Positive and Unlabeled Learning for Graph Classification

Yuchen Zhao
Department of Computer Science
University of Illinois at Chicago
Chicago, IL
Email: yzhao@cs.uic.edu

Xiangnan Kong
Department of Computer Science
University of Illinois at Chicago
Chicago, IL
Email: xkong4@uic.edu

Philip S. Yu
Department of Computer Science
University of Illinois at Chicago
Chicago, IL
Email: psyu@cs.uic.edu

***Abstract*—The problem of graph classification has drawn much attention in the last decade. Conventional approaches on graph classification focus on mining discriminative subgraph features under supervised settings. The feature selection strategies strictly follow the assumption that both positive and negative graphs exist. However, in many real-world applications, the negative graph examples are not available. In this paper we study the problem of how to select useful subgraph features and perform graph classification based upon only positive and unlabeled graphs. This problem is challenging and different from previous works on PU learning, because there are no predefined features in graph data. Moreover, the subgraph enumeration problem is NP-hard. We need to identify a subset of unlabeled graphs that are most likely to be negative graphs. However, the negative graph selection problem and the subgraph feature selection problem are correlated. Before the reliable negative graphs can be resolved, we need to have a set of useful subgraph features. In order to address this problem, we first derive an evaluation criterion to estimate the dependency between subgraph features and class labels based on a set of estimated negative graphs. In order to build accurate models for the PU learning problem on graph data, we propose an integrated approach to concurrently select the discriminative features and the negative graphs in an iterative manner. Experimental results illustrate the effectiveness and efficiency of the proposed method.**

***Keywords*-graph classification; positive and unlabeled data; feature selection;**

I. Introduction

Graphs are ubiquitous and very important tools to model diverse kinds of data with complex structures. Examples include social networks, Internet, chemical compounds, program flows, *etc.* There are increasing needs for building models to classify graph data. In software engineering, programmers like to study how to automatically identify bugs/errors in program flows [1]; in molecular drug discovery, researchers want to be able to automatically classify the chemical compounds' anti-cancer activities in order to find new drugs for cancers or chronic diseases [2], [3]. Motivated by these challenges, *graph classification* has received much attention in the last decade.

In the literature, many research efforts have been devoted to graph classification [4], [3], [5], [6]. Conventional methods focus on mining discriminative subgraph features [7] under supervised settings, which assume that both positive and negative examples are available. However, in many real-world applications, the negative graphs are missing, while only the positive and unlabeled graphs are available. Examples are as follows:

- In software engineering, the experts need to identify some program flows with errors as well as some correct examples. Usually, it is easy to determine some program flows with obvious errors. However it is almost impossible to confirm that a program is guaranteed to be completely bug-free or absolutely correct with no errors.
- In molecular drug discovery, researchers are more likely to publish results of newly discovered drugs with positive outcomes on a disease. However the results of chemicals with negative outcomes can rarely be published in the literature.

Thus, it is much desirable that if one can train an accurate classification model on graph data with only positive and unlabeled examples. This setting is also known as Positive and Unlabeled (PU) learning, which aims to learn from data with only positive and unlabeled examples. It has been shown to be useful in many data mining tasks [8], [9], [10], [11], such as text mining, uncertain data mining, stream mining, *etc.*

Formally, the graph PU learning problem corresponds to training a model to identify a subset of unlabeled graphs that are most likely to be negative graphs. These graphs are called *reliable negative graphs/examples*. Positive and unlabeled learning is particularly challenging on graph data. Conventional PU learning approaches can identify a group of reliable negative examples in the vector space. These approaches assume that the full set of useful features is available [9], [10], [11]. However, graph data are not directly represented in a feature space, and they require an additional subgraph feature mining process by evaluating the subgraph patterns in a graph data set. Usually the number of subgraphs in a graph data set is extremely large. Only a small number of the features are relevant to the classification task [7], [3]. What makes this problem even more challenging is that the subgraph enumeration problem and the subgraph isomorphism test problem are NP-hard. Thus, it is impossible

to enumerate all the subgraph features and adopt existing methods for PU learning.

