

Graph Classification with Imbalanced Class Distributions and Noise*

Shirui Pan and Xingquan Zhu

Centre for Quantum Computation & Intelligent Systems, FEIT
University of Technology Sydney, Australia
shirui.pan@student.uts.edu.au; xingquan.zhu@uts.edu.au

Abstract

Recent years have witnessed an increasing number of applications involving data with structural dependency and graph representations. For these applications, it is very common that their class distribution is imbalanced with minority samples being only a small portion of the population. Such imbalanced class distributions impose significant challenges to the learning algorithms. This problem is further complicated with the presence of noise or outliers in the graph data. In this paper, we propose an imbalanced graph boosting algorithm, igBoost, that progressively selects informative subgraph patterns from imbalanced graph data for learning. To handle class imbalance, we take class distributions into consideration to assign different weight values to graphs. The distance of each graph to its class center is also considered to adjust the weight to reduce the impact of noisy graph data. The weight values are integrated into the iterative subgraph feature selection and margin learning process to achieve maximum benefits. Experiments on real-world graph data with different degrees of class imbalance and noise demonstrate the algorithm performance.

1 Introduction

Graph classification has drawn increasing interests due to the rapid rising of applications involving complex network structured data with dependency relationships. Typical examples include identifying bugs in a computer program flow [Cheng *et al.*, 2009], categorizing scientific publications using co-authorship [Aggarwal, 2011], and predicting chemical compound activities in bioassay tests [Deshpande *et al.*, 2005].

There have been a number of studies on graph classification in recent years [Zhu *et al.*, 2012; Saigo *et al.*, 2009; Fei and Huan, 2010; Pan *et al.*, 2013; Kashima *et al.*, 2004]. Conventional algorithms can be roughly distinguished into two categories: (1) distance based approaches (including graph kernel, graph embedding, and transformation) [Kashima *et al.*, 2004; Riesen and Bunke, 2009]; and (2) discriminative subgraph feature based approaches [Saigo *et al.*, 2009; Kong and Yu, 2010]. The former utilizes distance measures to assess distance similarities between graph pairs for learning, and the latter focuses on selecting discriminative subgraph features from the training graphs. After obtaining discriminative subgraph patterns, one can: (1) transfer training graphs into vector format (by examining the appearance of subgraph features in each graph), so existing machine learning algorithm (such as Support Vector Machine (SVM) or Decision Trees) can be applied [Deshpande *et al.*, 2005], or (2) use each subgraph pattern as a decision stump and employ boosting style algorithms for graph classification [Fei and Huan, 2010; Saigo *et al.*, 2009].

The above algorithms, especially the discriminative subgraph based algorithms, have demonstrated good classification performances for graph databases with balanced class distributions (*i.e.* the percentage of samples in different classes is close to each other). However, in reality, class imbalance commonly exists in many applications. For instance, in NCI chemical compound graph datasets (available from <http://pubchem.ncbi.nlm.nih.gov>), only about 5% percent of molecules are active to the anti-cancer bioassay test, and the remaining 95% are inactive to the test. Learning on imbalanced datasets has been widely studied in the past years. Popular techniques include sampling [Liu *et al.*, 2009], Boosting [Leskovec and Shawe-Taylor, 2003], and SVM adapting [Veropoulos *et al.*, 1999]. He and Garcia provides a recent review on imbalanced data classification [He and Garcia, 2009]. Unfortunately, all existing learning methods for imbalanced data are only applicable to general data with vector representations, but not for graph data.

When dealing with imbalanced graph data, a simple solution is to use existing frameworks for handling generic imbalanced data [Liu *et al.*, 2009] to under-sample graphs in the majority class to obtain a relatively balanced graph dataset, and then apply some existing graph classification methods. However, as pointed out by Akbani *et al.* [2004], under-sampling imbalanced data may lose valuable information and cause a large angle between the ideal and learned hyperplane for margin based algorithms (*i.e.*, SVM), and eventually end up with poor performance. This problem will be further aggravated with the presence of noise (*i.e.* mislabeled samples) or outliers. For graph applications, due to the complexity

*This work is sponsored by Australian Research Council (ARC) Future Fellowship under Grant No. FT100100971.

in examining and labeling the structural networks, it is very difficult to obtain a completely noise-free dataset. So effective designs for handling imbalanced and noisy graph data are highly desirable. In summary, when classifying imbalanced and noisy graph data, the challenges caused by subgraph feature selection and classification are mainly threefolds:

- **Bias of subgraph features:** Because the whole dataset is dominated by graphs from the majority class, traditional subgraph selection based methods tend to select subgraph features presenting frequently in the majority (negative) class, which makes selected subgraphs less discriminative for minority class.
- **Bias of learning models:** Low presence of minority (positive) graphs makes the learning models (such as Boosting or SVM) bias to the majority class and eventually result in inferior performance on the minority class. In an extreme case, if the minority class samples are extremely rare, the classifier would tend to ignore minority class samples, and classify all graphs as negatives.
- **Impact of noise and outlier:** Most algorithms (*i.e.*, Boosting) are sensitive to noise and outliers, because in their designs, if a sample's predicted label is different from the sample's label, it will receive a larger weight value. As a result, the decision boundaries of the classifier is misled by noise, and eventually result in deteriorated accuracy.

