CogBoost: Boosting for Fast Cost-Sensitive Graph Classification

Shirui Pan, Jia Wu, and Xingquan Zhu, *Senior Member, IEEE*

Abstract—Graph classification has drawn great interests in recent years due to the increasing number of applications involving objects with complex structure relationships. To date, all existing graph classification algorithms assume, explicitly or implicitly, that misclassifying instances in different classes incurs an equal amount of cost (or risk), which is often not the case in real-life applications (where misclassifying a certain class of samples, such as diseased patients, is subject to more expensive costs than others). Although cost-sensitive learning has been extensively studied, all methods are based on data with instance-feature representation. Graphs, however, do not have features available for learning and the feature space of graph data is likely infinite and needs to be carefully explored in order to favor classes with a higher cost. In this paper, we propose, CogBoost, a fast cost-sensitive graph classification algorithm, which aims to minimize the misclassification costs (instead of the errors) and achieve fast learning speed for large scale graph data sets. To minimize the misclassification costs, CogBoost iteratively selects the most discriminative subgraph by considering costs of different classes, and then solves a linear programming problem in each iteration by using Bayes decision rule based optimal loss function. In addition, a cutting plane algorithm is derived to speed up the solving of linear programs for fast learning on large scale data sets. Experiments and comparisons on real-world large graph data sets demonstrate the effectiveness and the efficiency of our algorithm.

Index Terms—Graph classification, cost-sensitive learning, subgraphs, boosting, cutting plane algorithm, large scale graphs

1 Introduction

Due to the rapid advancement in networking and data collection technology, recent years have witnessed an increasing number of applications involving data with complex structure relationships, e.g., chemical compounds [1], social networks [2], and scientific publications [3]. Different from traditional data represented in deterministic feature space by using an instance-feature representation, structure data are not represented by using attribute vectors, but by graphs with dependency relationships between objects. Because graphs do not have features immediately available for learning, this challenge has motivated a number of graph classification methods, which either directly learn global similarities between graphs measured by graph kernels or graph embedding [1], [4], or select some informative subgraphs as features to represent graphs into feature space for learning [3], [5], [6], [7], [8], [9], [10], [11]. Nevertheless, all existing methods for graph classification suffer from two major deficiencies: ineffective for cost-sensitive classification and inefficient for large scale graphs.

- *S. Pan is with the College of Information Engineering, Northwest A&F University, Yangling 712100, China, and also with the Centre for Quantum Computation & Intelligent Systems, FEIT, University of Technology Sydney, Sydney, NSW 2007, Australia. E-mail: shirui.pan@student.uts.edu.au.*
- *J. Wu is with the Centre for Quantum Computation and Intelligent Systems, FEIT, University of Technology, Sydney, NSW 2007, Australia. E-mail: jia.wu@student.uts.edu.au.*
- *X. Zhu is with the Department of Computer & Electrical Engineering and Computer Science, Florida Atlantic University, Boca Raton, FL 33431. E-mail: xzhu3@fau.edu.*

Manuscript received 5 May 2014; revised 5 Dec. 2014; accepted 26 Dec. 2014. Date of publication 11 Jan. 2015; date of current version 2 Oct. 2015. Recommended for acceptance by H. Xiong. For information on obtaining reprints of this article, please send e-mail to: reprints@ieee.org, and reference the Digital Object Identifier below. Digital Object Identifier no. 10.1109/TKDE.2015.2391115

1.1 Cost-Sensitive Graph Classification

For graph classification, all existing methods assume, explicitly or implicitly, that misclassifying a positive graph incurs an equal amount of cost (or risk) to the misclassification of a negative graph, i.e., all misclassifications are subject to the same cost (In this paper, positive class and minority class are equivalent, and they both denote the class with the highest misclassification cost). The induced decisions are commonly referred to as *cost-insensitive*. In real-life graph applications, the equal-cost assumption is rarely the case, or at least too strong. Some examples are given as follows.

Biological domains. In structure based medical diagnose [12], [13], chemical compounds active against cancer are very rare and are expected to be carefully identified and investigated. A false negative identification (i.e., predicting an active compound to be inactive) has a much more severe consequence (i.e., a higher cost) than a false positive identification (i.e., predicting an inactive compound to be active). Therefore, a false negative and a false positive are inherently different and a false negative prediction may result in the delay and wrong diagnose, leading to severe complications (or extra costs) at a later stage.

Cyber security domains. In intrusion detection systems, each traffic flow can be represented as a graph by presenting traffic destinations (such as IP addresses and port numbers) as nodes. Malicious traffics may impose threat or damage to computer servers, leading to severe security issues, such as private information leakage or internet breakdown. Therefore, misclassification of a malicious

traffic (graph) has a much higher economic and social cost in terms of its potential impacts.

Motivated by its significance in practice, cost-sensitive learning has established itself as an active topic in data mining and machine learning areas [14], [15], [16], [17], [18], [19], [20], [21], [22], [23], [24]. Common solutions to cost-sensitive problems includes sampling [19], [25], decision tree modelling [17], [26], boosting [20], [21], and SVM adaptations [15], [22], [27]. However, all these methods are only dedicated to generic data sets with feature-vector representation, whereas graphs do not have features immediately available and only contain nodes and their dependency structure information. Indeed, simply enumerating subgraph structures as features is clearly a suboptimal solution for cost-sensitive learning, because substructure space is exponentially large w.r.t. the size of the graph and may be infinite. We need a good strategy to find high quality features to help avoid misclassifications on positive classes.

Recently, an igBoost [11] algorithm has been proposed to handle imbalanced graph data sets. The igBoost approach, extended from a standard cost-insensitive graph classification algorithm gBoost [5], assigns proper weight values to different classes by taking data imbalance into consideration, so the algorithm is potentially useful to tackle the cost-sensitive learning for graphs. However, the loss function defined in igBoost is not cost-sensitive but only aims to minimize the misclassification errors. As a result, if the training data are separable [22], the algorithm will have limited power to enforce a cost-sensitivity learning because it only tries to separate training samples without using costs associated to different classes to tune the decisions for minimum costs. From a statistical point of view, the minimum risk could be achieved by following Bayes decision rules to predict graph samples. The objective functions in [11] is non-optimal because it simply employs some heuristic schemes, rather than implements the Bayes decision rules to minimize the conditional risk for cost-sensitive setting. To summarize, the current boosting style algorithms are not targeting cost-sensitive learning problems for graph data.

1.2 Fast Training for Large Scale Graphs

Another deficiency of existing graph classification algorithms is that they are only designed for small size graph data sets and are inefficient to scale up to large size graph data sets. Taking existing boosting-based graph classification algorithms [5], [11] as examples, a boosting algorithm iteratively selects the most discriminative subgraphs from the graph data set and then solves a linear programming problem for graph classification. In practice, although one may use an appropriate support value in the first step to find subgraphs, by using subgraph mining based algorithm such as gSpan [28], solving linear programming in each iteration of the second step is a very time-consuming process, which prevents the algorithms from scaling up to large graph data sets.

In Fig. 1, we report the runtime of boosting-based graph classification algorithms with respect to different numbers of training graphs. For a small number of graphs, e.g. 100 to 1,000, both gBoost [5] and igBoost [11] are relatively efficient (requiring 5 to 300 seconds for training). However, when the number of training graphs is considerably large (25,000 graphs or more), the training time for both gBoost and igBoost increase dramatically (about 50,000 seconds), and requires over 13 hours to complete the training task! As big data applications [29] are becoming increasingly popular for different domains and result in graph data sets with large volumes, finding effective boosting algorithms for large scale graph data sets is highly desired.

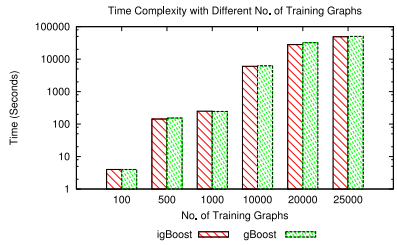

Fig. 1. Training time *w.r.t.* different number of graphs on NCI-1 data set for gBoost [5] and igBoost algorithm [11]. Runtime of existing graph classification algorithms exponentially grows *w.r.t.* the increase of the training set size.

If we consider both cost-sensitive learning and fast graph classification as a whole, the following issues should be taken into consideration to ensure the efficiency and the effectiveness of the algorithm:

1) *Cost-sensitive subgraph selection.* In a cost-sensitive setting, we are given a cost matrix representing misclassification costs. To ensure minimum costs for graph classification, we should take cost of individual samples into consideration to cost-sensitively select discriminative subgraphs. A cost-sensitive subgraph exploration process is, therefore, essential, but has not been addressed by existing research.
2) *Model learning with cost-sensitivity.* For existing boosting-based graph classification algorithms, they have non-optimal loss function, and therefore have limited capability to handle cost-sensitive problems. Alternatively, we should employ a proper loss function, which not only implements the cost-sensitive Bayes decision rule, but also approximates the Bayes risk. By doing this, the induced model will have maximum power for cost-sensitive graph classification.
3) *Fast training on large graph data sets.* Boosting algorithms are difficult to scale to large graphs because the optimization procedures involved in each iteration needs to resolve a large scale linear programming problem, which is typically time-consuming. We need new optimization techniques to enable fast training on large scale graph data sets.

Motivated by the above observations, we report in this paper, CogBoost, a fast cost-sensitive learning algorithm for graph classification. Instead of simply assigning weights to different classes, CogBoost employs a Bayes decision rule based loss function to guarantee minimum risk for prediction. Meanwhile, in order to identify discriminative subgraphs for cost-sensitive graph classification, CogBoost progressively selects the most informative subgraphs based on current learned model, and the newly selected subgraph is added to current feature set to refine the classifier model.

To enable fast training on large graph data sets, an advanced optimization technique, *cutting plane algorithm*, is derived to solve linear programming in an efficient way. Experiments on real-word large graph data sets demonstrate CogBoost's performance.

The remainder of the paper is structured as follow. Section 2 reviews related work. The problem definition and overall framework are discussed in Section 3. Section 4 reports our CogBoost algorithm for cost-sensitive graph classification. The cutting plane algorithm for fast training is reported in Section 5, followed by the time complexity analysis in Section 6. The experiments are reported in Section 7, and we conclude the paper in Section 8.

2 Related Work

Our work is closely related to graph classification and cost-sensitive learning.

Graph classification. Learning and classifying graph data have drawn much attention in recent years. Because graphs involve node-edge structures whereas most existing classification methods use instance-feature representation model, the major challenge of graph classification is to transfer graphs into proper format for learning methods to train classifiers. Existing methods in the area mainly fall into two categories: (1) global similarity-based approaches [1], [4]; and (2) local subgraph based approaches [5], [7], [8]. For global similarity-based methods, graph kernels and graph embedding are used to calculate the distance between a pair of graphs, and the distance matrix can be fed into a learning algorithm, such as *k*-NN and SVM, for graph classification.