Despite its value and significance, the PU learning for graph data has not been investigated in this context. If we consider PU learning and subgraph feature mining problems together, the major research problems are summarized as follows:

- *Lack of Features*: one problem with the graph PU learning lies in the complex structures and lack of features in the graph data. Traditional PU learning techniques focus on identifying reliable negative examples in a fixed feature space. However, graphs are not directly represented in a meaningful feature space. For the graph data, we usually need to first find a set of subgraph features, and then convert each graph into a vector based upon these features. The performance of the PU learners depends strongly on what subgraph features are used.
- *Lack of Negative Graphs*: another problem with the graph PU learning lies in the absence of negative training examples. Conventional graph classification approaches focus on mining discriminative subgraph features [7], [3] under supervised settings. Figure 1 illustrates the supervised graph classification process. In the subgraph feature mining step, it assumes that both positive and negative graph examples are available. However, when all the negative graphs are missing/unavailable, the conventional feature evaluation methods cannot work in this case.
- *Inaccuracy of Reliable Negative Examples*: Conventional PU learning algorithms work in a way by iteratively adding new discovered negative examples to a reliable negative set. The classification model is built based on the given positive set and reliable negative set at each iteration. However, the reliable negative set in a graph scenario is prone to contain more false negatives due to the fact that graph features do not naturally exist and are discovered gradually during the iterations. In other words, the examples in the reliable negative set are not that "reliable" for the graph data. Simply growing the reliable negative set will bring more errors to the classification model at each iteration. This will eventually lead to severe quality degradation.

A straightforward solution to the above problems is the two-stage graph PU learning approach. In this approach, we first mine a set of frequent subgraphs and use them as features. Then we apply traditional PU learning methods on the data with the above features. Obviously this approach is ineffective due to the fact that many discriminative features are not the most frequent ones [7], [3]. Thus some discriminative features will be missed by the frequent subgraph mining approach.

In this paper, we introduce a novel framework called

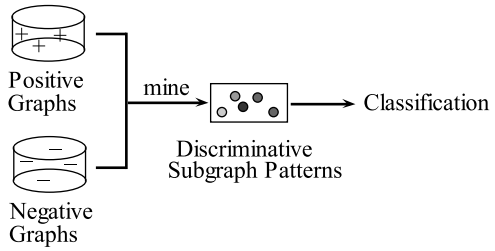


Figure 1. Supervised Graph Classification Process

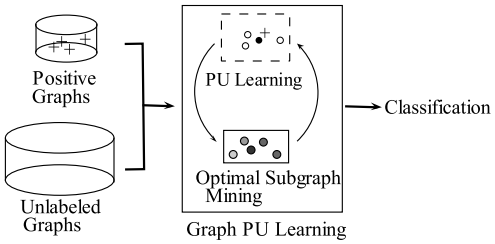


Figure 2. Graph PU Learning Process

GPU-Learning to mine useful subgraph features using only positive and unlabeled graphs. Our framework is illustrated in Figure 2. We first derive an evaluation criterion to estimate the dependency between subgraph features and class labels based on a set of estimated negative graphs. We then devise an integrated framework to evaluate the usefulness of subgraph features based on only positive and unlabeled graphs. This integrated framework can optimize both graph feature selection and class label assignment concurrently by capturing their dependencies. We iteratively update the evaluation of useful subgraph features and estimation of reliable negative graphs in the graph data set. Finally we perform comprehensive experiments on real-world graph classification tasks with only positive and unlabeled examples. The experiment results show that the proposed graph PU learning method can effectively boost the graph classification performance using only positive and unlabeled graphs.

The rest of the paper is organized as follows. In Section II, we discuss related work on graph classification and PU learning. In Section III, we propose an optimization framework on graph positive and unlabeled learning, and analyze how to solve the optimization problem by solving two subproblems. Section IV reports the experimental results. Section V presents the conclusions and summary.

II. Related Work

To the best of our knowledge, this paper is the first work on positive and unlabeled learning problem for graph classification. Some research works have been done in related areas.