To solve the above challenges, we propose, in this paper, a linear programming boosting algorithm, igBoost, for imbalanced graph classification. Our theme is to assign different weight to each graph, by taking class distributions and the importance of individual samples into consideration. To boost the classification performance, we use graph weight values to guide the discriminative subgraph feature selection (so subgraph selection can focus on important samples), and to guide a boosted linear programming task (so classifiers can also focus on important graphs to update the model). The key contributions of the paper include:

1. The proposed igBoost design is the first algorithm with capability to handle both imbalanced class distributions and noise in the graph datasets.
2. While existing graph classification methods consider subgraph feature section and model learning as two separated procedures, our algorithm provides an effective design to integrate subgraph mining (feature selection) and model learning (margin maximization) into a unified framework, so two procedures can mutually benefit each other to achieve a maximization goal.
3. Experiments on real-world data show that igBoost achieves very significant performance gain for imbalanced graph classification.

2 Preliminaries and Problem Definitions

2.1 Problem Definitions

DEFINITION 1 Connected Graph: A graph is denoted as $G = (\mathcal{V}, E, \mathcal{L})$, where $V = \{v_1, \dots, v_{n_v}\}$ is the vertices set,

$E \subseteq \mathcal{V} \times \mathcal{V}$ is the edge set, and $\mathcal{L}$ is a labeling function to assign labels to a node or an edge. A connected graph is a graph such that there is a path between any pair of vertices.

Each connected graph G_i is associated with a class label y_i , $y_i \in \mathcal{Y} = \{-1, +1\}$. In this paper, we consider $y_i = +1$ as the minority (positive) class, and $y_i = -1$ in the majority (negative) class.

DEFINITION 2 Subgraph: Given two graphs $G = (\mathcal{V}, E, \mathcal{L})$ and $g_i = (\mathcal{V}', E', \mathcal{L}')$, g_i is a subgraph of G (*i.e.* $g_i \subseteq G$) if there is an injective function $f: \mathcal{V}' \rightarrow \mathcal{V}$, such that $\forall (a, b) \in E'$, we have $(f(a), f(b)) \in E$, $\mathcal{L}'(a) = \mathcal{L}(f(a))$, $\mathcal{L}'(b) = \mathcal{L}(f(b))$, $\mathcal{L}'(a, b) = \mathcal{L}(f(a), f(b))$. If g_i is a subgraph of G ($g_i \subseteq G$), then G is a supergraph of g_i ($G \supseteq g_i$).

DEFINITION 3 Noisy graph samples and Outliers: Given a graph dataset $T = \{(G_1, y_1), \dots, (G_n, y_n)\}$, a noisy graph sample is a graph whose label is incorrectly labeled (*i.e.*, a positive graph is labeled as negative, or vice versa), and an outlier is a graph which is far away from its class center.

Graph Classification: Given a set of connected graph data $T = \{(G_1, y_1), \dots, (G_n, y_n)\}$, each of which is labeled as either positive or negative, our **aim** is to learn a classifier from the training graphs T to predict unseen test graphs with maximum accuracy on both classes.

2.2 Classification on Balanced Graph Data

The classification task on balanced graph data can be regarded as learning a hyperplane in high dimension space to maximize the soft margin between positive and negative examples [Saigo *et al.*, 2009], as show in Eq.(1), where n denotes the number of training graphs, m is the number of weak classifiers, and ξ_i is the slack variable:

$$\begin{aligned} \max_{w, \rho, \xi \in \mathbb{R}_+^n} \quad & \rho - D \sum_{i=1}^n \xi_i \\ \text{s. t.} \quad & y_i \sum_{j=1}^m w_j \cdot \hat{h}_{g_j}(G_i) + \xi_i \geq \rho, \quad i = 1, \dots, n \\ & \sum_{j=1}^m w_j = 1, \quad w_j \geq 0, \quad j = 1, 2 \dots m \end{aligned} \quad (1)$$

In Eq.(1), the first term ρ aims to maximize the margin, and the second term ξ_i attempts to minimize the penalty of misclassification for G_i , with parameter D denoting the misclassification weight. $\hat{h}_{g_j}(G_i)$ is the class label of G_i given by a weak subgraph classifier g_j , and w_j is its weight. For balanced graph data, this objective function aims to minimize the total number of misclassifications. For imbalanced graph datasets, the classifier will be biased to the majority class, and generate hyperplanes in favoring to classify examples as negatives. The idea of hyperplane for imbalanced graph datasets is very similar to the analysis in [Akbari *et al.*, 2004], where they use SVM to handle imbalanced vector datasets.

The second drawback of Eq.(1) is that it is sensitive to noisy graphs, even if the second term (the slack variable) allows some samples (such as noise) to appear at the opposite side of the hyperplane. This is mainly because there is no effective way to reduce the impact of noisy samples. So noise will continuously complicate the decision boundary and deteriorate the learning accuracy.