For subgraph feature based methods, the major goal is to identify significant subgraphs which can be used as signature for different classes [5], [7], [9], [30], [31]. By using subgraph features selected from the graph set, one can easily transfer graphs into a vector space so existing machine learning methods can be applied for classification [7], [30]. In [32], the authors propose a structure feature selection method to consider subgraph structural information for classification. Jin et al. [33] proposes to extract subgraph patterns and use their co-occurrence to build classifier for graph classification. Other method [34] regards subgraph selection as combinatorial optimization problem and uses heuristic rules, in combination with frequent subgraph mining algorithm such as gSpan [28], to find subgraph features.

After obtaining subgraph features, one can also employ Boosting algorithms for graph classification [5], [9], [35], [36]. In [9], the authors proposed to boost the subgraph decision stumps from frequent subgraphs, which means they first need to provide a minimum support to mine a set of frequent subgraphs and then utilize the function space knowledge for boosting. In contrast, gBoost [35] and its variants [5], [11] do not require a minimum support, the pruning search space relies on some carefully derived rules. gBoost adopts an Adaboost procedure in [35], and it is formalized as a mathematical margin maximization problem in its latter variant [5]. The mathematical LP-boost style of algorithm in [5] demonstrated that the method is effective and converges very fast.

The above algorithms, regardless of whether they transfer graphs into a feature space or boosting directly from the weak subgraph decision stumps, are designed for *cost-insensitivity* scenarios, i.e., they assume that all misclassifications are subject to the same cost/risk. Recently, an igBoost [11] algorithm has been proposed to deal with imbalanced graph data sets (we extend igBoost to handle dynamic data streams in [37]). igBoost assigns different weight values to different classes in order to combat class imbalance issue, which can deal with cost-sensitive problem to some extent. However, the loss function of igBoost is rather heuristic and cannot guarantee the minimization of the misclassification costs. In other words, igBoost is a sub-optimal solution to deal with cost-sensitive graph classification.

Cost-sensitive learning. Cost-sensitive learning has been extensively studied in the last decade. Existing approaches mainly fall into the following four categories: (1) Sampling methods [19]; (2) Decision tree approaches [17], [26]; (3) Boosting algorithms [20], [21], [38]; and (4) SVM adaptation [15], [22].

Sampling approaches [19] aim to re-weight training samples proportional to their cost values, by over-sampling or cost-proportionate rejection sampling. The main goal is to change sample distributions so that any classifier can be directly used to handle cost-sensitive problems. Decision tree modelling approaches [17], [26] incorporate costs into the tree construction process, so that misclassification cost at the leaves is minimized. Boosting algorithms [20], [21], [38], such as AdaCost [38], use the misclassification costs to update the training distributions on successive boosting rounds, which has been proved to be effective to reduce the upper bound of cumulative misclassification cost of the training set. SVM adaptation [15], [22] represents a set of approaches using SVM adaptation for cost-sensitive learning. They either shift the decision boundaries by adjusting the threshold of standard SVMs [27] or by introducing different penalty factors C_1 and C_{-1} for positive and negative SVM slack variables during training [15]. Recently a CS-SVM algorithm [22] is proposed by utilizing an optimal hinge loss function, and has demonstrated better performance than previous approaches [15], [27].

In summary, the scope of all existing cost-sensitive methods is limited to data in vector format. In this paper, we consider unique challenges of graph data and propose a novel algorithm for cost-sensitive graph classification.

3 Problem Definition and Overall Framework

Definition 1 (Connected Graph). *A graph is denoted as $G = (\mathcal{V}, E, \mathcal{L})$, where $V = \{v_1, \ldots, v_{n_v}\}$ is a set of vertices, $E \subseteq \mathcal{V} \times \mathcal{V}$ is a set of edges, and $\mathcal{L}$ is a labeling function assigning labels to a node or an edge. A connected graph is a graph such that there is a path between any pair of vertices.*

In this paper, each labeled graph G_i is a connected graph with a class label $y_i \in \mathcal{Y} = \{-1, +1\}$, i.e., we consider binary classification. If $y_i = +1$, G_i is a positive graph, or negative otherwise.

Definition 2 (Subgraph). *Given two connected 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 $\widehat{f}$: $\mathcal{V}' \rightarrow \mathcal{V}$, such that $\forall (a, b) \in E'$, we have $(\widehat{f}(a), \widehat{f}(b)) \in E$, $\mathcal{L}'(a) = \mathcal{L}(\widehat{f}(a))$,*

TABLE 1
Important Notations Used in the Paper

Symbols	Definition
$T = \{G_i, y_i\}_{i=1,\ldots,l}$	Training graphs with size l
l_+, l_-	Number of positive and negative graphs
$\boldsymbol{x}_i$	Vector representation of graph G_i
$\mathcal{F} = \{g_1, \ldots, g_m\}$	The full set of subgraphs
$\mathcal{S}$	Selected discriminative subgraphs
$\boldsymbol{w} = \{w_k\}_{k=1,\ldots,m}$	Weight vectors for all subgraphs
$f(G_i), f(\boldsymbol{x}_i)$	Classifier prediction on graph G_i
$L(f(G_i), y_i)$	A loss function
C_1, C_{-1}	Cost of positive and negative graphs, respectively
$\boldsymbol{\xi} = \{\xi_i\}_{i=1,\ldots,l}$	Vector, slack variables for cost-sensitive learning
ξ	Slack variable (a scalar) for cutting plane algorithm
$\boldsymbol{\mu} = \{\mu_i\}_{i=1,\ldots,l}$	Weight vectors of training graphs
C, γ	Parameters for cost-sensitive learning
T_{max}	Maximum number of iterations
min_sup	Mininum support for subgraph mining
ϵ	Cutting-plane termination threshold

$\mathcal{L}'(b) = \mathcal{L}(\widehat{f}(b))$, $\mathcal{L}'(a,b) = \mathcal{L}(\widehat{f}(a), \widehat{f}(b))$. *If g_i is a subgraph of G ($g_i \subseteq G$), G is a supergraph of g_i ($G \supseteq g_i$).*

Classifier model for graphs. A classifier model $f(\cdot)$, which is learned from a set of training graphs $T = \{(G_1, y_1), \ldots, (G_l, y_l)\}$, is a function to map a connected graph G_i from graph space $\mathcal{G}$ ($G_i \in \mathcal{G}$) to the label space $\mathcal{Y} = \{+1, -1\}$. For cost-sensitive learning, the classifier $f(\cdot)$ is required to minimize the expected misclassification cost/risk $R = E_{G_i,y_i}[L(f(G_i), y_i)]$, where $L(f(G_i), y_i)$ is a non-negative loss function with respect to the misclassification cost. A typical loss function is defined as follows:

$$L(f(G_i), y_i) = \begin{cases} 0 & : \quad f(G_i) = y_i \\ C_1 & : \quad f(G_i) = -1, y_i = 1 \\ C_{-1} & : \quad f(G_i) = 1, y_i = -1. \end{cases} \tag{1}$$

The loss function in Eq. (1) is extended from the standard 0-1 loss function, i.e., $L(f(\cdot), y) = I(f(\cdot) \neq y)$, where $I(\cdot)$ is an indicator function, $I(a) = 1$ if a holds, or $I(a) = 0$ otherwise. In Eq. (1), when $C_1 = C_{-1} = 1$, the function $L(f(G_i), y_i))$ is cost-insensitive and degenerates to the standard 0-1 loss. For cost-sensitive learning, a false negative $(f(G_i) = -1, y_i = 1)$ prediction usually incurs a larger cost than a false positive $(f(G_i) = 1, y_i = -1)$ prediction, i.e., $C_1 > C_{-1}$.

Given a set of training graphs $T = \{(G_1, y_1), \ldots, (G_l, y_l)\}$, and C_1 and C_{-1} for the cost of misclassification, cost-sensitive graph classification *aims* to build an optimal classification model $f(\cdot)$ from T to minimize the expected misclassification cost (also known as loss or risk) $R = E_{G_i,y_i}[L(f(G_i), y_i)]$. Some important notations used in the paper are listed in Table 1.

3.1 Overall Framework

In this paper, we propose a boosting framework for cost-sensitive graph classification. Our framework (Fig. 2) consists of three major steps.

1) *Optimal subgraph exploration.* One optimal subgraph is selected at each step, and the cost-sensitive discriminative subgraph exploration is guided by the model learnt from the previous step. The newly extracted subgraph is added to the most discriminative set $\mathcal{S}$ to enhance the learning in the next step.
2) *Risk minimization and fast training.* A linear program is solved to achieve minimum risk based on current selected subgraphs. To enable fast training on large scale graphs, a novel cutting plane algorithm is employed.
3) *Updating graph weights for new iteration.* After the linear program is solved, the weight values for training graphs are updated and the algorithm continues to next iteration until the whole algorithm converges.

4 Cost-Sensitive Learning for Graph Data

For graph classification, boosting [5], [11] has been previously used to identify subgraphs from the training graphs as features. After that, each subgraph is regarded as a decision stump (weak classifier) to build a boosting process:

$$\hbar_{g_k}(G_i; \pi_k) = \pi_k(2I(g_k \subseteq G_i) - 1); \tag{2}$$

where $\pi_k \in \mathcal{Y} = \{-1, +1\}$ is a parameter controlling the label of the classifier. In this paper, a weak classifier is written as $\hbar_{g_k}(G_i)$ for short.

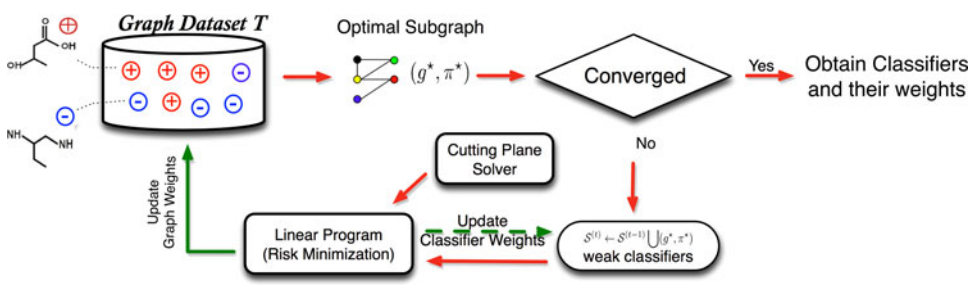

Fig. 2. The proposed fast cost-sensitive boosting for graph classification framework. In each iteration, CogBoost selects an optimal subgraph feature $(g^\star, \pi^\star)$ based on current learned model and weights of training graphs. Then $(g^\star, \pi^\star)$ is added to the current selected set S. Afterwards, CogBoost solves a linear programming problem to achieve cost/risk minimization. To enable fast training on large scale graph data sets, a cutting plane solver is used to improve the algorithm efficiency. After solving the linear program, two sets of weights (green arrows) are updated: (1) weights for training graphs, and (2) weights for weak learners (subgraphs). The feature selection and risk minimization procedures continue until CogBoost converges or reaches the predefined number of iterations.