The problem of PU learning has been investigated by many researchers. In [12], the authors conducted a theoretical study of PAC learning from positive and unlabeled examples. A number of practical approaches have also been proposed to study building classifiers from positive and unlabeled data [10], [13], [11]. Many of these algorithms follow a two-step strategy, which can be done by first extracting a set of *reliable negative* examples, and then performing the classification task on the positive examples and reliable negative examples by using classification algorithms such as Naive Bayes, SVM and EM. For example, [13] proposed the S-EM method, which uses an algorithm called Spy in the first step to obtain reliable negative examples. Then it uses EM algorithm as the second step to build classifiers. [11] reported the PEBL method. It uses the 1-DNF in the first step, and runs SVM iteratively in the second step for building classifiers. Despite extracting reliable negative examples from the unlabeled examples, it is also possible to identify positive examples from the unlabeled data by iteratively growing both the positive and reliable negative sets [14].

Besides the two-step strategy, [15] adopted a NB based method, PNB, to remove the effect of positive examples from the unlabeled set, but it requires the user to provide the probability of the positive label. One-class SVM can also be utilized for the purpose to classify positive and unlabeled examples, which uses only positive data to build a SVM classifier. One shortcoming of using one-class SVM to handle PU learning problem is it does not consider useful information in the unlabeled examples.

A number of approaches recently have been reported to address different environments under the PU learning setting, *e.g.* uncertain data [16], stream data [9], text [14], [10], and Web [11] *etc.* However, as we have mentioned, there is no PU learning algorithm devised for the graph domain.

Mining subgraph features from graph data have also been studied in recent years. The aim of such approaches is to extract useful subgraph features from a set of graphs by adopting some filtering criteria. Depending upon whether the label information is considered in the feature mining process, the existing works can roughly be classified into two types: unsupervised and supervised. In the unsupervised approaches, the frequencies are used as the subgraph feature evaluation criterion, where the aim is to collect frequently appearing subgraph features. For example, Yan and Han developed a depth-first search algorithm: gSpan [17], which can build a lexicographic order among the graphs, and map each graph to a unique minimum DFS code as its canonical label. Based on the lexicographic order, gSpan algorithm adopts the depth-first search in the DFS code tree to mine frequent connected subgraphs efficiently. There are also many other frequent subgraph feature mining approaches have been developed in the last decade, *e.g.* FSG [18], MoFa [19], FFSM [20], and Gaston [21]. Furthermore, supervised subgraph feature mining approaches have also been proposed in the literature, such as LEAP [3], CORK [4], which search for discriminative subgraph features for graph classifications. In addition, gSSC [22] addresses the problem of feature selection for graph classification under semi-supervised settings.

III. Graph PU-Learning Framework

In this section, we will introduce the **G**raph **P**ositive and **U**nlabeled **Learning** (GPU-Learning) framework which is used for building graph classifiers on positive and unlabeled examples. We will first introduce some notations and definitions. Assume that we have a set of graph objects, denoted as $\mathcal{D} = \{G_1, G_2, ..., G_n\}$. The graph objects in set $\mathcal{D}$ are defined as in Definition 1, and each graph object is associated with a class label. Let $Y = [y_1, y_2, ..., y_n]$ where y_i is the label of G_i. Thus y_i can be $+1$ (positive), -1 (negative) or 0 (unlabeled). The initial class labels in the graph data set $\mathcal{D}$ can only be positive and unlabeled.

Definition *1 (Graph Object):* Each graph object can be represented by $G = (V, E, L)$, where V is a set of vertices, $E \subseteq V \times V$ is a set of edges, and L is a function assigning labels to the vertices.

Definition *2 (Subgraph):* Let $G' = (V', E', L')$ and $G = (V, E, L)$ be two graphs. G' is a subgraph of G ($G' \subseteq G$) iff there exist an injective function $\psi : V' \to V$ s.t. (1) $\forall\ v \in V'$, $L'(v) = L(\psi(v))$; (2) $\forall (u, v) \in E'$, $(\psi(u), \psi(v)) \in E$. If G' is a subgraph of G, then G is a supergraph of G'.

Given a graph data set $\mathcal{D}$ and a minimum support θ, the frequent subgraphs are the subgraphs whose frequencies are greater than or equal to the minimum support θ. Given a set of subgraph features $\{g_1, g_2, ..., g_d\}$, a graph object G can be represented as a binary vector $x = [x_1, x_2, ..., x_d]^\top$ with d dimensions, where $x_i = 1$ if subgraph feature $g_i \subseteq G$; otherwise $x_i = 0$.