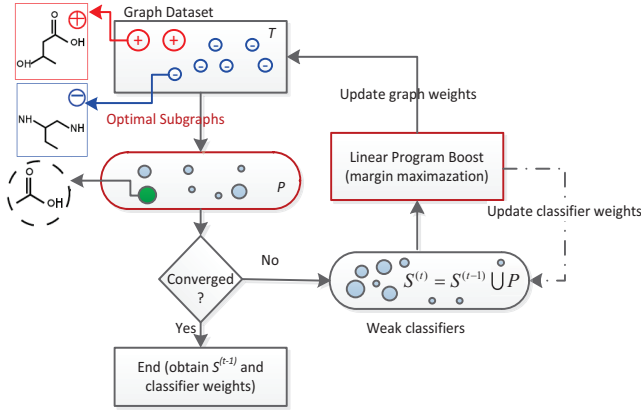


Figure 1: The proposed framework for imbalanced and noisy graph data classification. Initially, each positive graph in dataset T is assigned a weight value β times larger than a negative graph (the size of the circle corresponds to the graph weight). igBoost iteratively selects optimal subgraph features P from T , with the selected subgraph features P being added into a global set S . Afterward, igBoost solves a linear programming problem to get two sets of weights: (1) weights for training graphs, and (2) weights for weak learners (subgraphs). The feature selection and margin maximization procedures continue until igBoost converges.

3 Boosting for Imbalanced and Noisy Graphs

For the graph data $T = \{(G_1, y_1), \dots, (G_n, y_n)\}$, let $\mathcal{F} = \{g_1, \dots, g_m\}$ be the full set of subgraphs in T . We can use $\mathcal{F}$ as features to represent each graph G_i into a vector space as $x_i = \{x_i^{g_1}, \dots, x_i^{g_m}\}$, where $x_i^{g_j} = 1$ if $g_j \in G_i$, and otherwise 0. For imbalanced and noisy graph data, we propose an igBoost algorithm, which combines subgraph feature selection and graph classification into a framework, as shown in Fig. 1.

For a subgraph g_j , we can use it as a decision stump classifier (g_j, π_j) as follows:

$$h(G_i; g_j, \pi_j) = \begin{cases} \pi_j & : g_j \in G_i \\ -\pi_j & : g_j \notin G_i \end{cases} \quad (2)$$

Here, $\pi_j \in \mathcal{Q} = \{-1, +1\}$ is a parameter controlling the label of the classifier. As a result, the prediction rule for a graph G_i is a linear combination of the weak classifiers:

$$\mathcal{H}(G_i) = \sum_{(g_j, \pi_j) \in \mathcal{F} \times \mathcal{Q}} w_j h(G_i; g_j, \pi_j) \quad (3)$$

where w_j is the weight for classifier $h(G_i; g_j, \pi_j)$.

To effectively handle imbalanced class distributions and noisy graphs, our proposed learning algorithm is formulated as the following linear programming optimization problem:

$$\begin{aligned} \max_{w, \rho, \xi \in \mathbb{R}_+^N} \quad & \rho - D(\beta \sum_{\{i|y_i=+1\}} \varphi_i \xi_i + \sum_{\{j|y_j=-1\}} \varphi_j \xi_j) \\ \text{s. t.} \quad & y_i \sum_{j=1}^m w_j \cdot h(G_i; g_j, \pi_j) + \xi_i \geq \rho, i = 1 \dots n \\ & \sum_{j=1}^m w_j = 1, w_j \geq 0 \quad j = 1, 2 \dots m \end{aligned} \quad (4)$$

In the above objective function, n^+ and n^- denote the number of graphs in positive and negative classes ($n = n^+ + n^-$), respectively. D is a parameter controlling the cost of misclassification. Typically, $D = \frac{1}{vn}$ in our algorithm. In addition, there are two key components in our objective function:

- **Weight values for graphs in different classes:** Because the number of positive graphs is much smaller than negative graphs, they should carry higher misclassification penalties to prevent positive graphs from being overlooked for learning. In our formulation, the weights of graphs in the positive class are β times higher than the weights of negative graphs. The adjusted weight values, with respect to the class distributions, can help reduce the effect of class imbalance and prevent the learning model to be biased towards the majority class.
- **Weight values for graphs within the same class:** To handle noise and outliers, we introduce a membership value φ_i , for each graph G_i , to indicate how likely G_i is considered to be an outlier (a smaller φ_i values indicating that G_i is more likely being an outlier). By doing so, our algorithm can leverage the weight of individual graph to reduce the impact of outliers and noise on the learning process.

For the φ_i value in Eq.(4), we consider the density of each graph G_i to determine its likelihood score of being an outlier. Intuitively, if G_i is located far away from its class center, it is more likely being an outlier or a noisy graph (so φ_i will have a smaller value). Therefore, our approach to calculate φ_i is given as follows:

$$\varphi_i = \frac{2}{1 + e^{\tau d(G_i)}}; \quad (5)$$

In Eq. (5), $d(G_i)$ denotes the distance of graph G_i to its class center, and $\tau \in [0, 1]$ is a decay factor controlling the magnitude changing by the distance.

