and assigning more weight to the difficult, misclassified data, which is more likely to prevent over-fitting compared to cross-entropy loss [20]. ADASYN [12] differs from aforementioned methods by directly generating synthetic data of data points that are from minority classes and difficult to learn, where the number of nearest neighbors belonging to the majority class serves as a measure for classification difficulty. All of these methods are originally proposed on datasets not directly related to graphs.

2.2 Node classification Methods in Class-imbalanced Graphs

In class-imbalanced node classification tasks, classical GNNs such as GCN[15], GAT [32] or GraphSage [9] suffer from the bias towards the majority class, and may perform poorly on nodes from minority groups. Thus, methods from classification on imbalanced data are adapted to graphs. Similar to the synthetic data generation methods, GraphSMOTE[35] creates nodes from minority classes based on the nearest neighbors from the embedding space of GNN, where connections of generated nodes are created with an edge predictor. ImGAGN [25] uses GAN to generate nodes from minority classes to rebalance the data. In GraphENS [23], the neighbor memorization issue of applying traditional oversampling methods in graphs is identified and the synthetic nodes are created to reduce overfitting towards nodes in minority classes. The aforementioned

methods adapts rebalancing methods to node classification setting and GraphENS is the SOTA method among this class. On the other hand, Boost-GNN [29] introduces the methods that use misclassified nodes to class-imbalanced graphs. In a multi-stage procedure similar to Ada-boost, the misclassified nodes are assigned with more weights in loss and the ensembled vote of GNNs from all stages are provided. However, this method can not be directly combined with methods above. On the contrary, our model (GraphBoost) can not only be applied to vanilla GNNs but also be used together with synthetic node generating methods such as GraphENS.

3 PRELIMINARY

We first introduce the notation of graphs. Let $\mathcal{G} = (V, E)$ denote a graph, where V is the set of nodes with $|V| = n$ and $E \subset V \times V$ is a set of edges with $|E| = m$. The adjacency matrix $\mathbf{A} \in \mathbb{R}^{|V|\times|V|}$ is defined as $A_{ij} = 1$ if node v_i and v_j is connected, otherwise $A_{ij} = 0$. Suppose each node v_i has a d-dimensional feature $\mathbf{x}_i \in \mathbb{R}^d$ and a single value label $y_i \in \{1, 2, .., M\}$. The matrix $\mathbf{X} = [\mathbf{x}_i]_{v_i \in V} \in \mathbb{R}^{n\times d}$ contains the features of all nodes. Denote $V^{\mathcal{L}}$ as the set of training data where the labels of nodes are known. The size of each class in $V^{\mathcal{L}}$ can be highly imbalanced. The objective of node classification is to find out the labels $\{y_j\}$ of test set $V^{\mathcal{U}}$ given adjacency matrix $\mathbf{A}$, feature matrix and labels of the training set.

3.1 Convolution-based Graph Neural Network Classifier

Based on the Laplacian smoothing, the convolution-based GNN models propagate the information of node features across the nodes' neighbors in each layer. Specifically, in GCN, the layer-wise propagation rule can be defined as follows:

$$\mathbf{H}^{(l+1)} = \sigma(\mathbf{D}^{-1}\mathbf{A}\mathbf{H}^{(l)}\mathbf{W}^{(l)} + \mathbf{b}^{(l)}), \quad l = 0, 1, 2.., L-1 \tag{1}$$

where $\mathbf{W}^{(l)}$ and $\mathbf{b}^{(l)}$ are layer-specific trainable weight matrix and bias, respectively. $\sigma(\cdot)$ is an activation function. $\mathbf{D}$ is the diagonal degree matrix with $D_{ii} = \sum_j A_{ij}$. Hence, $\mathbf{D}^{-1}\mathbf{A}$ represents normalization of adjacency matrix $\mathbf{A}$. The initial layer $\mathbf{H}^{(0)}$ is the feature matrix $\mathbf{X}$. The final layer $\mathbf{H}^{(L)}$ followed by a $softmax$ layer can be viewed as the prediction of one-hot representation for the true label $\mathbf{y}$, which is the output of the neural network.

3.2 Framework of Semi-GAN

In semi-GAN, the classifier C and generator G play a non-cooperative game, where the classifier aims to classify the unlabeled data as well as distinguish the generated data from the real data, while the generator attempts to match the features of generated data with the features of real data. Therefore, the objective function for the classifier can be divided into two parts [26]. The first part is the supervised loss function

$$\mathcal{L}_{sup} = -\mathbb{E}_{v,y\sim p_{V^{\mathcal{L}}}} \log P_C(y|v, y \le M), \tag{2}$$

which is the log probability of the node label predictions on the real nodes in training data. The second part is the loss function for differentiating the real nodes and generated nodes $\mathcal{L}_{gen} = -\mathbb{E}_{v\sim p_{V^{\mathcal{L}}}} \log[1 - P_C(y = M+1|v)] - \mathbb{E}_{v\sim p_{V^G}} \log P_C(y = M+1|v)$, which is the log probability of the (M + 1)-th class for real nodes $V^{\mathcal{L}}$ and generated nodes V^G. The classifier C can be trained by minimizing the total loss function

$$\mathcal{L}_C = \mathcal{L}_{sup} + \mathcal{L}_{gen}. \tag{3}$$

For the objective function of generator, [26] found minimizing feature matching loss in Equation 4 achieved superior performance in practice

$$\mathcal{L}_G = ||\mathbb{E}_{v\sim p_{V^{\mathcal{L}}}}(\mathbf{f}(v)) - \mathbb{E}_{\mathbf{z}\sim p_{\mathbf{z}}(\mathbf{z})}(\mathbf{f}(G(\mathbf{z})))||_2^2, \tag{4}$$

where the feature matching function $\mathbf{f}(\cdot)$ maps the input into a feature space, and $\mathbf{z} \sim p_{\mathbf{z}}(\mathbf{z})$ is noise drawn from some given distributions. [6] provided a theoretical justification that complementary generator G was able to boost the performance of classifier C.