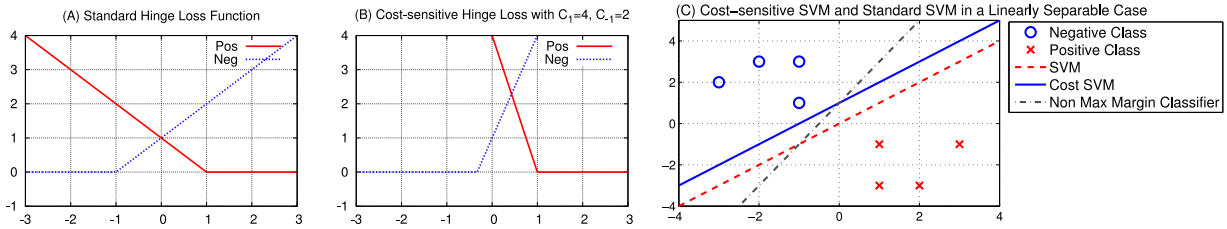

Fig. 3. Different loss functions and formulations with respect to support vector machines (SVMs): (A) Standard Hinge Loss, (B) Cost-sensitive Hinge Loss with $C_1 = 4$ and $C_{-1} = 2$, and (C) Different SVM formulations with standard hinge loss and cost-sensitive hinge loss (**cf**.[22]).

Let $\mathcal{F} = \{g_1, \ldots, g_m\}$ 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 $\boldsymbol{x}_i = \{\hbar_{g_1}(G_i), \ldots, \hbar_{g_m}(G_i)\}$, with $\boldsymbol{x}_i^k = \hbar_{g_k}(G_i)$. In the following section, we use G_i and $\boldsymbol{x}_i$ interchangeably as they are both refereed to the same graph.

The prediction rule for a graph G_i is a linear combination of weak classifiers:

$$f(\boldsymbol{x}_i) = \boldsymbol{w}^T \boldsymbol{x_i} = \sum_{(g_k, \pi_k) \in \mathcal{F} \times \mathcal{Y}} w_k \hbar_{g_k}(G_i), \tag{3}$$

where $\boldsymbol{w} = \{w_k\}_{k=1,\ldots,m}$ is the weight vector for all weak classifiers. The predicted class label of $\boldsymbol{x}_i$ is +1 (positive) if $f(\boldsymbol{x}_i) > 0$ or -1 otherwise.

Similar to SVM, gBoost [5] aims to achieve minimum loss *w.r.t.* a standard hinge loss function $L(f, y) = \lfloor 1 - yf \rfloor_+$, where $\lfloor x \rfloor_+ = \max(x, 0)$. igBoost [11] extends gBoost by assigning larger weight values to graphs in positive classes. Both gBoost and igBoost are not optimal when dealing with cost-sensitive cases, because their loss functions do not follow the Bayes decision rules to minimize the expected risk/loss. In this section, we will first present an optimal hinge loss function, and then formalize our algorithm as a boosting paradigm.

4.1 Optimal Cost-Sensitive Loss Function

A graph classifier $f(\cdot)$ maps a graph G_i to a class label $y_i \in \{-1, 1\}$. Assume graphs and class labels are drawn from probability distribution $P_{\mathcal{G}}(G_i)$ and $P_{\mathcal{Y}}(y_i)$, respectively. Given a non-negative loss function $L(f(G_i), y_i)$, a classifier $f(G_i)$ is optimal if it minimizes the loss/risk $R = E_{G_i, y_i}[L(f(G_i), y_i)]$. Let $\eta = P_{\mathcal{Y}|\mathcal{G}}(1|G_i)$ be the probability of G_i being 1, from a Bayes decision rule point of view, this is equivalent to minimizing the conditional risk,

$$\begin{aligned} E_{\mathcal{Y}|\mathcal{G}}(L(f(G_i), y_i)|G = G_i) = &\eta L(f(G_i), 1) \\ &+ (1 - \eta) L(f(G_i), -1). \end{aligned} \tag{4}$$

The loss function in Eq. (1) is a Bayes consistent loss function [39], i.e., it implements the Bayes decision rule to achieve minimum conditional risk (Eq. (4)). This suggests that ideally Eq. (1) can be used to design some cost-sensitive algorithms for minimizing conditional risk. However, Eq. (1) is extended by a 0-1 loss function. Minimizing the 0-1 loss is computationally expensive because it is not convex. State of the art algorithms usually use surrogate loss functions to approximate the 0-1 loss (e.g., SVM and gBoost [5] employ hinge loss). The hinge loss induced SVM algorithms enforce maximum margins between the support vectors and the hyperplanes, in order to achieve good classification performance.

A recent work on SVM [22] theoretically suggests that the standard hinge loss can be extended to be cost-sensitive, by setting the loss function $L(f(G_i), y_i)$ as follows:

$$L(f(G_i), y_i) = \begin{cases} \lfloor C_1 - C_1 \cdot f(G_i) \rfloor_+ & : \quad y_i = 1 \\ \lfloor 1 + (2C_{-1} - 1) \cdot f(G_i) \rfloor_+ & : \quad y_i = -1. \end{cases} \tag{5}$$

It is proved in [22] that the new hinge loss function Eq. (5) implements the Bayes decision rule. Additionally, Eq. (5) also enjoys the merit of maximum margin principal for classification. The standard hinge loss and its cost-sensitive hinge loss are illustrated in Fig. 3. They have different explanations with respect to the loss and the margins (distance from support vectors to the hyperplane). Specifically, for standard hinge loss (Fig. 3A), the positive and negative classes both have equal margins (unit margins); whereas for cost-sensitive hinge loss (Fig. 3B), the negative class has a much smaller margin when the positive class still have a unit margin. As shown in Fig. 3C, the margins for positive and negative classes are uneven when cost-sensitive hinge loss function Eq. (5) is utilized in a SVM formulation.

Note that the loss function employed in igBoost [11] is heuristically adapted from standard hinge loss, which does not necessarily follow the Bayes decision rule. In other words, it is a sub-optimal loss function for cost-sensitive learning. In the following section, we will use the cost-sensitive hinge loss function in Eq. (5), and re-formulate it into a linear program boosting framework.

4.2 Cost-Sensitive Formulation for Graphs

Motivated by the optimal loss function in Eq. (5), we formalize our learning task as the following regularized risk minimization problem:

$$\begin{aligned} \min_{\boldsymbol{w}} \quad & \|\boldsymbol{w}\| + \frac{C}{l} \left\{ \sum_{\{i|y_i=1\}} L(f(\boldsymbol{x}_i), 1) + \sum_{\{j|y_j=-1\}} L(f(\boldsymbol{x}_j), -1) \right\} \\ s.t. \quad & \boldsymbol{w} \succeq 0. \end{aligned} \tag{6}$$

In Eq. (6), we enforce the weight for each subgraph to be positive, i.e., $\boldsymbol{w} \succeq 0$. We also impose 1-norm regularization on $\boldsymbol{w}$ (i.e., $\|\boldsymbol{w}\|$) to favor sparse solutions with many variables being exactly 0. This strategy is similar to the problem of LASSO for variable shrinking [40]. By using i and j as the index of positive and negative training graphs, respectively, and denoting C a parameter to trade-off the regularization term and loss term, the objective function in Eq. (6) can be reformulated as follows:

$$\begin{aligned} \min_{\boldsymbol{w},\boldsymbol{\xi}} \quad & \|\boldsymbol{w}\| + \frac{C}{l}\left\{C_1 \sum_{\{i|y_i=1\}} \xi_i + \gamma \sum_{\{j|y_j=-1\}} \xi_j\right\} \\ s.\,t. \quad & f(\boldsymbol{x}_i) \geq 1 - \xi_i, y_i = 1 \\ & f(\boldsymbol{x}_j) \leq -\frac{1}{\gamma} + \xi_j, y_j = -1 \\ & f(\boldsymbol{x}_i) = \sum_{k=1}^{m} w_k \cdot \hbar_{g_k}(G_i) \\ & \boldsymbol{w} \succeq 0, \boldsymbol{\xi} \succeq 0, \gamma = 2C_{-1} - 1. \end{aligned} \tag{7}$$

In Eq. (7), ξ_i and ξ_j are slack variables concerning the loss of misclassifying a positive and a negative graph, respectively. In this case, the cost-sensitivity is controlled by C_1 and γ, which impose a smaller margin on negative examples than positive examples (In example shown in Figs. 3B and 3C, we have $C_1 = 4$ and $\gamma = 2C_{-1} - 1 = 3$, the margin for negative example is $\frac{1}{\gamma} = \frac{1}{3}$). As suggested in [39], we can set γ as a parameter subject to $1 \leq \gamma \leq C_1$ instead of a fixed value $(2C_{-1} - 1)$ to achieve better classification.

Solving objective function in Eq. (7) requires a complete set of subgraph features (i.e., represent G_i as $\boldsymbol{x}_i = \{\hbar_{g_1}(G_i), \ldots, \hbar_{g_m}(G_i)\}$), which are unavailable unless we enumerate the whole subgraph space in advance. In practice, this is likely impossible because the whole subgraph set is very large or even infinite. In the following section, we will transfer this formulation to its Lagrange dual problem and use a boosting algorithm to solve it in an iterative way.

4.3 Boosting for Cost-Sensitive Learning on Graphs

The Lagrange dual of a problem usually provides additional insights to the original (primal) problem. The dual problem of Eq.(7) is[1]

$$\begin{aligned} \max_{\boldsymbol{\mu}} \quad & \sum_{i=1}^{l_+} \mu_i + \frac{1}{\gamma} \sum_{j=1}^{l_-} \mu_j \\ s.t. \quad & \sum_{i=1}^{l_+} \mu_i \hbar_{g_k}(G_i) - \sum_{j=1}^{l_-} \mu_j \hbar_{g_k}(G_j) \leq 1, \forall g_k \in \mathcal{F} \\ & 0 \leq \mu_i \leq \frac{CC_{-1}}{l}, i = 1, \ldots, l^+ \\ & 0 \leq \mu_j \leq \frac{\gamma C}{l}, j = 1, \ldots, l_- \end{aligned} \tag{8}$$

where l_+ and l_- indicate the number of graphs in positive and negative sets ($l = l_+ + l_-$), respectively. While solving the primal problem in Eq. (7) returns a vector $\boldsymbol{w}$ indicating the weights of each subgraphs, the dual problem in Eq. (8) will produce a vector $\boldsymbol{\mu} = \{\mu_i\}_{i=1,\ldots,l}$. Nevertheless, Eq. (7) and Eq. (8) will generate the same objective values.