For graph classification, our learning problem in Eq.(4) requires a whole feature set $\mathcal{F} = \{g_1, \dots, g_m\}$. However, we cannot obtain this feature set unless we enumerate the whole graph set, which is NP-complete. Therefore, Eq.(4) cannot be solved directly. *Column generation (CG)* [Nash and Sofer, 1996], a classic optimization technique, provides an alternative solution to solve this problem. Instead of directly solving the primal problem in Eq.(4), the CG technique works on the dual problem. Starting from an empty set of constraints, CG iteratively selects the most violated constraints until there is no more violated constraints. The final optimal solution, under the set of iteratively selected constraints, is equal to the optimal solution under all constraints.

We can write the dual problem of Eq.(4) as follows:

$$\begin{aligned} \min_{\mu, \gamma} \quad & \gamma \\ \text{s. t.} \quad & y_i \sum_{j=1}^n \mu_j \cdot h(G_i; g_j, \pi_j) \leq \gamma, \quad j = 1, 2 \dots m \\ & 0 \leq \mu_i \leq \beta \varphi_i D \quad \text{if } y_i = +1 \\ & 0 \leq \mu_i \leq \varphi_i D \quad \text{if } y_i = -1 \\ & \sum_{i=1}^n \mu_i = 1, \quad \mu_i \geq 0 \end{aligned} \quad (6)$$

Note that each constraint $y_i \sum_{i=1}^n \mu_i \cdot \bar{h}(G_i; g_j, \pi_j) \leq \gamma$ in Eq.(6) indicates a subgraph pattern (g_j, π_j) over the whole training graphs, and we have m constraints in total (equal to the number of subgraphs which is very large or probably infinite), whereas in Eq. (4) there are only n constraints (equals to number of training graphs).

Although the number of constraints in Eq. (6) may increase ($m \gg n$), we can solve it more effectively by combining the subgraph mining and CG techniques. Specifically, we (1) first discover the top- k optimal subgraph patterns that violate the constraints most in each iteration, and (2) solve the sub-problem based on the selected top- k constraints. After solving Eq.(6) based on selected constraints, we can obtain $\{\mu_i\}_{i=1, \dots, n}$, which can be regarded as the new weights for training graphs, so that we can iteratively perform subgraph feature selection in the next round (See Fig.1). Such top- k constraints techniques are known as *Multiple Prices* [Luenberger, 1997] in column generation.

To apply multiple prices, We first define the *discriminative score* for each subgraph based on the constraints in Eq.(6).

DEFINITION 4 Discriminative Score: for a subgraph decision stump (g_j, π_j) , its discriminative score is defined as:

$$\mathbf{i}(g_j, \pi_j) = y_i \sum_{i=1}^n \mu_i \cdot \bar{h}(G_i; g_j, \pi_j) \quad (7)$$

We can sort the subgraph patterns based on their discriminative scores in a descending order, and select the top- k subgraphs, which form the most violated constraints.

Suppose $\mathcal{S}^{(t)}$ is the set of decision stumps (subgraphs) discovered by column generation so far at t^{th} iteration. Let $\gamma^{(t)}$ and $u^{(t)}$ be the optimal solution for t^{th} iteration. Then our algorithm solves such a linear problem in t^{th} iteration.

$$\begin{aligned} \min_{\gamma^{(t)}, \mu^{(t)}} \quad & \gamma^{(t)} \\ \text{s.t.} \quad & y_i \sum_{i=1}^n \mu_i^{(t)} \bar{h}(G_i; g_j, \pi_j) \leq \gamma^{(t)}, \forall (g_j, \pi_j) \in \mathcal{S}^{(t)} \\ & 0 \leq \mu_i^{(t)} \leq \beta \varphi_i D \quad \text{if } y_i = +1 \\ & 0 \leq \mu_i^{(t)} \leq \varphi_i D \quad \text{if } y_i = -1 \\ & \sum_{i=1}^n \mu_i^{(t)} = 1, \quad \mu_i \geq 0 \end{aligned} \quad (8)$$

Note that in Eq.(8), the value φ_i will change at each iteration. Specifically, we first compute the class centers for positive and negative graphs respectively, by using current selected subgraphs $\mathcal{S}^{(t)}$ (transfer each graph as a vector based on $\mathcal{S}^{(t)}$), and then obtain the value φ_i according to Eq. (5).