3.3 Graph Embedding Methods

Graph embedding matches complex graph information into low dimensional vector space, which is applied in many machine learning tasks with graphs. There are three types of graph embedding methods: random walk-based methods, matrix factorization-based methods, and deep learning model-based methods[10, 21]. The graph embedding methods represent node v_i with vector $u_i \in \mathbb{R}^{d_u}$ and the matrix $\mathbf{U} \in \mathbb{R}^{n\times d_u}$ contains the embedded information for the whole graph. In this paper, we use the embedding information $\mathbf{U}$ of the original graph to generate the adjacency information of synthetic nodes. Therefore, the factorization-based methods such as GraRep [2], Graph Factorization [1] , and Adjacency/Laplacian Spectral Embedding [24] are considered in this paper, as adjacency information can be recovered using the node embedding obtained by these methods. Among these methods, Laplacian Spectral Embedding (LSE) is selected, as it is efficient in terms of computing time and enjoys highest accuracy when recovering the original adjacency matrix $\mathbf{A}$ in our setting. LSE is based on the eigen decomposition of the normalized Laplacian adjacency matrix $\mathcal{L}(\mathbf{A}) = \mathbf{D}^{-1/2}\mathbf{A}\mathbf{D}^{-1/2}$ [24]. $\mathcal{L}(\mathbf{A})$ can be decomposed as $\mathcal{L}(\mathbf{A}) = \mathbf{W}\Sigma\mathbf{W}^T$, where $\mathbf{W}$ is the matrix of eigenvectors and Σ contains the eigenvalues. The embedding matrix $\mathbf{U} = \mathbf{W}_{d_u}\Sigma_{d_u}^{\frac{1}{2}}$ can then be obtained where the embedding dimension d_u corresponds to the number of top eigenvalues selected. Tuning of d_u is presented in Section B of Appendix.

4 FRAMEWORK OF GRAPHBOOST

To improve the performance of GNN on class-imbalanced node classification tasks, we propose the following multi-stage boost node generation framework that utilizes misclassified nodes during training. In this framework, we first discuss the generator of synthetic nodes and the construction of classifiers. Then we formally present the main algorithm. Furthermore, we discuss how this boosting node-generation framework can be incorporated with existing synthetic node-generating methods (e.g. GraphENS [23]) in imbalanced data.

4.1 Construction of the Generator

The generator G generates a batch of B fake nodes $V^G = \{v_j\}_{j=1}^B$ by creating the feature matrix $\mathbf{X}_\mathbf{G} \in \mathbb{R}^{B\times d}$ and the augmented adjacency matrix $\tilde{\mathbf{A}} = \begin{bmatrix} \mathbf{A}_{00} & \mathbf{A}_{10}^T \\ \mathbf{A}_{10} & \mathbf{A}_{11} \end{bmatrix} \in \mathbb{R}^{(n+B)\times(n+B)}$. Here x_j is the feature vector corresponding to the j-th fake node. The submatrix $\mathbf{A}_{00}$ is the original adjacency matrix of the existing nodes V. $\mathbf{A}_{10}$ represents

the adjacency relations between the real nodes and fake nodes, with $\tilde{\mathbf{A}}_{i,j} = 1$ indicating that the fake node v_j is connected with the real node v_i and $\tilde{\mathbf{A}}_{i,j} = 0$ otherwise. $\mathbf{A}_{11}$ is the adjacency matrix within the B generated fake nodes.

For a single fake node v_j among $\{v_j\}_{j=1}^{B}$, its distribution $p_G(v_j)$ can be represented as the joint distribution of the corresponding feature, embedding vector, and adjacency relation $p_G(\mathbf{x}_j, \mathbf{u}_j, \mathbf{a}_j)$, where $\mathbf{a}_j = \tilde{\mathbf{A}}_{j,:}$. The feature $\mathbf{x}_j$ will be generated by the distribution $p_{G_1}(\mathbf{x}_j)$. To generate $\mathbf{a}_j$, we will first apply the LSE embedding in Section 3.3, and embed the adjacency matrix of original graph $\mathbf{A}_{00}$ by $\mathbf{U} \in \mathbb{R}^{n \times d_u}$. Then the embedded vector $\mathbf{u}_j$ for the fake node j can be generated by $p_{G_2}(\mathbf{u}_j)$. Note that for the terms $a_{i,j}$ in $\mathbf{A}_{10}$ and $\mathbf{A}_{11}$, $a_{i,j} \sim Bernoulli(p = \mathbf{u}_i^T \mathbf{u}_j)$. That is, the links within the fake nodes, together with the links between the existing nodes and fake nodes are constructed based on the cross-product of the embedding vectors. The distribution for generated node $p_G(v_0)$ can be denoted by $p_G(v_0) = p_G(\mathbf{x}_0, \mathbf{u}_0, \mathbf{a}_0) = p_G(\mathbf{x}_0, \mathbf{u}_0) p(\mathbf{a}_0 | \mathbf{u}_0)$. Hence, the combined adjacency matrix can be represented in the following expression

$$\tilde{\mathbf{A}} = \begin{bmatrix} \mathbf{A}_{00} & \mathbf{A}_{10}^T \\ \mathbf{A}_{10} & \mathbf{A}_{11} \end{bmatrix} \in \mathbb{R}^{(n+B)\times(n+B)}, \tag{5}$$

and the combined feature vector is

$$\tilde{\mathbf{X}} = \begin{bmatrix} \mathbf{X} \\ \mathbf{X}_G \end{bmatrix} \in \mathbb{R}^{(n+B)\times d}.$$

The diagonal degree matrix $\tilde{\mathbf{D}} \in \mathbb{R}^{(n+B)\times(n+B)}$ can be denoted as $\begin{bmatrix} \mathbf{D}_* & \mathbf{0} \\ \mathbf{0} & \mathbf{D}_B \end{bmatrix}$, where $\mathbf{D}_* \in \mathbb{R}^{n \times n}$ with $D_{*,ii} = \sum_j A_{00,ji} + \sum_b A_{10,bi}$ and $\mathbf{D}_B \in \mathbb{R}^{B \times B}$ with $D_{B,ii} = \sum_b A_{11,ib} + \sum_j A_{10,ij}$.