Insights of dual problem. (1) The solution $\{\mu_i\}_{i=1,\ldots,l}$ can be interpreted as the weight values of graphs in order to achieve minimum loss. (2) Each constraint $\sum_{i=1}^{l_+} \mu_i h_{g_k}(G_i) - \sum_{j=1}^{l_-} \mu_j h_{g_k}(G_j) \leq 1$ in Eq. (8) indicates a subgraph pattern g_k over the whole training graphs. It provides a natural metric to assess the cost-sensitive discriminative power of a subgraph.

Definition 3 (Cost-sensitive Discriminative Score). *For a subgraph decision stump $\hbar_{g_k}(G_i)$, its cost-sensitive discriminative score over all training graphs is:*

$$\Theta(g_k, \pi_k) = \sum_{i=1}^{l_+} \mu_i \hbar_{g_k}(G_i) - \sum_{j=1}^{l_-} \mu_j \hbar_{g_k}(G_j). \tag{9}$$

The first constraint of Eq. (8) requires discriminative scores for all subgraphs ≤ 1, which latter will serve as an termination condition of our iterative algorithm.

Linear program boosting framework. Because we do not have a predefined feature set $\mathcal{F}$ in advance, we cannot solve Eq. (7) or Eq. (8). Therefore, we propose to use *column generation (CG)* techniques [41] to solve the objective function (Eq. (7)). The idea of CG is to begin with an empty feature set $\mathcal{S}$, and iteratively selects and adds one feature/column to $\mathcal{S}$ which violates the constraint in the dual problem (Eq. (8)) mostly. After $\mathcal{S}$ is updated, CG re-solves the primal problem Eq. (7). This procedure continues until no more subgraph violates the constraint in (Eq. (8)).

Algorithm 1. CogBoost Algorithm for Graph Classification

Require:
$T_l = \{(G_1, y_1), \ldots, (G_l, y_l)\}$: Training Graphs;
T_{max}: Maximum number of iteration;
Ensure:
$f(\boldsymbol{x}_i) = \sum_{(g_k,\pi_k)\in S} w_k^{(t-1)} \hbar_{g_k}(G_i)$: Classifier;
1: $\mathcal{S} \leftarrow \emptyset$;
2: $t \leftarrow 0$;
3: **while** true **do**
4: Obtain the most discriminative decision stump $(g^\star, \pi^\star)$; //Algorithm 2;
5: **if** $\Theta(g^\star, \pi^\star) \leq 1 + \Delta$ **or** $t = T_{max}$ **then**
6: **break;**
7: $\mathcal{S} \leftarrow \mathcal{S} \bigcup (g^\star, \pi^\star)$;
8: Solve Eq. (7) based on $\mathcal{S}$ to get $\boldsymbol{w}^{(t)}$, and Lagrange multipliers of Eq. (8) $\boldsymbol{\mu}^{(t)}$;
9: $t \leftarrow t + 1$;
10: **return** $f(\boldsymbol{x}_i) = \sum_{(g_k,\pi_k)\in S} w_k^{(t-1)} \hbar_{g_k}(G_i)$;

Our cost-sensitive graph boosting framework is illustrated in Algorithm 1. CogBoost iteratively selects the most discriminative subgraph $(g^\star, \pi^\star)$ at each round (step 4). If the current optimal pattern no longer violates the constraint or it has reached the maximum number of iterations T_{max}, the iteration process stops (steps 5-6). Because in the last few iterations, the optimal value only changes subtlety, we add a small value Δ to relax the stopping condition (typically, we use $\Delta = 0.01$ in our experiments). On step 8, we solve the linear programming problem based on the selected subgraphs to recalculate two sets of weight values: (1) $\boldsymbol{w}^{(t)}$, the weights for subgraph decision stumps in $\mathcal{S}$; and (2) $\boldsymbol{\mu}^{(t)}$, the weights of training graph for optimal subgraph mining in the next round, which can be obtained from the Lagrange multipliers of the primal problem. Once the algorithm converges or the

1. The derivation from the primal problem Eq. (7) to dual problem Eq. (8) is shown in Appendix A.1, which can be found on the Computer Society Digital Library at http://doi.ieeecomputersociety.org/10.1109/TKDE.2014.2391115.

number of maximum iteration is reached, CogBoost returns the final classifier model $f(\boldsymbol{x}_i)$ on step 10.

4.4 Cost-Sensitive Subgraph Exploration

To learn the classification model, we need to find the most discriminative subgraph which considers each training graph's weight in each step (step 4 in Algorithm 1). The subgraph exploration is completely model driven, i.e., we select a subgraph which violates the constraint in Eq. (8) mostly. Based on the definition of discriminative score in Eq. (9), we need to perform a weighted subgraph mining over training graphs.

In CogBoost, we employ a Depth-First-Search (DFS) based algorithm gSpan [28] to enumerate subgraphs. The key idea of gSpan is that each subgraph has a unique DFS Code, defined by its lexicographic order of the discovery time during the search process. Two subgraphs are isomorphism iff they have the same minimum DFS Code. 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 further employ a branch-and-bound scheme to prune the search space of DFS Code tree by utilizing an upper bound of discriminative score [5] for each subgraph pattern.

Algorithm 2. Cost-sensitive Subgraph Exploration

Require:
$T_l = \{(G_1, y_1), \ldots, (G_l, y_l)\}$: Labeled Graphs;
$\boldsymbol{\mu} = \{\mu_1, \ldots, \mu_l\}$: Weights for labeled graph examples;
min_sup: mininum support for subgraph mining;
Ensure:
$(g^\star, \pi^\star)$: The most discriminative subgraph;
1: $\tau = 0$, $(g^\star, \pi^\star) \leftarrow \emptyset$;
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 **or** $sup(g_p) < min\_sup$ **then**
5: **continue**;
6: Compute score $\Theta(g_p, \pi_p)$ for subgraph g_p according Eq. (9);
7: **if** $\Theta(g_p, \pi_p) > \tau$ **then**
8: $(g^\star, \pi^\star) \leftarrow (g_p, \pi_p)$; $\tau \leftarrow \Theta(g_p, \pi_p)$;
9: **if** The upperbound of score $\widehat{\Theta}(g_p) > \tau$ **then**
10: Depth-first search the subtree rooted from node g_p;
11: **return** $(g^\star, \pi^\star)$;

Our subgraph mining algorithm is listed in Algorithm 2. The minimum value τ and optimal subgraph $(g^\star, \pi^\star)$ are initialized on step 1. We prune out duplicated subgraph features or subgraph with low support ($sup(\cdot)$ returns the support of a subgraph) on steps 4-5, and compute the discriminative score $\Theta(g_p, \pi_p)$ for g_p on step 6. If $\Theta(g_p, \pi_p)$ is larger than τ, we update the optimal subgraph on step 8. We use a branch-and-bound pruning rule in [5] to prune the search space on steps 9-10. Finally, the optimal subgraph pattern $(g^\star, \pi^\star)$ is returned on step 11.

5 Fast Training for Large Scale Graphs

For CogBoost algorithm, it needs to iteratively mine an optimal subgraph (step 4 of Algorithm 1) and solve a linear problem (step 8 of Algorithm 1). To enable fast training for large scale graph data sets, we can apply a proper support and some heuristic techniques to step 4, such as reusing the search space during the enumeration of subgraphs rather than re-mining subgraph from scratch, just as [5] does. For step 8, we derive a *cutting plane* algorithm to speed up the training process.

5.1 From l-Slacks to 1-Slack Formulation

Eq. (7) on step 8 of Algorithm 1 has $l = l_+ + l_-$ slack variables ξ_i and ξ_j, inspired by the techniques used in the SVM formulation [42], we propose to solve it efficiently by reducing the number of slack variables as follows,

$$\begin{aligned} \min_{\boldsymbol{w},\xi} \quad & \|\boldsymbol{w}\| + C\xi \\ s.\,t. \quad \forall \boldsymbol{c} \in \{0,1\}^l, \quad & \frac{1}{l}\boldsymbol{w}^T\left\{C_1\sum_{y_i=1} c_i\boldsymbol{x}_i - \gamma\sum_{y_j=-1} c_j\boldsymbol{x}_j\right\} \\ & \geq \frac{1}{l}\left\{C_1\sum_{y_i=1} c_i + \sum_{y_j=1} c_j\right\} - \xi, \\ & \boldsymbol{w} \succeq 0, \xi \geq 0. \end{aligned} \tag{10}$$

The above formulation only has one slack variable ξ, which can be proved to be equal to Eq. (7)[2], with $\xi = \{C_1\sum_{\{i|y_i=1\}}\xi_i + \gamma\sum_{\{j|y_j=-1\}}\xi_j\}/l$. Note that although Eq. (10) has 2^l constraints in total, such a formulation can be solved by cutting plane algorithm in linear time by iteratively selecting a small number of most violated constraints (*Cutting Planes*). This leads to an efficient solution to the optimization, so our algorithm can effectively scale to large data sets.