Our imbalanced graph boosting framework is illustrated in Algorithm 1. To handle class imbalance, the initial weights of each positive graph μ_i is set to β times of negative graphs (step 1). Then igBoost iteratively selects top- k subgraphs $P = \{(g_i, \pi_i)\}_{i=1, \dots, k}$ at each round (step 5). In step 6, we obtain the most optimal score $\mathbf{i}(g^*, \phi^*)$. If the current optimal pattern no longer violates the constraint, the iteration process stops (steps 7-8). Because in the last few iterations, the optimal value only change subtly, we add a small value ε to relax the stopping condition (typically, we use $\varepsilon = 0.05$ in

Algorithm 1 Boosting for Imbalanced Graph Classification

Require:

$T = \{(G_1, y_1), \dots, (G_n, y_n)\}$: Graph Datasets

Ensure:

$\mathcal{H}(G_i) = \sum_{(g_j, \pi_j) \in \mathcal{S}^{(t-1)}} w_j^{(t-1)} \bar{h}(G_i; g_j, \pi_j)$: Classifier;

1: $\mu_i = \begin{cases} \varsigma^+ & : y_i = +1 \\ \varsigma^- & : y_i = -1 \end{cases}$, where $\frac{\varsigma^+}{\varsigma^-} = \beta$, $\sum_{i=1}^n \mu_i = 1$;

2: $\mathcal{S}^{(0)} \leftarrow \emptyset$; $\gamma^{(0)} \leftarrow 0$;

3: $t \leftarrow 0$;

4: **while** true **do**

5: Obtain top- k subgraph decision stumps $P = \{(g_i, \pi_i)\}_{i=1, \dots, k}$; //Algorithm 2;

6: $\mathbf{i}(g^*, \pi^*) = \max_{(g_j, \pi_j) \in P} \mathbf{i}(g_j, \pi_j)$

7: **if** $\mathbf{i}(g^*, \pi^*) \leq \gamma^{(t-1)} + \varepsilon$ **then**

8: **break**;

9: $\mathcal{S}^{(t)} \leftarrow \mathcal{S}^{(t-1)} \cup P$;

10: Obtain the membership value φ_i for each graph example G_i based on $\mathcal{S}^{(t)}$ and Eq. (5);

11: Solve Eq. (8) to get $\gamma^{(t)}$, $\mu^{(t)}$, and Lagrange multipliers $w^{(t)}$;

12: $t \leftarrow t + 1$;

13: **return** $\mathcal{H}(G_i) = \sum_{(g_j, \pi_j) \in \mathcal{S}^{(t-1)}} w_j^{(t-1)} \bar{h}(G_i; g_j, \pi_j)$;

our experiments). In steps 9-11, we solve the linear programming problem based on the selected subgraphs to recalculate two set of weights: (1) $\{\mu_i\}_{i=1, \dots, n}$, the weights of training graph for optimal subgraph mining in the next round; (2) $\{w_j\}_{j=1, \dots, |\mathcal{S}^{(t)}|}$, the weights for subgraph decision stumps in $\mathcal{S}^{(t)}$, which can be obtained from the Lagrange multipliers of dual problem. Once the algorithm converges, igBoost returns the final classifier model $\mathcal{H}(G_i)$ in step 13.

Optimal subgraphs mining. In order to mine the top- k subgraphs in step 5 of Algorithm 1, we need to enumerate the entire set of subgraph patterns from the training graphs T . In igBoost, we employ a Depth-First-Search (DFS) based algorithm gSpan [Yan and Han, 2002] to enumerate subgraphs. The key idea of gSpan is that each subgraph has a unique DFS Code, which is defined by a lexicographic order of the discovery time during the search process. By employing a depth first search strategy on the DFS Code tree (where each node is a subgraph), gSpan can enumerate all frequent subgraphs efficiently. To speed up the enumeration, we utilize a branch-and-bound pruning rule [Saigo *et al.*, 2009] to prune the search space effectively:

Theorem 1 Given a subgraph feature g_j , let

$$s_+^{(g_j)} = 2 \sum_{\{i|y_i=+1, g_j \in G_i\}} \mu_i^{(t)} - \sum_{i=1}^n y_i \mu_i \quad (9)$$

$$s_-^{(g_j)} = 2 \sum_{\{i|y_i=-1, g_j \in G_i\}} \mu_i^{(t)} - \sum_{i=1}^n y_i \mu_i \quad (10)$$

$$\mathbf{i}(g_j) = \max(s_+^{(g_j)}, s_-^{(g_j)}) \quad (11)$$

If $g_j \subseteq g'$, then the discriminative score $\mathbf{i}(g', \pi') \leq \mathbf{i}(g_j)$.

Because a subgraph decision stump can take class label from $\mathcal{Q} = \{+1, -1\}$, we calculate its maximum score based on each possible value, and select the maximum one as the upperbound. Due to page limitations, we omit the proof here.