For the generated nodes, the distribution of their feature component $\mathbf{X}$ and topological embedding $\mathbf{U}$, p_G, is obtained as a result of training the generator component, such that their distribution will be close to the feature and embedding of some target set of nodes. The concatenated feature and embedding of the target nodes are then denoted as $\mathcal{E}_{target} = \begin{bmatrix} \mathbf{X}_{target} & \mathbf{U}_{target} \end{bmatrix}$ with distribution $p_{\mathcal{E}}$. The objective of the generator is to produce synthetic nodes close to the features and connection relations of target nodes, $\mathcal{E}_{target}$, which is achieved by minimizing the loss of generator, L_G . For L_G, when we use VAE as the generator, it takes the form of

$$L_G(\theta_1, \theta_2) = -E_{z\sim q_{\theta_2}, x\sim p_{\mathcal{E}}}[log p_{\theta_2}(x|z)] + D_{KL}(q_{\theta_1}(z|x)||p_{\theta_2}(z)),$$

where θ_1 represents the parameters of encoder, θ_2 represents the parameters of decoder, z is the vectors generated by decoder and x is the target data. When we use some generators other than VAE, L_G may take different forms such as feature matching loss,

$$\left\| E_{x\sim p_{\mathcal{E}}} f(x) - E_{z\sim p_z} f(G(z)) \right\|,$$

as is in [26]. After training generator with L_G, B synthetic nodes are generated from it.

4.2 Analysis of Classifier in Boosting Stages

For classifier of nodes, we adopt the convolution-based GNN, such as GCN, GAT [32] or SAGE [9] as the classifier. The classifier is applied to the enlarged graph $\tilde{\mathcal{G}} = [\tilde{\mathbf{X}}, \tilde{\mathbf{A}}]$ to obtain the prediction $\tilde{\mathbf{y}}$ of nodes $V \cup V^G$.

Considering the layer-wise propagation of GCN (Equation 1) as the classifier, the propagation rule can be denoted as

$$\begin{aligned}
\tilde{\mathbf{H}}^{(l+1)} &= \sigma(\tilde{\mathbf{D}}^{-1}\tilde{\mathbf{A}}\tilde{\mathbf{H}}^{(l)}\mathbf{W}^{(l)} + \tilde{\mathbf{b}}^{(l)}) \\
&= \sigma(\begin{bmatrix} \mathbf{D}_*^{-1} & \mathbf{0} \\ \mathbf{0} & \mathbf{D}_B^{-1} \end{bmatrix} \begin{bmatrix} \mathbf{A}_{00} & \mathbf{A}_{10}^T \\ \mathbf{A}_{10} & \mathbf{A}_{11} \end{bmatrix} \begin{bmatrix} \mathbf{H}_*^{(l)} \\ \mathbf{H}_0^{(l)} \end{bmatrix} \mathbf{W}^{(l)} + \begin{bmatrix} \mathbf{b}^{(l)} \\ \mathbf{b}_B^{(l)} \end{bmatrix}) \\
&= \sigma(\begin{bmatrix} \mathbf{D}_*^{-1}\mathbf{A}_{00}\mathbf{H}_*^{(l)} + \mathbf{D}_*^{-1}\mathbf{A}_{10}^T\mathbf{H}_0^{(l)} \\ \mathbf{D}_B^{-1}\mathbf{A}_{10}\mathbf{H}_*^{(l)} + \mathbf{D}_B^{-1}\mathbf{A}_{11}\mathbf{H}_0^{(l)} \end{bmatrix} \mathbf{W}^{(l)} + \begin{bmatrix} \mathbf{b}^{(l)} \\ \mathbf{b}_B^{(l)} \end{bmatrix}) \\
&= \sigma(\begin{bmatrix} \mathbf{D}_*^{-1}\mathbf{A}_{00}\mathbf{H}_*^{(l)}\mathbf{W}^{(l)} + \mathbf{b}_*^{(l)} \\ (\mathbf{D}_B^{-1}\mathbf{A}_{10}\mathbf{H}_*^{(l)} + \mathbf{D}_B^{-1}\mathbf{A}_{11}\mathbf{H}_0^{(l)})\mathbf{W}^{(l)} + \mathbf{b}_B^{(l)} \end{bmatrix}) = \begin{bmatrix} \mathbf{H}_*^{(l+1)} \\ \mathbf{H}_0^{(l+1)} \end{bmatrix},
\end{aligned}$$

where the first layer is chosen as the enlarged feature matrix $\tilde{\mathbf{H}}^{(0)} = \tilde{\mathbf{X}}$. Bias vector $\tilde{\mathbf{b}}^{(l)}$ has dimension $(n + B)$ which is denoted as $[\mathbf{b}^{(l)T}, \mathbf{b}_B^{(l)T}]^T$. We denote $\mathbf{b}_*^{(l)} = \mathbf{D}_*^{-1}\mathbf{A}_{10}^T\mathbf{H}_0^{(l)}\mathbf{W}^{(l)} + \mathbf{b}^{(l)}$ to make the format clear. From the formula of $\tilde{\mathbf{H}}^{(l+1)}$, we know the layer propagation of real nodes (first n rows) follows the same format as the GCN layer propagation in Equation 1.

For the last layer $\tilde{\mathbf{H}}^{(L)} \in \mathbb{R}^{(n+B)\times M}$, we adopt the strategy in [26] to obtain the $(M + 1)$ class label $\tilde{\mathbf{y}}$ by

$$\tilde{\mathbf{y}} = soft\max(\tilde{\mathbf{H}}^{(L)} || \mathbf{0}_{(n+B)\times 1}), \tag{6}$$

where $||$ denotes concatenation and $\mathbf{0}_{(n+B)\times 1} \in \mathbb{R}^{(n+B)\times 1}$ is a zero matrix. The loss function for the classifier in our framework follows the same format in Equation 3.

4.3 Improvement Using Misclassified Nodes

Motivated by the boosting methods such as Adaboost, we propose a K-stage training framework for both the classifier and the generator. In stage k, we find out the nodes in training sets that are misclassified by the classifier and set those nodes as target nodes for generators. In stage $k + 1$, the generator will generate fake nodes with features and embeddings close to those misclassified nodes in training set.

In this framework, training of the classifier in the first stage is the same as the training process of vanilla GNN, where parameters of GNN are updated to minimize the loss function in equation 2. Then the misclassified nodes of this GNN in the training set will be identified and set as target nodes for the generator in the next stage. In stage $k \geq 2$, the generator is trained based on the target nodes with L_G. During each epoch of stage k, it will produce B fake nodes. Those synthetic nodes are then labeled as class $M + 1$ and added to the training set. The GNN from the previous stage will then be trained on the updated training data with loss function in equation 3.