The dual of l-slack formulation in Eq. (8) provides solution $\boldsymbol{\mu}$ which can interpret the graph weights for subgraph mining in the next iteration. To establish the same relationship between the new objective function Eq. (10) and the graph weights $\boldsymbol{\mu}$, we also refer to its dual problem, which is given as follows[3]:

$$\begin{aligned} \max_{\boldsymbol{\lambda_c}} \quad & \frac{c_1}{l}\sum_{\boldsymbol{c}}\lambda_{\boldsymbol{c}}\sum_i c_i + \frac{1}{l}\sum_{\boldsymbol{c}}\lambda_{\boldsymbol{c}}\sum_i c_j \\ s.t. \quad & \frac{C_1}{l}\sum_{\boldsymbol{c}}\lambda_{\boldsymbol{c}}\sum_i c_i\boldsymbol{x}_i^k - \frac{\gamma}{l}\sum_{\boldsymbol{c}}\lambda_{\boldsymbol{c}}\sum_j c_j\boldsymbol{x}_j^k \leq 1, \forall k \\ & 0 \leq \sum_{\boldsymbol{c}}\lambda_{\boldsymbol{c}} \leq C. \end{aligned} \tag{11}$$

Comparing the dual problems of Eq. (8) and Eq. (11), they are identical if:

$$\mu_{i|y_i=1} = \frac{C_1}{l}\sum_{\boldsymbol{c}}\lambda_{\boldsymbol{c}}c_i;\ \mu_{j|y_j=-1} = \frac{\gamma}{l}\sum_{\boldsymbol{c}}\lambda_{\boldsymbol{c}}c_j. \tag{12}$$

5.2 Cutting-Plane Algorithm for Fast Training

The basic idea of cutting-plane algorithm is similar to the column generation algorithm, or it can be regarded as a row

2. Appendix A.2, available in the online supplemental material, proves the equality of Eq. (7) and Eq. (10).

3. The derivation from Eq. (10) to Eq. (11) is given in Appendix A.3, available in the online supplemental material.

generation algorithm (each constrain in Eq. (11) is a row). Instead of considering all constrains (rows) as a whole, our cutting-plane algorithm considers only the most violated constraint (row) each time. The selected most violated constraints form a working set $\mathcal{W}$, and whole process follows an iterative procedure to solve the problem. By doing this, the linear program can be solve efficiently.

Our detailed cutting plane algorithm is shown in Algorithm 3. Initially, the working set $\mathcal{W}$ is an empty set on step 1. In each iteration, we solve the optimization problem based on current working set $\mathcal{W}$ on step 3 ($\boldsymbol{w} = 0$ and $\xi = 0$ for the first iteration). Steps 4-6 find the most violated constraint, which is determined by the cost-sensitive loss function in Eq. (5). Step 9 adds the current most violated constraint to the working set. The iteration continues until it reaches the convergence (steps 7-8). Also, we add a small constant ϵ (In experiments, we set $\epsilon = 0.01$ as default value) to enable early termination of iterations.

Algorithm 3. Cutting Plane Algorithm for Linear Problem Eq. (7)

Require:
$\{\boldsymbol{x}_1, \ldots, \boldsymbol{x}_l\}$: Training graphs with subgraph representation.
C, C_1, C_{-1}: Parameters for classifier learning.
ϵ: Cutting-plane termination threshold.
Ensure:
$\boldsymbol{w}$: Classifier weights; $\boldsymbol{\mu}$: Graph weights;
1: Initialize $\mathcal{W} \leftarrow \emptyset$;
2: **while** true **do**
3: Obtain primal and dual solutions $\boldsymbol{w}, \xi, \boldsymbol{\lambda}$ by solving

$$\min_{\boldsymbol{w},\boldsymbol{\xi}} \quad \|\boldsymbol{w}\| + C\xi$$

$$s.\ t.\ \forall \boldsymbol{c} \in \mathcal{W}, \boldsymbol{w}^T \frac{1}{l}\left\{ C_1 \sum_{y_i=+1} c_i \boldsymbol{x}_i - \gamma \sum_{y_j=-1} c_j \boldsymbol{x}_i \right\} \geq \frac{1}{l}\left(C_1 \sum_{y_j=1} c_i + \sum_{y_i=-1} c_j \right) + \xi, \qquad \boldsymbol{w} \succeq 0, \xi > 0.$$

4: **for** $i = 1 \cdots l$ **do**
5: applying the following rule the find the most constraint variables on positive graphs ($y_i = 1$)

$$c_i = \begin{cases} 1: & y_i \cdot f(\boldsymbol{x}_i) < 1 \\ 0: & else \end{cases}$$

6: applying the following rule to negative graphs ($y_j = -1$)

$$c_j = \begin{cases} 1: & \gamma \cdot y_i \cdot f(\boldsymbol{x}_i) < 1 \\ 0: & else \end{cases}$$

7: **if** $\frac{1}{l}(C_1 \sum_{y_i=1} c_i + \sum_{y_j=-1} c_j) - \boldsymbol{w}^T \frac{1}{l}(C_1 \sum_{y_i=+1} c_i \boldsymbol{x}_i - \gamma \sum_{y_j=-1} c_j \boldsymbol{x}_i) \leq \xi + \epsilon$ **then**
8: **break;**
9: $\mathcal{W} \leftarrow \mathcal{W} \bigcup \boldsymbol{c}$;
10: update μ_i and μ_j according to Eq. (12);
11: **return** $\boldsymbol{w}$ and $\boldsymbol{\mu}$;

Our cutting plane algorithm can always return an ϵ-tolerance accurate solution (approximate the solution of Eq. (7) very well). It is efficient because each time we solve a linear program in a small working set, so the cutting plan algorithm is independent of the number of sample size. This essentially ensures that our solutions can scale to very large graph data sets.

6 Time Complexity Analysis: Theoretical Aspect and Practice

The time complexity of CogBoost includes two major components: (1) mining a cost-sensitive discriminative subgraphs $\mathcal{O}(P(l))$ (step 4 of Algorithm 1), and (2) solving a linear program problem $\mathcal{O}(Q(l))$ (step 8 of Algorithm 1), where P and Q are functions for mining subgraph and solving LP problem of size l. For subgraph mining, CogBoost employs a gSpan based algorithm (Algorithm 2) for subgraph enumeration in the first iteration ($\mathcal{O}(P(l))$), and reuses the search space [5] of the first iteration ($\mathcal{O}(\overline{P}(l))$). Because re-using search space can significantly reduce the mining time, we have $\mathcal{O}(\overline{P}(l)) \ll \mathcal{O}(P(l))$. Suppose the number of iterations of CogBoost (Algorithm 1) is T_{max}, the total time complexity of CogBoost is:

$$\mathcal{O} = \mathcal{O}(P(l)) + (T_{max} - 1)\mathcal{O}(\overline{P}(l)) + T_{max}\mathcal{O}(Q(l)). \qquad (13)$$

6.1 Time Complexity of Subgraph Mining

Theoretical aspect. Intuitively, because the subgraph space may be infinitely large, the time complexity for subgraph mining is NP-hard, and $O(P(l))$ for subgraph mining is inevitable for graph classification. Therefore all existing subgraph feature selection algorithms for graph classification [5], [7], [34] rely on some upper-bounds to prune the search space. In CogBoost, we incorporate the upper-bound in [5] and the support threshold min_sup to reduce the subgraph space. It is worth noting that CogBoost can still function property even if users do not specify the min_sup value for subgraph mining. If min_sup were not specified, CogBoost will only rely on the upper-bound in [5] to prune the search space.

Practice. In practice, we observe that when the data set is considerably large (e.g., 40,000 chemical compounds or more), setting a support threshold $min\_sup = 5\%$ can significantly speed up the mining progress. However, setting a threshold may incur missing of discriminative subgraphs because some infrequent subgraphs are not checked. Accordingly, we suggest removing the min_sup threshold for small graph data set while setting a proper support for large data sets. The proper support value depends on the domains of applications. For instance, when the average number of nodes and edges of the graph data set are large, a large support (5-10 percent) is preferred. On the other hand, if the average number of nodes and edges are small, a small support (about 1-3 percent) is a good choice. For real-world applications, it is useful to first check the statistics of the graph samples before carrying out the graph classification tasks.

6.2 Time Complexity of LP Solving

Theoretical aspect. To solve the LP problem, Eq. (7) is solvable in polynomial time $O(Q(l)) = \mathcal{O}(l^k)$ with some constant k [43]. In other words, gBoost [5] and igBoost [11] needs

TABLE 2
Graph Data Sets Used in the Experiments

Data sets	#Pos	#Total	#Nodes	#Edges	Descriptions
NCI-1	1,793	37,349	26	28	Lung Cancer
NCI-33	1,467	37,022	26	28	Melanoma
NCI-41	1,350	25,336	27	29	Prostate Cancer
NCI-47	1,735	37,298	26	28	Central Nerve
NCI-81	2,081	37,549	26	28	Colon Cancer
NCI-83	1,959	25,550	27	29	Breast Cancer
NCI-109	1,773	37,518	26	28	Ovarian
NCI-123	2,715	36,903	26	28	Leukemia
NCI-145	1,641	37,041	26	28	Renal Cancer
Twitter	66,458	140,949	4	5	Sentiment

polynomial time for this step. By using cutting plane algorithm, the time complexity would be $O(Q(l)) = \mathcal{O}(sl)$, where s is the number of non-zeros features in the original problem (please refer to [42] for detailed analysis). Therefore, CogBoost can significantly reduce the runtime when the graph sample size l is large. In Section 7, our experiments will soon demonstrate that the improvement of LP problem solving without using cutting plan algorithm is marginal.

Practice. The cutting plane algorithm uses a working set $\mathcal{W}$ (in Algorithm 3), $\mathcal{W}$ overlaps significantly during two consecutive iterations of Algorithm 1. Therefore, in our implementations, we re-use top 200 most violated constrains in $\mathcal{W}$ in the previous iteration, which can significantly improve the algorithm efficiency. In practice, the classifier weights $\boldsymbol{w}$ in two consecutive iterations may be very close to each other, one can also use the warm-start technique (using $\boldsymbol{w}$ in previous iteration as initial value for linear problem solving) to speed up the learning process.

7 Experiments

In this section, we evaluate CogBoost in terms of its average misclassification cost (or average cost) and runtime performance. The *average cost* is calculated by using the total misclassification costs divided by the number of test instances. The lower the average costs, the better the algorithm performance is. The runtime performance is evaluated based on the actual runtime of the algorithm.

7.1 Experimental Settings

Two types of real-life data sets, NCI chemical compounds and Twitter graphs, are used in our experiments. Table 2 summaries the statistics of the two benchmark data sets.

NCI graph data sets are commonly used as the benchmark for graph classification. In our experiments, we download nine NCI data sets from PubChem.[4] Each NCI data set belongs to a bioassay task for anticancer activity prediction, where each chemical compound is represented as a graph, with atoms representing nodes and bonds as edges. A chemical compound is positive if it is active against the corresponding cancer, or negative otherwise.

Table 2 summarizes the NCI graph data used in our experiments. We have removed disconnected graphs and graphs with unexpected atoms (some graphs have atoms represented as '*') in the original graphs. Columns 2-3 show the number of positive graphs and the total number of graphs in each data set, and columns 4-5 indicate the average number of nodes and edges in each data set, respectively.

Because of the inherently short and sparse nature, twitter sentiment analysis (i.e., predicting whether a tweet reflects a positive or a negative feeling) is a difficult task. To build a graph data set, we collect tweets from Sentiment140[5], a Twitter Sentiment classification site, and represent each tweet as a graph by using tweet content, with nodes in each graph denoting the terms and/or smiley symbols (e.g, :-D and :-P) and edges indicating the co-occurrence relationship between two words or symbols in each tweet. To ensure the quality of the graph, we only use tweets containing 20 or more words. In our experiments, we use tweets from April 6 to June 16 to generate 140,949 graphs (in a chronological order). Note that this data set has been used for graph stream classification in our previous study [37]. In this paper, we aggregate all graphs as one data set without considering their temporal order.