Algorithm 2 Optimal Subgraphs Mining

Require:

$T = \{(G_1, y_1), \dots, (G_n, y_n)\}$: Graph Datasets;
 $\mu = \{\mu_1, \dots, \mu_n\}$: Weights for graph examples;
 k : Number of optimal subgraph patterns;

Ensure:

$P = \{(g_i, \pi_i)\}_{i=1, \dots, k}$: The top- k subgraphs;
1: $\eta = 0, P \leftarrow \emptyset$;
2: **while** Recursively visit the DFS Code Tree in gSpan **do**
3: $g_p \leftarrow$ current visited subgraph in DFS Code Tree;
4: **if** g_p has been examined **then**
5: **continue**;
6: Compute score $\mathbf{i}(g_p, \pi_p)$ for subgraph g_p according Eq.(7);
7: **if** $|P| < k$ or $\mathbf{i}(g_p, \pi_p) > \eta$ **then**
8: $P \leftarrow P \cup (g_p, \pi_p)$;
9: **if** $|P| > k$ **then**
10: $(g_q, \pi_q) \leftarrow \arg \min_{(g_x, \pi_x) \in P} \mathbf{i}(g_x, \pi_x)$;
11: $P \leftarrow P / (g_q, \pi_q)$;
12: $\eta \leftarrow \min_{(g_x, \pi_x) \in P} \mathbf{i}(g_x, \pi_x)$
13: **if** The upperbound of score $\mathbf{i}(g_p) > \eta$ **then**
14: Depth-first search the subtree rooted from node g_p ;
15: **return** $P = \{(g_i, \pi_i)\}_{i=1, \dots, k}$;

From Theorem 1, we know that once a subgraph g_j is generated, all its super-graphs are upperbounded by $\mathbf{i}(g_j)$. Therefore, this theorem can help prune the search space efficiently.

Our branch-and-bound subgraph mining algorithm is listed in Algorithm 2. The minimum value η and optimal set P are initialized in step 1. We prune the duplicated subgraph features in step 4-5, and compute the discriminative score $\mathbf{i}(g_p, \pi_p)$ for g_p in step 6. If $\mathbf{i}(g_p, \pi_p)$ is larger than η or the current set P has less than k subgraph patterns, we add (g_p, π_p) to the feature set P (steps 7-8). Meanwhile, when the size of P exceeds the predefined size k , we remove subgraph with the minimum discriminative score (steps 9-11). We use branch-and-bound pruning rule, according to Theorem 1, to prune the search space in steps 13. Note that this is essential for our mining algorithm, because we don't provide a support threshold for subgraph mining. Finally, the optimal set P is returned in step 15.

4 Experimental Study

Benchmark datasets: We validate the performance of the proposed algorithm on real-world imbalanced chemical compounds datasets collected from PubChem¹. Each chemical compound dataset belongs to a bioassay activity prediction task, where each chemical compound is represented as a graph, with atoms representing nodes and bonds as edges. A chemical compound is labeled positive if it is active against the corresponding bioassay test, or negative otherwise.

Table 1 summarizes the nine benchmark datasets used in our experiments, where columns 2-3 show the number of positive graphs and the total number of graphs in the original datasets. After removing disconnected graphs and graphs with unexpected atoms, we obtain new datasets with slightly different sizes, as shown in columns 4-5. Meanwhile, as shown in column 6, all these datasets are naturally imbalanced, with positive graphs varying from 4.4% to 31.5%.

¹<http://pubchem.ncbi.nlm.nih.gov>

Table 1: Imbalanced Chemical Compound Datasets

Bioassay ID	Original Compounds		New Compounds		Pos%
	#Pos	#Total	#Pos	#Total	
602330	85	1927	80	1839	4.4
602125	110	1297	90	1217	7.4
1431	109	1260	104	1217	8.6
624146	105	1285	105	1096	9.6
624319	211	1405	197	1355	14.5
1878	195	1048	194	1026	18.9
2102	212	1063	193	1006	19.2
1317	328	1215	250	1100	22.7
111	371	1056	256	812	31.5

Baselines: We compare our approach with the state-of-the-art algorithm gBoost [Saigo *et al.*, 2009], which is a boosting style graph classification algorithm and has demonstrated good performance compared to traditional frequent subgraph based algorithms (such as mining a set of subgraphs as features and then employing SVM for classification). For gBoost, we conduct experiments by using two variants to handle the class imbalance issue:

- **gBoost on balanced graph dataset (denoted as gBoost-b):** By following traditional approaches to handle imbalanced data, we first under-sample graphs in the majority (negative/inactive) class to create a balanced graph dataset, and then collect gBoost's performance on the balanced datasets.
- **gBoost on imbalanced graph dataset directly (denoted as gBoost-Im):** The gBoost algorithm is directly applied to the imbalanced graph datasets.

Our algorithm igBoost employs two key components to handle imbalanced and noisy graph datasets: (1) different weight values between positive vs. negative classes, and (2) nonuniform weights for examples within the same class. To evaluate the effectiveness of each individual components, we first validate the first component by only adding β value to the objective function in Eq. (4), with this method denoted by **igBoost- β** . Then we add both β and the membership value φ_i to each graph, so the algorithm considers class imbalance and noise in the graphs as the same time. We denote this algorithm by **igBoost**.

Measurement: For imbalanced datasets, accuracy is no longer an effective metric to assess the algorithm performance, so we use AUC (Area Under the ROC Curve) as the performance measurement in our experiments.