During this process, the GNN and its validation accuracy of each stage will be stored. In stage k, the aggregated prediction of GNN in stage 1 to k will be calculated with voting weight equal to the validation accuracy. Thus, we can obtain testing accuracy and validating accuracy for the ensembled model in each stage. After training the classifier in stage K, the prediction of the ensembled model with the highest validation score will be returned as the output.

4.4 Combine with Other Synthetic Node Generator

Our framework can also be combined with other synthetic node generators in class imbalanced graphs, such as GraphENS [23]. Suppose other generators G^{other} (e.g., GraphENS) generate B_0 synthetic nodes. In this case, the adjacency matrix would be $\tilde{\mathbf{A}}_0$ =

$\begin{bmatrix} \mathbf{A}_{00} & \mathbf{A}_{01} \\ \mathbf{A}_{10} & \mathbf{A}_{11} \end{bmatrix} \in \mathbb{R}^{(n+B_0)\times(n+B_0)}$, where $\mathbf{A}_{01}, \mathbf{A}_{11}, \mathbf{A}_{11}$ correspond to synthetic nodes and links generated by other methods. Note that the topological structure in $\mathbf{A}_{01}, \mathbf{A}_{11}, \mathbf{A}_{11}$ can be updated dynamically through the training process (see e.g. in GraphENS). Thus, the embedding of $\tilde{\mathbf{A}}_0$ keeps changing and learning the embedding of whole matrix $\tilde{\mathbf{A}}_0$ may cause converging issue for the generative models. Therefore, for synthetic link generating, we use the the LSE embedding $\mathbf{U}$ of the original graph $\mathbf{A}_{00}$ rather than $\tilde{\mathbf{A}}_0$, and the generated fake nodes by our model will only be linked with the original nodes in $\mathbf{A}_{00}$ based on generated embedding vectors. Our generator G will then be trained in the same way based on loss L_G and generate B fake nodes The augmented adjacency matrix will then be

$$\tilde{\mathbf{A}} = \begin{bmatrix} \mathbf{A}_{00} & \mathbf{A}_{01} & \mathbf{A}_{20}^T \\ \mathbf{A}_{10} & \mathbf{A}_{11} & 0 \\ \mathbf{A}_{20} & 0 & \mathbf{A}_{22} \end{bmatrix} \in \mathbb{R}^{(n+B_0+B)\times(n+B_0+B)}. \tag{7}$$

The generated nodes by other generators and ours are supplementary to each other, each providing certain performance boost on the vanilla GNN. This algorithm is formally presented in Algorithm 1.

5 EXPERIMENTS

5.1 Datasets

The datasets in our experiment are based on the three standard citation network benchmark datasets - Cora, Citeseer, and Pubmed [27]. We closely follow the settings in Park et al. [23] to transfer the original training part of the datasets into imbalanced version of Cora-LT, Citeseer-LT, and Pubmed-LT, where the ratio of the majority class in the training sets to the minority class is 100. These imbalanced citation network datasets can mirror real-world academic scenarios, where some popular or well-established research fields gather a substantial amount of publications, while some emerging or specialized research domains attract a much smaller volume of papers. Therefore, node classification on these imbalanced citation network datasets is not only crucial and but also highly relevant to actual situations in academia. Figure 3 in Appendix A illustrates the distribution of class sizes in the three datasets.

5.2 Experiment Setup and Result

Our goal in this experiment is to demonstrate that using our boosting framework to create synthetic nodes can improve the performance of convolution-based GNNs on the imbalanced datasets in Section 5.1. Furthermore, when using our method in conjunction with GraphENS (SOTA), further improvement on accuracy can be achieved.

For the classifier component, three widely used convolution-based GNNs, GCN, GAT, and SAGE are selected in our framework. Apart from the three vanilla GNN models, GraphENS is also added as the baseline on top of those GNNs. For GraphENS, the scale of upsampling is set to be 1, as is recommended by Park et al. [23]. That is, the minority class will be upsampled to reach the size of the majority class in training data. When the baseline model is a vanilla GNN, the generated boosting nodes are directly added to the original graphs. When the baseline model is GNN combined with GraphENS, the boosting nodes will be added together with synthetic nodes provided by GraphENS, as is stated in Section 4. Results of other baseline methods designed for imbalanced graphs such as GraphSMOTE [35] and DR-GCN[28] on these three imbalanced datasets are also added from Park et al. [23].

For the generator in Algorithm 1, we use VAE by Kingma and Welling [14]. For both encoder and decoder, we use fully connected Rectified Linear Unit (ReLU) network with 1 hidden layer of width 128. The dimension of latent vector sampled from decoder is 64. In each stage, the VAE is trained with 2500 epochs to learn the feature/embedding of misclassified nodes in the training set.

To ensure fair comparison, the total number of training epochs for GNN in both baseline methods and our model are set to be 2000. In particular, when applying our K-stage training framework, the number of training epochs in the first stage is set to be $T_{c,1} = 2000 - \sum_{k=2}^{K} T_{c,k}$ and the training epoch numbers in the following $K-1$ stages for boosting are $T_{c,k}$. For the hyper-parameters in our framework, such as the number of synthetic nodes B, number of boost stages K, and epoch number $T_{c,k}$ of boost stages $k \geq 2$, they can be tuned based on the validation accuracy. In particular, B is selected among $50, 100, 150, \cdots, 800$ (with ablation study in Section 5.3), K is selected among $2, 3, 4, 5$ and $T_{c,k}$ is selected from $25, 50, 75$.

For the implementation of vanilla GNNs and GraphENS on Cora-LT, Citeseer-LT, and Pubmed-LT, we use the code provided by Park et al. [23]. Consistent with their study, we choose the layer number of the GNN from the set $\{1, 2, 3\}$ and choose the layer width of GNN from $\{64, 128, 256\}$ during our experiments. For different combination of GNN types and datasets, the specific layer number and width of GNN are selected from the sets above such that the performance of vanilla GNN or GraphENS reported in Park et al. [23] can be achieved. The outcomes of different method combinations are displayed in Table 1. This table includes three performance metrics: the accuracy score, the balanced accuracy score (the average of true positive rate of each class), and the average F1 score (the average of F1 score of each class). From Table 1, it is shown that our proposed method outperforms vanilla GNN in terms of accuracy score (Acc) on most dataset-GNN combinations. Furthermore, when our model is combined with GraphENS, the accuracy (Acc) of GraphENS can be further enhanced.