Baselines We compare our proposed CogBoost algorithm with the following baseline algorithms.

- *gBoost* [5] is a state-of-the-art boosting method, which has demonstrated good performance for graph classification.
- *igBoost* [11] extends gBoost to handle imbalanced graph data sets. The weight of a minority (positive) graph is assigned a value β times higher than the weight of a majority (negative) graph.
- *Fre+CSVM* first mines a set of frequent subgraphs (with a minimum support 3 percent) from the entire graph data set, and selects the top-K most frequent subgraphs as features. Afterwards, each graph data set is transformed into vector format by checking the existence of selected subgraphs in the original graph data sets. Finally, a cost-sensitive support vector machine algorithm [15], [18] is applied to the transferred vectors.
- *gSemi+CSVM*[6] employs a gSemi [7] algorithm to mine top-K discriminative subgraphs from the entire graph data set, and then transfers the original graph database into vectors. Similar to Fre+CSVM, the cost-sensitive SVM algorithm [18] is used to learn a model from the transferred vectors.

To validate the effectiveness of the cutting plane solver in our CogBoost algorithm for large scale graphs, we implement two variants of CogBoost,

- *CogBoost-a.* This variant discards the cutting plane module and solves the linear program of Eq. (7) directly. In other words, it uses all l slack variables.
- *CogBoost-1.* The CogBoost-1 utilizes the cutting plane module to solve the linear program (Eq. (10)) for large scale graphs, i.e., it has only one (i.e., 1) slack variable each time.

4. http://pubchem.ncbi.nlm.nih.gov

5. http://help.sentiment140.com/home

6. We encounter an out-of-memory error for gSemi+CSVM algorithm on Twitter graph data set, because gSemi algorithm [7] needs to do matrix calculation to select subgraphs. Java fails to create such a large "double" matrix (about 100,000*100,000).

For each graph data set, we randomly split it into two subsets. The training set consists of 70 percent of the graph data set, and the rest is used as the test set. The results reported in the paper are based on the average of five times random train/test split repetitions. Note that for gBoost [5] and igBoost [11], the previous studies only validate their performance using a rather small number of graphs (from several hundreds to several thousands graphs), whereas in our experiments, our training data is much larger (e.g., 140,949 graphs for Twitter data set).

Parameter settings. For fair comparisons, the default misclassification cost for positive graphs is set as $C_1 = 20$ for NCI graphs, which is actually the approximated imbalanced ratio ($\frac{|Neg|}{|Pos|}$) of these graph data sets. For Twitter graphs, a large C_1 will result in that all graphs are classified in one class for all algorithms. To avoid this case, we set the default value $C_1 = 3$. For all experiments, the cost of negative graphs is always set as $C_{-1} = 1$ for all data sets. As suggested in [39], we selected the best parameter γ instead of fixing it to $2C_{-1} - 1$ for CogBoost algorithm. For igBoost, we set $\beta = C_1$, so positive class graphs have a weight value β times higher than negative graphs. The regularization parameter in our algorithm is C, and $D = 1/v$ for gBoost and igBoost. To make them comparable, we vary C from $\{0.1, 1, 10, 100, 1{,}000, 10{,}000\}$, and v from $\{0.01, 0.2, 0.4, 0.6, 0.8, 1.0\}$. These candidate values are set according to the property of each algorithm. min_sup is set to 5 percent for NCI graphs and 0.5 percent for Twitter graphs.

Because both igBoost and gBoost require over 10 hours to complete a classification task, it is impractical to select the best parameters for each algorithm on the whole training graphs on each data set. Therefore, we select the parameters for each algorithm which achieves the minimum misclassification cost over a sample of 5,000 training graphs on each data set. Then we train the classifiers with these selected parameters on the whole training graphs. For Fre+CSVM and gSemi+CSVM, the number of most informative subgrahps K is always equal to T_{max} employed in another boosting algorithm, i.e., we ensure that all algorithms use the same number of features for graph classification.

Unless specified otherwise, other parameters for our algorithm are set as follows: $T_{max} = 50$ and $\epsilon = 0.01$.

All our algorithms are implemented using a Java package MoSS[7] and Matlab toolbox CVX.[8] MoSS provides a framework for frequent subgraph mining, and CVX serves as a module for solving linear programs. JavaBuilder provided by Matlab bridges MoSS and CVX into a united framework. All our experiments are conducted on a cluster node of Red Hat OS with 12 processors (X5690 @3.47 GHz) and 48 GB memory.

7.2 Experimental Results

In this section, we evaluate the effectiveness of CogBoost for cost-sensitive learning and fast cutting-plane training in terms of average cost and runtime performance. The experimental results for NCI and Twitter graphs under default parameter settings are illustrated in Fig. 4.

7. http://www.borgelt.net/moss.html
8. http://cvxr.com/cvx/

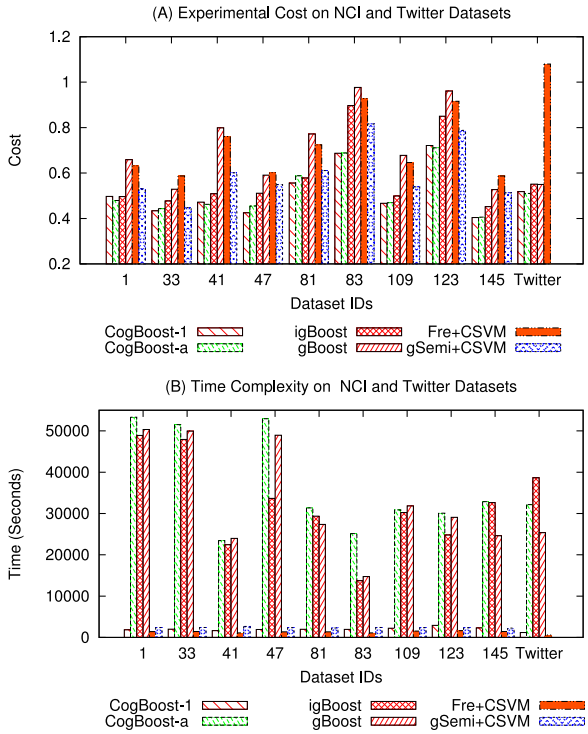

Fig. 4. Experimental results. (A) Average cost, (B) Time complexity.

Average cost. For average cost, Fig. 4A demonstrates that gBoost has the worst performance on five out of 10 data sets (i.e., the largest average cost). This is mainly because gBoost is a cost-insensitive algorithm, which considers that all training graphs are equally important in terms of their costs. As a result, gBoost fails to leverage the costs of graph samples to discover subgraph features mostly discriminative for differentiating graphs in the positive class, leading to deteriorated classification performance.

For igBoost, Fre+CSVM, and gSemi+CSVM, all of them have a mechanism to assign weight values to different classes. Fig. 4A shows that igBoost outperforms Fre+CSVM and gSemi+CSVM, which is mainly attributed to igBoost's integration of discriminative subgraph selection and classifier learning for graph classification. For Fre+CSVM and gSemi+CSVM, they decompose subgraph selection and classifier learning into two separated steps, without integrating them to gain mutual benefits, i.e., the subgraphs selected by frequency and gSemi score [7] may not be a good feature set for SVM learning. As a result, Fre+CSVM and gSemi+CSVM are inferior to igBoost. This is, in fact, consistent with previous studies [5], which confirmed that gBoost outperforms a frequent subgraph based algorithm (mine frequent subgraphs as features and then apply SVMs).

The experimental results in Fig. 4A show that CogBoost outperforms igBoost. This is because the loss function in igBoost is not a cost-sensitive, but heuristically adapted from the hinge loss function (i.e., simply assigning different weights to different classes). Therefore, it does not necessarily implement the Bayes decision rule and cannot guarantee minimum conditional risk.

In contrast, CogBoost-1 and CogBoost-a adopt an optimal cost-sensitive loss function which implements the Bayes

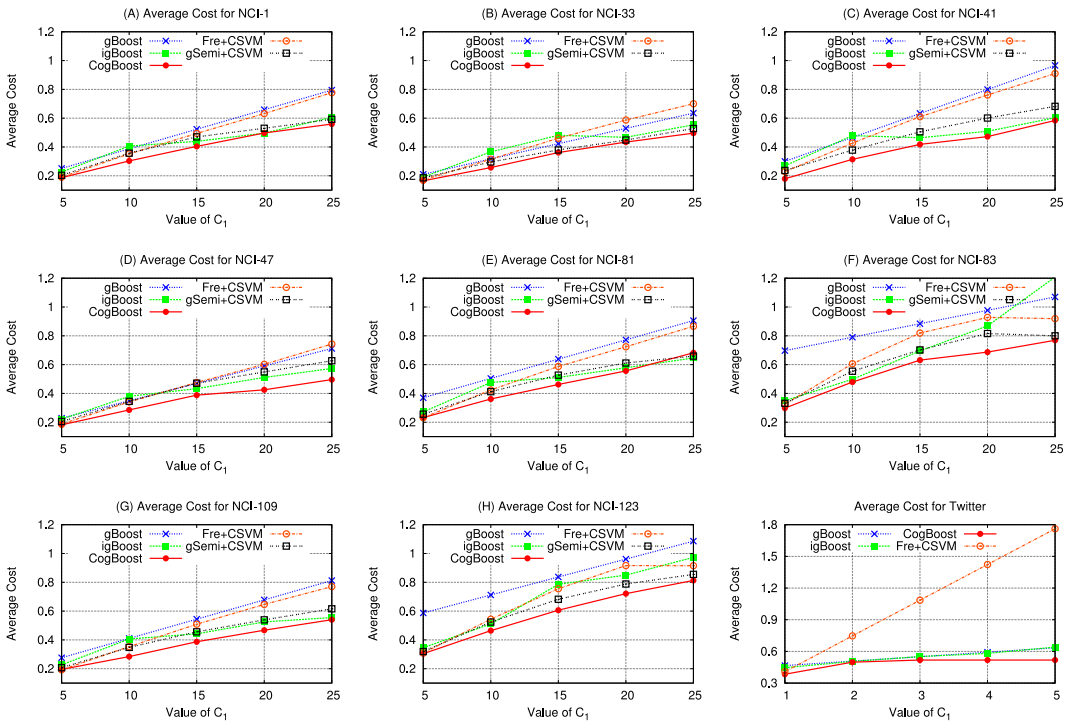

Fig. 5. Average Cost with respect to different C_1 value.

decision rule to achieve minimum cost. Evidently, both CogBoost-1 and CogBoost-a outperform gBoost over all graph data sets with significant performance gain, and outperforms igBoost for most graph data sets.