The parameters for all algorithms are set as follows: k is set to 25; v ($D = \frac{1}{nv}$) is varied in range $\{0.2, 0.3, 0.4, 0.5\}$ for all algorithms. $\beta = \frac{|Neg|}{|Pos|}$ is the imbalanced ratio, and the decay factor τ in Eq. (5) is chosen from $\{0.05, 0.1, 0.2, 0.3, 0.4\}$ for igBoost algorithms. For each combination of these parameters, we conduct 5-cross-validation on each dataset and record average result (*i.e.*, we perform 20 trials of experiments with different combinations of v and τ). To compare each method as fairly as possible, we select the best result out of 20 trials of different parameters for each method.

Experimental Results: Table 2 reports the results on vari-

Table 2: AUC Values on Different Graph Datasets

Bioassay-ID	gBoost-Im	gBoost-b	igBoost- β	igBoost
602330	0.542	0.656	0.650	0.657
602125	0.732	0.872	0.937	0.944
1431	0.623	0.791	0.812	0.825
624146	0.552	0.836	0.839	0.859
624319	0.824	0.775	0.829	0.834
1878	0.864	0.854	0.874	0.884
2102	0.681	0.686	0.686	0.711
1317	0.751	0.637	0.748	0.781
111	0.802	0.775	0.793	0.825
Average	0.708	0.765	0.796	0.814

ous imbalanced graph datasets. For severely imbalanced data such as 602330, 602125, 1431, and 624146, gBoost-Im has the worst performance. This is because gBoost is designed for balanced data only. The subgraph features selected by gBoost, on highly imbalanced graph data, will be dominated by the negative class, making it less discriminative to classify positive graphs. Furthermore, the classifier learned by gBoost also favors negative class. As a result, most positive graphs will be misclassified, resulting in low AUC values (*e.g.* only 0.542 for a highly imbalanced dataset 602330 whose positive class only has 4.4% of samples).

For gBoost-b, it under-samples the imbalanced graph data before applying gBoost, which helps alleviate the skewness of highly imbalanced graph data and obtains relatively better results than gBoost-Im (on 602330, 602125, 1431, and 624146). However, under-sampling essentially changes the sample distributions and may lose valuable information in the sampled graph data, which may result in inferior results. As shown in datasets from 624319 to 111, for moderately imbalanced graph datasets, the under-sampling method gBoost-b is inferior to gBoost-Im.

The results in Table 2 demonstrate the effectiveness of the proposed algorithms and validate the two key components of igBoost. By assigning different weights to different classes, our algorithm considers the class distributions to suppress the skewness effect of imbalance and prevent the classifier from being severely biased to negative class. As a result, Table 2 demonstrates that igBoost- β outperforms gBoost-Im and gBoost-b on 7 datasets out of 9. For instance, it achieves 20.5% improvements over gBoost-Im on 602125, and 11.1% gain over gBoost-b on 1317 datasets, respectively. By integrating importance of each graph with respect to its class center, igBoost is intended to handle outliers and noise in an effective way. The results in column 5 of Table 2 validate the effectiveness of this key design. igBoost achieves noticeable improvements over igBoost- β on all datasets. The overall results in Table 2 asserts that igBoost outperforms the state-of-the-art gBoost on all datasets, regardless of whether gBoost is applied to the original imbalanced data or on balanced data by under-sampling. The average AUC values over nine benchmark datasets show that igBoost achieves 4.9% and 10.6% performance gain over gBoost-b and gBoost-Im, respectively.

Comparisons on Noisy Graph Data: To validate that igBoost is indeed robust and effective in handling noise in the

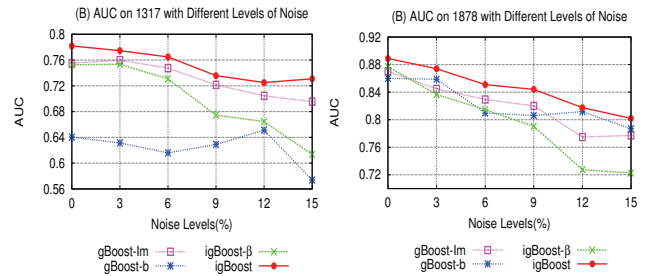


Figure 2: Experiments on Different Levels of Noise.

graph data, we manually inject some noise into the graph datasets by randomly flipping the class labels of $L\%$ of graphs in the training data. Because majority graphs are negative, such a random approach will have a more severe impact on positive class than negative class. The results in Fig. 2 show that the increase of the noise levels result in deteriorated AUC values for all algorithms. This is because noise complicates the decision boundaries and make learner difficult to separate positive and negative classes. Among all algorithms, igBoost- β suffers the most performance loss, this is because igBoost- β only considers the class imbalance, so a mislabeled noisy positive graph will receive a large weight, which significantly deteriorates the quality of the hyper-planes learnt from the data. Indeed, if a negative graph G_i is falsely labeled as a positive graph, G_i will have large distance to the class center of all positive graphs, because G_i is still close and similar to the negative graphs in the feature space. By using G_i 's distance to the class center to adjust its impact on the objective function, the results in Fig 2 confirm that igBoost outperforms all baseline algorithms under all degrees of noise. This validates that combining class distributions and distance penalties of individual graph indeed help igBoost effectively handle graph data with severely imbalanced class distributions and noise.