5.3 Ablation Study on Synthetic Nodes

In this section, we further study how the generated nodes affect the performance of our framework. We first conduct an experiment where the VAE generator is replaced by a white noise placeholder, such that features of synthetic nodes are white noise vectors from uniform distribution and the edges of generated nodes are randomly generated based on the edge density in original graph. The accuracy of this modified model combined with GraphENS and GCN on Cora/CiteSeer/PubMed are 77.66(0.10)%, 66.55(0.17)% and 78.28(0.27)%, respectively, which are worse than the results obtained by our proposed model (77.98 (0.08)%, 66.84 (0.18)% and 79.78 (0.24)%) in Table 1 . These results demonstrate the importance of the generating nodes close to the misclassified nodes.

In another study, we vary the batch size B in $\{100, 200, \cdots, 700, 800\}$ on Cora and PubMed for GraphBoost- GraphENS with SAGE. The accuracy of our proposed framework with respect to a different choice of B is reported in Figure 2. It shows the importance of batch size selection and that generating moderate-size of synthetic nodes can achieve the best performance in terms of accuracy.

Table 1: Summary of results in terms of classification accuracy score (Acc), balanced accuracy score (Bacc) and average F1 score (F1) of Cora-LT, CiteSeer-LT and Pubmed-LT. The baseline results of Vanilla GNN, DR-GCN, GraphSMOTE, and GraphENS are from Table 1 in Park et al. [23]. "GraphBoost-" represents the baseline method combined with our node generating framework. In each combination of GNN and dataset, the best result is highlighted as bold. All the GraphBoost- results are obtained by averaging over 5 repetitions, with standard error presented in the following parentheses.

	Dataset	Cora-LT			CiteSeer-LT			PubMed-LT		
	Method	Acc	Bacc	F1	Acc	Bacc	F1	Acc	Bacc	F1
GCN	Vanilla	73.66 (0.28)	62.72 (0.39)	63.70 (0.43)	53.90 (0.70)	47.32 (0.61)	43.00 (0.70)	70.76 (0.74)	57.56 (0.59)	51.88 (0.53)
	GraphBoost-Vanilla	73.98 (0.33)	63.83 (0.77)	64.30 (0.58)	56.62 (0.70)	49.85 (0.68)	45.61 (0.87)	71.02 (0.34)	57.77 (0.27)	52.16 (0.21)
	DR-GCN	73.90 (0.29)	64.30 (0.39)	63.10 (0.57)	56.18 (1.10)	49.57 (1.08)	44.98 (1.29)	72.38 (0.19)	58.86 (0.15)	53.05 (0.13)
	GraphSMOTE	76.76 (0.31)	69.31 (0.37)	70.21 (0.64)	62.58 (0.30)	55.94 (0.34)	54.09 (0.37)	75.98 (0.22)	70.96 (0.36)	71.85 (0.32)
	GraphENS	77.76 (0.09)	**72.94** (0.15)	**73.13** (0.11)	**66.92** (0.21)	**60.19** (0.21)	58.67 (0.25)	78.12 (0.06)	74.13 (0.22)	74.58 (0.13)
	GraphBoost-GraphENS	**77.98** (0.08)	72.60 (0.12)	72.97 (0.11)	66.84 (0.18)	60.16 (0.17)	**58.74** (0.16)	**79.78** (0.24)	**77.19** (0.42)	**77.39** (0.34)
GAT	Vanilla	73.60 (0.26)	62.75 (0.37)	63.53 (0.35)	56.76 (0.39)	50.15 (0.34)	46.59 (0.44)	71.26 (0.77)	58.86 (0.82)	54.91(1.12)
	GraphBoost-Vanilla	74.02 (0.27)	64.90 (0.56)	64.61 (0.68)	56.18 (0.36)	49.71 (0.35)	46.43 (0.46)	70.84 (0.46)	58.28 (0.50)	54.00 (0.78)
	DR-GCN	74.42 (0.55)	64.17 (1.00)	64.20 (0.67)	56.74 (0.74)	50.02 (0.75)	45.82 (1.03)	71.52 (0.25)	59.18 (1.06)	54.88 (2.33)
	GraphSMOTE	76.92 (0.31)	70.03 (0.51)	70.47 (0.55)	64.04 (0.38)	57.33 (0.39)	55.43 (0.49)	77.12 (0.49)	73.59 (1.16)	74.40 (0.72)
	GraphENS	78.10 (0.13)	**73.45** (0.19)	**73.48** (0.19)	66.90 (0.29)	60.20 (0.30)	58.70 (0.26)	78.24 (0.15)	74.27 (0.35)	74.68 (0.30)
	GraphBoost-GraphENS	**78.72** (0.23)	73.03 (0.25)	73.37 (0.32)	**67.30** (0.41)	**60.59** (0.39)	**59.02** (0.40)	**78.46** (0.12)	**74.78** (0.14)	**75.21** (0.12)
SAGE	Vanilla	72.08 (0.53)	61.97 (0.67)	61.97 (0.75)	50.76 (0.46)	44.56 (0.49)	40.43 (0.93)	64.54 (0.35)	53.07 (0.55)	48.80 (1.13)
	GraphBoost-Vanilla	72.40 (0.52)	61.73 (0.69)	61.70 (0.72)	50.98 (0.30)	44.72 (0.24)	40.80 (0.23)	66.04 (0.96)	54.40 (1.05)	50.23 (1.61)
	DR-GCN	73.28 (0.46)	63.32 (0.68)	62.95 (1.12)	50.80 (0.50)	44.51 (0.41)	39.02 (0.65)	64.90 (0.52)	52.84 (0.42)	47.56 (0.43)
	GraphSMOTE	74.34 (0.30)	64.76 (0.49)	65.88 (0.50)	58.98 (0.39)	52.11 (0.38)	50.27 (0.74)	70.02 (0.21)	63.04 (0.67)	63.43 (0.54)
	GraphENS	77.26 (0.13)	70.07 (0.28)	70.25 (0.31)	63.98 (0.38)	57.33 (0.42)	55.23 (0.43)	79.60 (0.19)	74.90 (0.49)	75.83 (0.43)
	GraphBoost-GraphENS	**77.48** (0.33)	**70.30** (0.43)	**70.58** (0.48)	**65.70** (0.22)	**58.85** (0.22)	**56.63** (0.24)	**80.26** (0.15)	**75.43** (0.38)	**76.38** (0.31)

6 CONCLUSION

We propose a novel framework, GraphBoost, to improve the convolution-based classifier in imbalanced node classification tasks through generating synthetic nodes that are close to the misclassified nodes. We use embedding of adjacency matrix to capture the topological structure information of the original graph and generate the topological embedding as well as features of fake nodes jointly. The synthetic nodes will then be added to the original imbalanced training data or the training data rebalanced by other methods such as GraphENS (SOTA). In experiemnt, it is shown that the fake nodes generated by our framework can enhance the original GNN or achieve further accuracy improvement based on GraphENS.