Runtime performance. The algorithm runtime in Fig. 4B shows that gBoost, igBoost, and CogBoost-a all require an order of magnitude more time over CogBoost-1, Fre+CSVM, and gSemi+CSVM. For instance, CogBoost-1 only needs about 1,846 seconds on NCI-1 data set whereas all other boosting algorithms take about over 50,000 seconds to complete the task. Overall, CogBoost-1 is 25 times faster than all other boosting algorithms. This result validates that reformulating our problem from Eq. (7) to a new problem (Eq. (10)) and using cutting plane algorithm to solve it can efficiently speed up the problem solving.

Note that Fre+CSVM and gSemi+CSVM have a little less runtime than CogBoost-1, this is because they only solve the SVM formulation (quadratic program) once, while our algorithm iteratively solves a linear program in each iteration.

Comparing the runtime of NCI and Twitter data sets, we found that although twitter data set is significant larger than NCI, the time consumption for NCI and Twitter does not differ much. This is because the average number of nodes and edges for twitter data set is much smaller than NCI, making it much efficient for subgraph mining for all algorithms.

Runtime consumption details for boosting algorithms. To better understand why CogBoost is more efficient than its peers, we investigate detailed runtime consumption in each step for boosting algorithms. These boosting methods all consist of two key steps in each iteration: i.e., optimal subgraph mining and linear problem solving. Accordingly, we report the algorithm runtime in each iteration in Fig. 6, and report average time consumption in Table 3.

Table 3 and Fig. 6 show that, on average, subgraph mining can be done in less than 20 seconds for all algorithms. At the first iteration, the subgraph mining step requires a significant amount of runtime. This is because gSpan needs to generate the search tree until the pruning condition is satisfied. Creating a new node is time consuming, because the list of embeddings is updated, and the minimality of the DFS code has to be checked (See [5] for more details). In latter iterations, the time consumption for this process can be reduced greatly because the searching space is reused. The node creation is necessary only if it were not created in

TABLE 3
Average Time Consumption in Each Iteration (Seconds)

	gBoost	igBoost	CogBoost-a	CogBoost-l
Subgraph Mining	8.24	18.24	19.39	16.33
LP Optimization	993.42	954.66	1046.12	21.58

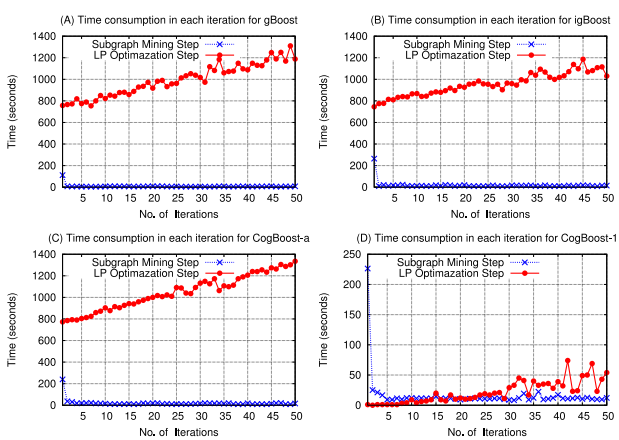

Fig. 6. Runtime performance in each iteration. Runtime consumption for (A) gBoost, (B) igBoost, (C) CogBoost-a, and (D) CogBoost-1.

previous iterations. As a result, we can observe that the algorithm is more efficient in the latter iterations.

As for the LP optimization steps, gBoost, igBoost, and CogBoost-a all consume much more time than CogBoost-1. This is because they all need to solve a linear problem (similar to Eq. (7) for gBoost and igBoost) with l slack variables $\xi_{i|i=1,\ldots,l}$ in each iteration (l is the total number of graph examples). When l is large, it will require a very large amount of time to solve the linear problem. In contrast, CogBoost-1 solves the linear problem (Eq. (10)) with only one single slack variable ξ by using cutting plane algorithms (Algorithm 3). This new formulation can greatly reduce the time required for linear problem solving.

The results in Table 3 and Fig. 6 show that CogBoost-1 only needs about 21.58 seconds for one iteration whereas all other algorithms require about 1,000 seconds to complete this step. Because LP optimization step is the most computationally intensive step for boosting algorithms, CogBoost-1 is much faster than all existing boosting algorithms for graph classification.

Comparison of CogBoost-1 and CogBoost-a. The results in Fig. 4 show that CogBoost-1 can always achieve similar (or very close) classification performance as CogBoost-a. This is because CogBoost-1 can always return an ϵ-tolerance solution to CogBoost-a in each iteration. This, in fact, empirically proves the correctness of our CogBoost-1 formulation. Because CogBoost-1 can achieve accurate solutions to CogBoost-a but with much less runtime consumption than CogBoost-a, in the following experiments, we will report CogBoost-1 (termed as *CogBoost*) for comparison with other algorithms.

Meanwhile, we will mainly focus on the classification performance for cost-sensitive learning because the time complexity is relatively stable for each graph data set with the same number of training graphs.

7.2.1 *Performance* w.r.t. *Different Cost Values*

In order to study the algorithm performance *w.r.t.* different cost values, we vary the C_1 values from 5 to 25 for NCI graphs and 1 to 5 for Twitter graphs and report the algorithm performance in Fig. 5 where the x-axis in each subfigure shows the C_1 values and the y-axis show the average costs of different methods.

Fig. 5 shows that when increasing the C_1 value, the average costs of all algorithms will increase. This is because the increasing of C_1 value results in a higher misclassification cost of positive graphs. Comparing to gBoost, igBoost achieves less average cost on most graph data sets. This is mainly attributed to the uneven weight assignment scheme for different classes adopted in igBoost, which allows igBoost to deal with the cost-sensitive problem to some extent.

For all data sets, CogBoost achieves minimum average cost with respect to different C_1 values. This is attributed to the optimal hinge loss function employed in CogBoost, which implements the Bayes decision rules and forces CogBoost to favor high cost samples in order to minimize the misclassification costs. This result is actually consistent with results from a previous study [22], Which addresses cost-sensitive support vector machine algorithm for vector data, while CogBoost is a boosting algorithm for graph classification.

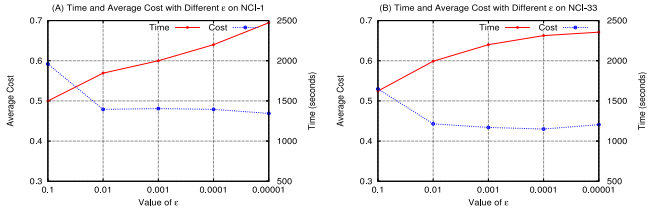

Fig. 7. Average cost (left y-axis) and algorithm runtime (right y-axis) with respect to different ϵ values ($x-$axis). (A) NCI-1, and (B) NCI-33.

7.2.2 *Performance* w.r.t. *Different* ϵ *Values*

In CogBoost, the parameter ϵ controls CogBoost's solutions in solving the cutting plane algorithm (Algorithm 3). In order to validate ϵ's impact on the algorithm performance, we vary ϵ values and report CogBoost's performance in Fig. 7.

Fig. 7 shows that for large ϵ values (e.g. $\epsilon = 0.1$ on NCI-1 data set), the corresponding average cost is also large. This is because a large ϵ value returns a solution far away from the optimal solution and results in poor performance for CogBoost. As ϵ continuously decreases (from 0.1 to 0.00001), the average cost on both NCI-1 and NCI-33 data sets decrease. This is because with a small ϵ value, CogBoost can return accurate solution for classification. However, the runtime consumption for smaller ϵ values will also increase because more iterations are required in the cutting algorithm. Our empirical results suggest that a moderate value (such as $\epsilon = 0.01$) has a good tradeoff between time complexity and average cost. So we set $\epsilon = 0.01$ as a default value in our experiments.

In summary, our experiments suggest that cost-sensitive graph classification is a much more complicated problem than traditional cost-sensitive learning, mainly because that graph classification heavily replies on subgraph feature exploration. Simply converting a graph data set into a vector representation, by using frequent subgraph features, and then applying cost-sensitive learning (like Fre+CSVM does) is far from optimal. Indeed, subgraph features play vital role for graph classification. By using a cost-sensitive subgraph exploration process and a cost-sensitive loss function, CogBoost demonstrates its superb performance for cost-sensitive graph classification.

8 Conclusion

In this paper, we formulated a cost-sensitive graph classification problem for large scale graph data sets. We argued that many real-world applications involve data with dependency structures and the cost of misclassifying samples in different classes is inherently different. This problem motivates us to consider effective graph classification algorithms with cost-sensitive capability and suitable for large scale graph data sets. To solve the problem, we proposed a fast boosting algorithm, CogBoost, which embeds the costs into the subgraph exploration and learning process. The boosting procedure utilizes an optimal loss function to minimize the misclassification costs by implementing the Bayes decision rule. To enable fast training on large scale graphs, a cutting plane

formulation is derived so that the linear problem can be solved efficiently in each iteration. Experimental results on large real-life graph data sets validate our designs.

ACKNOWLEDGMENTS

The authors thank anonymous reviewers whose criticisms and insightful comments helped to improve this paper. This work is partially supported by US National Science Foundation under Grant No. IIP-1444949, and by National Scholarship for Building High Level Universities, China Scholarship Council (No. 2011630030). X. Zhu is the corresponding author.

REFERENCES

[1] H. Kashima, K. Tsuda, and A. Inokuchi, *Kernels for Graphs*, Cambridge (Massachusetts): MIT Press, 2004, ch. In: Schlkopf B, Tsuda K, Vert JP, editors. Kernel methods in computational biology.

[2] M. Fang and D. Tao, "Networked bandits with disjoint linear payoffs," in *Proc. 20th ACM SIGKDD Int. Conf. Knowl. Discovery Data Mining*, 2014, pp. 1106–1115.

[3] S. Pan, X. Zhu, C. Zhang, and P. S. Yu, "Graph stream classification using labeled and unlabeled graphs," in *Proc. IEEE 29th Int. Conf. Data Eng.*, 2013, pp. 398–409.

[4] K. Riesen and H. Bunke, "Graph classification by means of lipschitz embedding," *IEEE Trans. Syst., Man, Cybern.-B*, vol. 39, no. 6, pp. 1472–1483, Dec. 2009.