5 Conclusion

In this paper, we investigated graph classification with imbalanced class distributions and noise. We argued that existing endeavors inherently overlooked the class distributions in graph data, so the selected subgraph features are biased to the majority class, which makes algorithms vulnerable to imbalanced class distributions and noisy samples. In the paper, we proposed a boosting framework, which considers class distributions and the distance penalty of each graph to its class center to weight individual graphs. Based on the weight values, we combine subgraph selection and margin optimization between positive and negative graphs to form a boosting framework, so the selected subgraph can help find better margins and the optimized margins further help select better subgraph features. Experiments on real-world imbalanced and noisy graph datasets validate our design. The novelty that distinguishes igBoost from others is twofold: (1) igBoost is the first design aims to handle both class imbalance and noise for graph data; and (2) igBoost integrates subgraph mining and model learning (margin maximization) into a unified framework, whereas traditional methods separate them into two isolated procedures, without using one to benefit the other.

References

- [Aggarwal, 2011] C. Aggarwal. On classification of graph streams. In *Proc. of SDM*, Arizona, USA, 2011.
- [Akbari et al., 2004] R. Akbari, S. Kwek, and N. Japkowicz. Applying support vector machines to imbalanced datasets. *Machine Learning: ECML 2004*, pages 39–50, 2004.
- [Cheng et al., 2009] H. Cheng, D. Lo, Y. Zhou, X. Wang, and X. Yan. Identifying bug signatures using discriminative graph mining. In *ISSTA*, pages 141–152, 2009.
- [Deshpande et al., 2005] M. Deshpande, M. Kuramochi, N. Wale, and G. Karypis. Frequent substructure-based approaches for classifying chemical compounds. *IEEE Trans. on Knowl. and Data Eng.*, 17:1036–1050, 2005.
- [Fei and Huan, 2010] H. Fei and J. Huan. Boosting with structure information in the functional space: an application to graph classification. In *Proc. of ACM SIGKDD*, Washington DC, USA, 2010.
- [He and Garcia, 2009] H. He and E.A. Garcia. Learning from imbalanced data. *IEEE Transactions on Knowledge and Data Engineering*, 21(9):1263–1284, 2009.
- [Kashima et al., 2004] H. Kashima, K. Tsuda, and A. Inokuchi. *Kernels for Graphs*, chapter In: Schölkopf B, Tsuda K, Vert JP, editors. Kernel methods in computational biology. MIT Press, Cambridge (Massachusetts), 2004.
- [Kong and Yu, 2010] X. Kong and P. Yu. Semi-supervised feature selection for graph classification. In *Proc. of ACM SIGKDD*, Washington, DC, USA, 2010.
- [Leskovec and Shawe-Taylor, 2003] J. Leskovec and J. Shawe-Taylor. Linear programming boosting for uneven datasets. In *Proc. of ICML*, page 456, 2003.
- [Liu et al., 2009] X.Y. Liu, J. Wu, and Z.H. Zhou. Exploratory undersampling for class-imbalance learning. *Systems, Man, and Cybernetics, Part B: Cybernetics, IEEE Transactions on*, 39(2):539–550, 2009.
- [Luenberger, 1997] D.G. Luenberger. *Optimization by vector space methods*. Wiley-Interscience, 1997.
- [Nash and Sofer, 1996] S.G. Nash and A. Sofer. Linear and nonlinear programming. *Newyark McGraw-Hill*, 1996.
- [Pan et al., 2013] S. Pan, X. Zhu, C. Zhang, and P. S. Yu. Graph stream classification using labeled and unlabeled graphs. In *International Conference on Data Engineering (ICDE)*. IEEE, 2013.
- [Riesen and Bunke, 2009] K. Riesen and H. Bunke. Graph classification by means of lipschitz embedding. *IEEE Trans. on SMC - B*, 39:1472–1483, 2009.
- [Saigo et al., 2009] H. Saigo, S. Nowozin, T. Kadowaki, T. Kudo, and K. Tsuda. gboost: a mathematical programming approach to graph classification and regression. *Machine Learning*, 75:69–89, 2009.
- [Veropoulos et al., 1999] K. Veropoulos, C. Campbell, N. Cristianini, et al. Controlling the sensitivity of support vector machines. In *Proc. of the IJCAI*, volume 1999, pages 55–60. Citeseer, 1999.
- [Yan and Han, 2002] X. Yan and J. Han. gspan: Graph-based substructure pattern mining. In *Proc. of ICDM*, Maebashi City, Japan, 2002.
- [Zhu et al., 2012] Y. Zhu, J.X. Yu, H. Cheng, and L. Qin. Graph classification: a diversified discriminative feature selection approach. In *Proc. of CIKM*, pages 205–214. ACM, 2012.