The limitation of our method can be the training time for the generator, where VAE training in our framework generally takes approximately 1-2 minutes to converge. The total training duration per repetition in our experiment takes 5 to 10 minutes. Given the extensive hyperparameter combinations, this results in a significant tuning time. Future work could focus on enhancing our model's implementation efficiency or exploring alternative generator types within our framework.

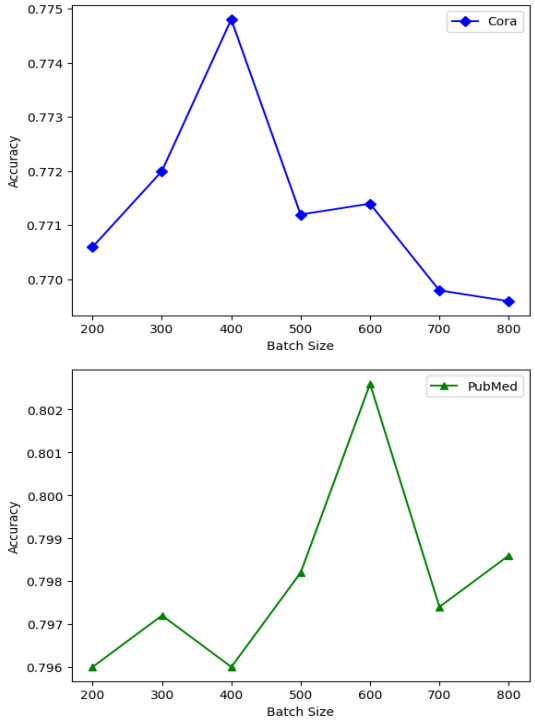

Figure 2: Ablation study for batch size, B, of generated fake nodes. It is shown that moderate size of fake nodes can boost the classifier performance to the largest extent.

REFERENCES

[1] Amr Ahmed, Nino Shervashidze, Shravan Narayanamurthy, Vanja Josifovski, and Alexander J Smola. 2013. Distributed large-scale natural graph factorization. In *Proceedings of the 22nd international conference on World Wide Web*. 37–48.

[2] Shaosheng Cao, Wei Lu, and Qiongkai Xu. 2015. Grarep: Learning graph representations with global structural information. In *Proceedings of the 24th ACM international on conference on information and knowledge management*. 891–900.

[3] Nitesh V Chawla, Kevin W Bowyer, Lawrence O Hall, and W Philip Kegelmeyer. 2002. SMOTE: synthetic minority over-sampling technique. *Journal of artificial intelligence research* 16 (2002), 321–357.

[4] Nitesh V Chawla, Aleksandar Lazarevic, Lawrence O Hall, and Kevin W Bowyer. 2003. SMOTEBoost: Improving prediction of the minority class in boosting. In *European conference on principles of data mining and knowledge discovery*. Springer, 107–119.

[5] Yin Cui, Menglin Jia, Tsung-Yi Lin, Yang Song, and Serge Belongie. 2019. Class-balanced loss based on effective number of samples. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*. 9268–9277.

[6] Zihang Dai, Zhilin Yang, Fan Yang, William W Cohen, and Russ R Salakhutdinov. 2017. Good semi-supervised learning that requires a bad gan. In *Advances in neural information processing systems*. 6510–6520.

[7] Wei Fan, Salvatore J Stolfo, Junxin Zhang, and Philip K Chan. 1999. AdaCost: misclassification cost-sensitive boosting. In *Icml*, Vol. 99. 97–105.

[8] Yoav Freund and Robert E Schapire. 1997. A decision-theoretic generalization of on-line learning and an application to boosting. *Journal of computer and system sciences* 55, 1 (1997), 119–139.

[9] Will Hamilton, Zhitao Ying, and Jure Leskovec. 2017. Inductive representation learning on large graphs. In *Advances in neural information processing systems*. 1024–1034.

[10] William L Hamilton, Rex Ying, and Jure Leskovec. 2017. Representation learning on graphs: Methods and applications. *arXiv preprint arXiv:1709.05584* (2017).

[11] Hui Han, Wen-Yuan Wang, and Bing-Huan Mao. 2005. Borderline-SMOTE: a new over-sampling method in imbalanced data sets learning. In *International conference on intelligent computing*. Springer, 878–887.

[12] Haibo He, Yang Bai, Edwardo A Garcia, and Shutao Li. 2008. ADASYN: Adaptive synthetic sampling approach for imbalanced learning. In *2008 IEEE international joint conference on neural networks (IEEE world congress on computational intelligence)*. IEEE, 1322–1328.

[13] Nathalie Japkowicz and Shaju Stephen. 2002. The class imbalance problem: A systematic study. *Intelligent data analysis* 6, 5 (2002), 429–449.

[14] Diederik P Kingma and Max Welling. 2013. Auto-encoding variational bayes. *arXiv preprint arXiv:1312.6114* (2013).

[15] Thomas N Kipf and Max Welling. 2016. Semi-supervised classification with graph convolutional networks. *arXiv preprint arXiv:1609.02907* (2016).

[16] Miroslav Kubat, Stan Matwin, et al. 1997. Addressing the curse of imbalanced training sets: one-sided selection. In *Icml*, Vol. 97. Citeseer, 179.

[17] Tsung-Yi Lin, Priya Goyal, Ross Girshick, Kaiming He, and Piotr Dollár. 2017. Focal loss for dense object detection. In *Proceedings of the IEEE international conference on computer vision*. 2980–2988.