[5] H. Saigo, S. Nowozin, T. Kadowaki, T. Kudo, and K. Tsuda, "gboost: A mathematical programming approach to graph classification and regression," *Mach. Learn.*, vol. 75, pp. 69–89, 2009.

[6] Y. Zhu, J. Yu, H. Cheng, and L. Qin, "Graph classification: a diversified discriminative feature selection approach," in *Proc. 21st ACM Int. Conf. Inform. Knowl. Management*, 2012, pp. 205–214.

[7] X. Kong and P. Yu, "Semi-supervised feature selection for graph classification," in *Proc. 16th ACM SIGKDD Int. Conf. Knowl. Discovery Data Mining*, 2010, pp. 793–802.

[8] J. Tang and H. Liu, "An unsupervised feature selection framework for social media data," *IEEE Trans. Knowl. Data Eng.*, vol. 26, no. 12, pp. 2914–2927, Dec. 2014.

[9] H. Fei and J. Huan, "Boosting with structure information in the functional space: An application to graph classification," in *Proc. 16th ACM SIGKDD Int. Conf. Knowl. Discovery Data Mining*, 2010, pp. 643–652.

[10] J. Wu, X. Zhu, C. Zhang, and P. Yu, "Bag constrained structure pattern mining for multi-graph classification," *IEEE Trans. Knowl. Data Eng.*, vol. 26, no. 10, pp. 2382–2396, Oct. 2014.

[11] S. Pan and X. Zhu, "Graph classification with imbalanced class distributions and noise," in *Proc. Int. Joint Conf. Artif. Intell.*, 2013, pp. 1586–1592.

[12] P. Tiwari, J. Kurhanewicz, and A. Madabhushi, "Multi-kernel graph embedding for detection, gleason grading of prostate cancer via mri/mrs," *Med. Image Anal.*, vol. 17, no. 2, pp. 219–235, 2013.

[13] M. Deshpande, M. Kuramochi, N. Wale, and G. Karypis, "Frequent substructure-based approaches for classifying chemical compounds," *IEEE Trans. Knowl. Data Eng.*, vol. 17, no. 8, pp. 1036–1050, Aug. 2005.

[14] P. Domingos, "MetaCost: A general method for making classifiers cost-sensitive," in *Proc. 5th ACM SIGKDD Int. Conf. Knowl. Discovery Data Mining*, 1999, pp. 155–164.

[15] K. Veropoulos, C. Campbell, and N. Cristianini, "Controlling the sensitivity of support vector machines," in *Proc. Int. Joint Conf. Artif. Intell.*, 1999, pp. 55–60.

[16] C. Elkan, "The foundations of cost-sensitive learning," in *Proc. Int. Joint Conf. Artif. Intell.*, vol. 17, no. 1, 2001, pp. 973–978.

[17] S. Zhang, Z. Qin, C. X. Ling, and S. Sheng, "Missing is useful": Missing values in cost-sensitive decision trees," *IEEE Trans. Knowl. Data Eng.*, vol. 17, no. 12, pp. 1689–1693, Dec. 2005.

[18] F. R. Bach, D. Heckerman, and E. Horvitz, "Considering cost asymmetry in learning classifiers," *J. Mach. Learn. Res.*, vol. 7, pp. 1713–1741, 2006.

[19] B. Zadrozny, J. Langford, and N. Abe, "Cost-sensitive learning by cost-proportionate example weighting," in *Proc. 3rd IEEE Int. Conf. Data Mining*, 2003, p. 435.

[20] H. Masnadi-Shirazi and N. Vasconcelos, "Cost-sensitive boosting," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 33, no. 2, pp. 294–309, Feb. 2011.

[21] A. C. Lozano and N. Abe, "Multi-class cost-sensitive boosting with p-norm loss functions," in *Proc. 14th ACM SIGKDD Int. Conf. Knowl. Discovery Data Mining*, 2008, pp. 506–514.

[22] H. Masnadi-Shirazi and N. Vasconcelos, "Risk minimization, probability elicitation, and cost-sensitive SVMs," in *Proc. Int. Conf. Mach. Learn.*, 2010, pp. 759–766.

[23] N. Abe, B. Zadrozny, and J. Langford, "An iterative method for multi-class cost-sensitive learning," in *Proc. 10th ACM SIGKDD Int. Conf. Knowl. Discovery Data Mining*, 2004, pp. 3–11.

[24] X. Zhu and X. Wu, "Class noise handling for effective cost-sensitive learning by cost-guided iterative classification filtering," *IEEE Trans. Knowl. Data Eng.*, vol. 18, no. 10, pp. 1435–1440, Oct. 2006.

[25] X. Zhu and X. Wu, "Cost-constrained data acquisition for intelligent data preparation," *IEEE Trans. Knowl. Data Eng.*, vol. 17, no. 11, pp. 1542–1556, Nov. 2005.

[26] S. Lomax and S. Vadera, "A survey of cost-sensitive decision tree induction algorithms," *ACM Comput. Surv.*, vol. 45, no. 2, pp. 16:1–16:35, 2013.

[27] G. Karakoulas and J. Shawe-Taylor, "Optimizing classifiers for imbalanced training sets," in *Proc. Adv. Neural Inform. Process. Syst. II*, 1999, p. 253–259.

[28] X. Yan and J. Han, "gspan: Graph-based substructure pattern mining," in *Proc. IEEE Int. Conf. Data Mining*, 2002, p. 721.

[29] X. Wu, X. Zhu, G. Wu, and W. Ding, "Data mining with big data," *IEEE Trans. Knowl. Data Eng.*, vol. 26, no. 1, pp. 97–107, Jan. 2014.

[30] S. Ranu and A. Singh, "Graphsig: A scalable approach to mining significant subgraphs in large graph databases," in *Proc. IEEE Int. Conf. Data Eng.*, 2009, pp. 844–855.

[31] J. Wu, Z. Hong, S. Pan, X. Zhu, C. Zhang, and Z. Cai, "Multi-graph learning with positive and unlabeled bags," in *Proc. SIAM Int. Conf. Data Min.*, 2014, pp. 1–12.

[32] H. Fei and J. Huan, "Structure feature selection for graph classification," in *Proc. 17th ACM Conf. Inform. Knowl. Manage.*, 2008, pp. 991–1000.

[33] N. Jin, C. Young, and W. Wang, "Graph classification based on pattern co-occurrence," in *Proc. 18th ACM Conf. Inform. Knowl. Manage.*, 2009, pp. 573–582.

[34] M. Thoma, H. Cheng, A. Gretton, J. Han, H. Kriegel, A. Smola, L. Song, P. Yu, X. Yan, and K. Borgwardt, "Near-optimal supervised feature selection among frequent subgraphs," in *Proc. SIAM Int. Conf. Data Mining*, 2009, pp. 1076–1087.

[35] T. Kudo, E. Maeda, and Y. Matsumoto, "An application of boosting to graph classification," in *Proc. 18th Annu. Conf. Neural Inform. Process. Syst.*, 2004, pp. 729–736.

[36] J. Wu, S. Pan, X. Zhu, and Z. Cai, "Boosting for multi-graph classification," *IEEE Trans. Cybernetics*, vol. 45, no. 3, pp. 430–443, Mar. 2015.

[37] S. Pan, J. Wu, X. Zhu, and C. Zhang, "Graph ensemble boosting for imbalanced noisy graph stream classification," *IEEE Trans. Cybernetics*, vol. 45, no. 5, pp. 940–954, May 2015.

[38] W. Fan, S. J. Stolfo, J. Zhang, and P. K. Chan, "Adacost: Misclassification cost-sensitive boosting," in *Proc. 16th Int. Conf. Mach. Learn.*, 1999, pp. 97–105.

[39] H. Masnadi-Shirazi, N. Vasconcelos, and A. Iranmehr, "Cost-sensitive support vector machines," *arXiv:1212.0975*, 2012.

[40] R. Tibshirani, "Regression shrinkage and selection via the lasso," *J. Roy. Statist. Soc.. Series B (Methodological)*, vol. 58, pp. 267–288, 1996.

[41] S. Nash and A. Sofer, *Linear and Nonlinear Programming*. New york, NY, USA: McGraw-Hill, 1996.

[42] T. Joachims, "Training linear SVMs in linear time," in *Proc. 12th ACM SIGKDD Int. Conf. Knowl. Discovery Data Mining*, 2006, pp. 217–226.

[43] N. Megiddo, "On the complexity of linear programming," in *Proc. 5th World Congress Adv. Econ. Theory*, 1987, pp. 225–268.

Shirui Pan received the master's degree in computer science from Northwest A&F University, Yangling, Shaanxi, China, in 2011. Since September 2011, he has been working towards the PhD degree in the Centre for Quantum Computation and Intelligent Systems (QCIS), Faculty of Engineering and Information Technology, University of Technology, Sydney (UTS). His research focuses on data mining and machine learning.

Jia Wu received the bachelor's degree in computer science from the China University of Geosciences (CUG), Wuhan, China, in 2009. Since September 2009, he has been working towards the PhD degree under the Master-Doctor combined program in computer science from CUG. Besides, he is also working towards the PhD degree in QCIS Centre, Faculty of Engineering and Information Technology, University of Technology, Sydney (UTS), Australia. His research focuses on data mining and machine learning.

Xingquan Zhu received the PhD degree in computer science from Fudan University, Shanghai, China. He is an associate professor in the Department of Computer & Electrical Engineering and Computer Science, Florida Atlantic University. Prior to that, he was with the Centre for Quantum Computation & Intelligent Systems, University of Technology, Sydney, Australia. His research interests mainly include data analytics, data mining, machine learning, and bioinformatics. Since 2000, he has published more than 180 refereed journal and conference papers in these areas, including two Best Paper Awards and one Best Student Paper Award. Dr. Zhu was the recipient of an ARC Future Fellowship in 2010. He is an associate editor of the *IEEE Transactions on Knowledge and Data Engineering* (2014-date), and is serving on the editorial board of Journal of Big Data (2014-date), the *International Journal of Social Network Analysis and Mining* (2010-date) and *Network Modeling Analysis in Health Informatics and Bioinformatics Journal* (2014-date). He was the program committee co-chair for the 14th IEEE International Conference on Bioinformatics and BioEngineering (BIBE-2014), IEEE International Conference on Granular Computing (GRC-2013), 23rd IEEE International Conference on Tools with Artificial Intelligence (ICTAI-2011), and the 9th International Conference on Machine Learning and Applications (ICMLA-2010). He also served as a conference co-chair for ICMLA-2012. He also serves (or served) as program vice chairs, finance chairs, publicity co-chairs, program committee members for many international conferences, including ACM-KDD, IEEE-ICDM, and ACM-CIKM. He is a senior member of the IEEE.