[18] Charles X Ling and Chenghui Li. 1998. Data mining for direct marketing: Problems and solutions.. In *Kdd*, Vol. 98. 73–79.

[19] Mohammadreza Mohammadrezaei, Mohammad Ebrahim Shiri, and Amir Masoud Rahmani. 2018. Identifying fake accounts on social networks based on graph analysis and classification algorithms. *Security and Communication Networks* 2018 (2018).

[20] Keisuke Nemoto, Ryuhei Hamaguchi, Tomoyuki Imaizumi, and Shuhei Hikosaka. 2018. Classification of rare building change using cnn with multi-class focal loss. In *IGARSS 2018-2018 IEEE International Geoscience and Remote Sensing Symposium*. IEEE, 4663–4666.

[21] S. Pan, R. Hu, S. F. Fung, G. Long, J. Jiang, and C. Zhang. 2019. Learning graph embedding with adversarial training methods. *IEEE Transactions on Cybernetics* 50, 6 (2019), 2475–2487.

[22] Shirui Pan and Xingquan Zhu. 2013. Graph classification with imbalanced class distributions and noise. In *Twenty-Third International Joint Conference on Artificial Intelligence*. Citeseer.

[23] Joonhyung Park, Jaeyun Song, and Eunho Yang. 2021. GraphENS: Neighbor-Aware Ego Network Synthesis for Class-Imbalanced Node Classification. In *International Conference on Learning Representations*.

[24] Carey E Priebe, Youngser Park, Joshua T Vogelstein, John M Conroy, Vince Lyzinski, Minh Tang, Avanti Athreya, Joshua Cape, and Eric Bridgeford. 2019. On a two-truths phenomenon in spectral graph clustering. *Proceedings of the National Academy of Sciences* 116, 13 (2019), 5995–6000.

[25] Liang Qu, Huaisheng Zhu, Ruiqi Zheng, Yuhui Shi, and Hongzhi Yin. 2021. Imgagn: Imbalanced network embedding via generative adversarial graph networks. In *Proceedings of the 27th ACM SIGKDD Conference on Knowledge Discovery & Data Mining*. 1390–1398.

[26] Tim Salimans, Ian Goodfellow, Wojciech Zaremba, Vicki Cheung, Alec Radford, and Xi Chen. 2016. Improved techniques for training gans. In *Advances in neural information processing systems*. 2234–2242.

[27] Prithviraj Sen, Galileo Namata, Mustafa Bilgic, Lise Getoor, Brian Galligher, and Tina Eliassi-Rad. 2008. Collective classification in network data. *AI magazine* 29, 3 (2008), 93–93.

[28] Min Shi, Yufei Tang, Xingquan Zhu, David Wilson, and Jianxun Liu. 2020. Multi-class imbalanced graph convolutional network learning. In *Proceedings of the Twenty-Ninth International Joint Conference on Artificial Intelligence (IJCAI-20)*.

[29] Shuhao Shi, Kai Qiao, Shuai Yang, Linyuan Wang, Jian Chen, and Bin Yan. 2021. Boosting-GNN: Boosting Algorithm for Graph Networks on Imbalanced Node Classification. *Frontiers in Neurorobotics* (2021), 154.

[30] AH Schistad Solberg, Geir Storvik, Rune Solberg, and Espen Volden. 1999. Automatic detection of oil spills in ERS SAR images. *IEEE Transactions on geoscience and remote sensing* 37, 4 (1999), 1916–1924.

[31] Jingru Tan, Changbao Wang, Buyu Li, Quanquan Li, Wanli Ouyang, Changqing Yin, and Junjie Yan. 2020. Equalization loss for long-tailed object recognition. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*. 11662–11671.

[32] Petar Veličković, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Lio, and Yoshua Bengio. 2017. Graph attention networks. *arXiv preprint arXiv:1710.10903* (2017).

[33] Zonghan Wu, Shirui Pan, Fengwen Chen, Guodong Long, Chengqi Zhang, and Philip S Yu. 2019. A comprehensive survey on graph neural networks. *arXiv preprint arXiv:1901.00596* (2019).

[34] Shunxin Xiao, Shiping Wang, Yuanfei Dai, and Wenzhong Guo. 2022. Graph neural networks in node classification: survey and evaluation. *Machine Vision and Applications* 33, 1 (2022), 1–19.

[35] Tianxiang Zhao, Xiang Zhang, and Suhang Wang. 2021. Graphsmote: Imbalanced node classification on graphs with graph neural networks. In *Proceedings of the 14th ACM international conference on web search and data mining*. 833–841.

Algorithm 1 GraphBoost: K Stage Boosting Node Generation Algorithm with GNN

Data: Adjacency matrix $\mathbf{A}$, node feature $\mathbf{X}$, initialized classifier C and initialized generator G, empty validation accuracy vector v^{val} and $v^{val,aggre}$, dimension of LSE embedding d_u, the number of stages K, the number of epochs $T_{c,k}$ and $T_{g,k}$ for C and G in stage k respectively, the batch size of fake nodes B. **Optional**: generator G^{other} from other methods (e.g. the generator in Park et al. [23]).

Output: Prediction $\tilde{\mathbf{Y}}$

Embed adjacency matrix $\mathbf{A}$ through LSE with dimension d_u and return embedding matrix $\mathbf{U}$;

Stage $k = 1$**:** Set epoch index $iter_T = 0$;

while $iter_T \leq T_{c,1}$ **do**

(Augment $\mathbf{A}$ and $\mathbf{X}$ with synthetic nodes from G^{other};)

Train C by minimizing $\mathcal{L}_{sup}$ in equation 2; $iter_T = iter_T + 1$;

end

Save the weights of trained classifier C in stage $k = 1$;

Obtain index set of misclassified nodes in training set I_{mis};

$k = k + 1$;

while $k \leq K$ **do**

Stage k**:** Train generator G for $T_{g,k}$ epochs based on target nodes in I_{mis} through loss function L_G in Section 4.1;

Set epoch index $iter_T = 0$;

while $iter_T \leq T_{c,k}$ **do**

(Augment $\mathbf{A}$ and $\mathbf{X}$ with synthetic nodes from other generator G^{other};)

Apply generator G to sample feature $\mathbf{X}$ and embedding $\mathbf{U}$ of B synthetic nodes.

Combine the fake nodes V^G to the graph and obtain $\tilde{\mathbf{A}}$ and $\tilde{\mathbf{X}}$ from Equation 5 (equation 7 if G^{other} is included) and Equation for $\tilde{\mathbf{X}}$ in Section 4.1;

Train C by minimizing $\mathcal{L}_C$ in Section 3.2 ; $iter_T = iter_T + 1$;

end

Save the weights of trained classifier C in stage k;

Update the misclassified index set I_{mis} as misclassified nodes in this stage and v_k^{val} based on validation score of classifier C;

Obtain aggregate classifier $\tilde{C}_{1:k}$ (validation accuracy at each stage, v_k^{val}, as voting weight) and update validation score of $\tilde{C}_{1:k}$ to $v_k^{val,aggre}$; $k = k + 1$;

end

Select $\tilde{C}_{1:k}$ $(k = 1, 2, ..., K)$ with best $v_k^{val,aggre}$ and produce prediction $\tilde{Y}$ based on this model. $\tilde{Y}$ is then the **Output**.

A ADDITIONAL ILLUSTRATIONS

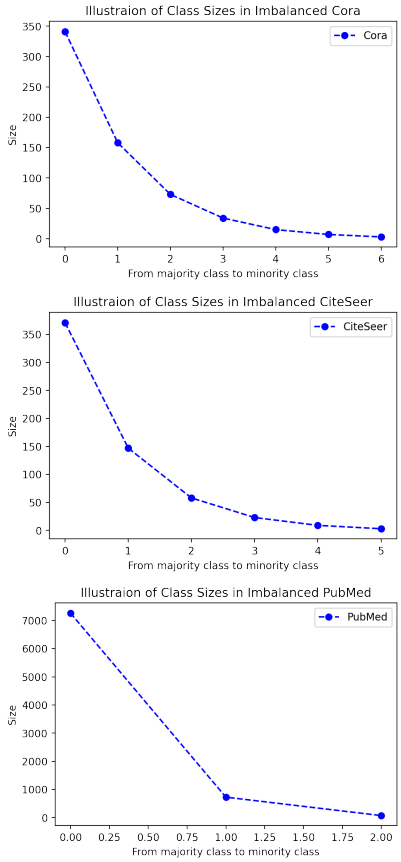

Figure 3: The Upper, Middle, and Lower graphs represent the node class sizes of training set in imbalanced Cora-LT, Citeseer-LT, and Pubmed-LT, respectively. The sizes of classes show a long tailed trend.

B ABLATION: CHOICE OF EMBEDDING DIMENSION IN LSE

In GraphBoost, the embedding of adjacency matrix $\mathbf{A}$ is critical for capturing the topological structure of the original graph. When applying LSE, the dimension of embedding vector d_u needs to be carefully chosen. In this experiment, we apply the Bayesian information criterion (BIC) to choose dimension d_u. The formula of BIC is $BIC = -2\log(L) + \log(\frac{n(n-1)}{2})d_u$, where $\log(L)$ is the log-likelihood of the adjacency matrix given the embedding vectors, n is the size of the adjacency matrix and d is the embedding dimension. Recall that the LSE embedding $\mathbf{U} \in \mathbb{R}^{n\times d_u}$ recovers a probability matrix of node links, $\mathbf{U}\mathbf{U}^T$. $\log(L)$ can thus be calculated based on the observed adjacency matrix $\mathbf{A}$ and estimated probability matrix $\mathbf{U}\mathbf{U}^T$. For the dataset, Cora/CiteSeer/PubMed, directly calculating the BIC score on the whole adjacency matrix won't provide a reasonable result, as the adjacency matrix $\mathbf{A}$ is highly sparse. Therefore, we apply a mask matrix on A and $\mathbf{U}\mathbf{U}^T$ when calculating the likelihood, such that the non-zero part of $\mathbf{A}$ is kept while the zero part of $\mathbf{A}$ is randomly sampled to achieve the same size as the non-zero part. Denote the ratio of the masked part and the original adjacency matrix size to be r. Such sampling will affect the log-likelihood part of $\log(L)$, thus, we will also apply r on the penalty term. The BIC score for this masked likelihood will then be $BIC = -2\log(L_{mask}) + r\log(N)d_u$. The BIC scores corresponding to different embedding dimensions for Cora and CiteSeer are shown in Figure 4. The recommended dimensions d_u for LSE embedding are 27 for the dataset CORA, 19 for the dataset CiteSeer and 86 for PubMed, such that d_u is associated with the lowest BIC score.

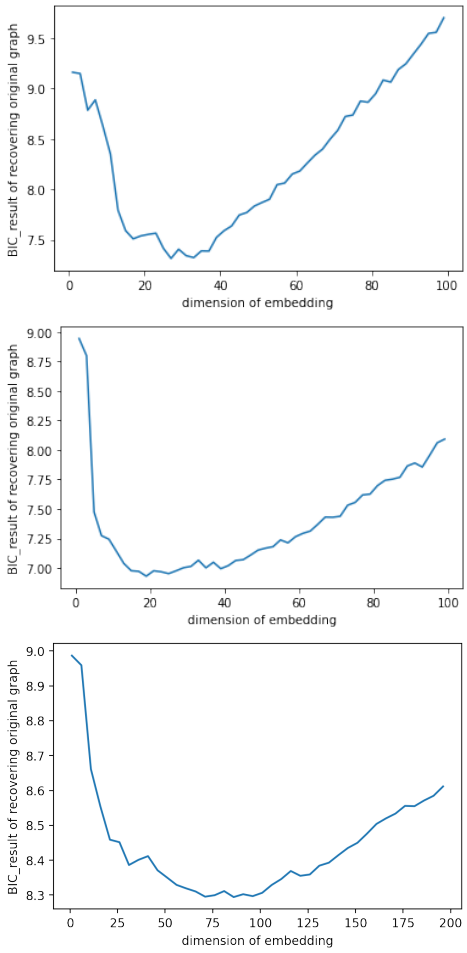

Figure 4: The Upper, Middle, and Lower graphs represent the trend of BIC score with LSE dimension d_u on the adjacency matrix of data Cora, CiteSeer, and PubMed, respectively. Dimension d_u associated with lowest BIC score is 27 for the dataset CORA, 19 for the dataset CiteSeer and 86 for PubMed, respectively.