In order to solve the graph PU learning problem, the key challenges are described as follows:

(a) How can we identify the *reliable negative graphs* among the unlabeled graphs? The most reliable negative graphs are defined as a set of unlabeled graph which are most likely to be negative graphs.

(b) How can we evaluate the usefulness of the subgraph features based on only positive and unlabeled graph examples?

The above two problems are closely related to each other in graph PU learning. The reasons are as follows. In problem (a), we need to identify a set of reliable negative graphs. However, before the reliable negative graphs can be estimated, a set of useful subgraph features must be available. The performance of PU learning for estimating reliable negative graphs directly depends on the quality of the subgraph features that are used. The better the feature

set, the more effectively we can estimate the reliable negative graphs. In addition, for problem (b), we need to select a set of useful features for the graph classification task, where the feature evaluation performance directly depends on the quality of the reliable negative graphs that are selected in the PU learning process. The more accurate estimation we have on the negative graphs, the more effectively we can evaluate the useful subgraph features in the feature selection process. In this case, the qualities of reliable negative examples and subgraph features are closely correlated. Thus we propose the following optimization framework to concurrently identify reliable negative graph examples and find useful subgraph features.

A. Optimization Framework

The goal of the framework is to simultaneously find a set of optimal subgraph features and identify a list of reliable negative examples. We first formally introduce the notations as follows:

- $S = \{g_1, g_2, ..., g_d\}$: a given set of subgraph features that we use to predict the labels. Usually we want to find a subset T ($T \subseteq S$, $T \neq \emptyset$) which can optimize the evaluation criteria between subgraph features and labels.
- t: an integer which determines the maximum number of subgraph features to be selected from the set S.
- $\epsilon(T, Y)$: an evaluation function to estimate the dependency between the selected feature set T and the labels of the graphs including the estimated labels for the unlabeled graphs. We note that the value of evaluation function is controlled by two variables: one is the subgraph feature set T, and the other is the label assignments Y. We want to maximize the function by selecting the most optimal subgraph features as well as estimating the negative examples accurately.
- T^*: the optimal set of subgraph features $T^* \subseteq S$.
- X: a $d \times n$ matrix representing the graph data set $\mathcal{D} = \{G_1, G_2, ..., G_n\}$ where each G_i is represented by a binary feature vector based on feature S. $X = [x_1, x_2, ..., x_n] = [f_1, f_2, ..., f_d]^\top$, where $X = [X_{ij}]_{d \times n}$, and $X_{ij} = 1$ iff $g_i \subseteq G_j$, otherwise $X_{ij} = 0$.
- C: an $n \times n$ diagonal matrix representing the confidence levels of (estimated) labels, *i.e.* the probabilities of the graphs to have certain labels. In details, C_{ii} indicates the probability of graph G_i to have the class label y_i. Since the positive labels are presumably given, the confidence of positive graph examples is always 1, *i.e.* $C_{ii} = 1$ when the label of graph G_i is positive. For the case of estimated negative examples, $C_{ii} \in (0, 1]$ iff the label of graph G_i is estimated to be negative.

In order to simultaneously optimize the subgraph feature selection and label assignment, we will use the following framework:

$$(T^*, Y^*) = \underset{T \subseteq S, Y \in \{-1,1\}^n}{\arg\max} \epsilon(T, Y) \qquad s.t. \ \ |T| \leq t. \quad (1)$$

where we select a subset of subgraph features T from S, and the number of subgraph features in the set T must be less than or equal to t.

In order to solve the above optimization problem, we assume that the optimal subgraph features and the estimated labels for the unlabeled graphs should follow the properties of *dependency maximization*, *i.e.*, graphs with the same labels are more likely to have similar features. In details, from the subgraph feature selection perspective, if we are given a set of estimated negative graphs among the unlabeled graphs, the optimal subgraph features should be able to maximize the dependency between the features and the labels of graphs including the estimated negative ones. From the PU learning perspective, given a set of subgraph features, the optimal set of estimated negative graphs are those that can maximize the dependency between the features and labels of the graphs. *Dependency maximization* has also been successfully applied to many research topics, e.g. vector space dimensionality reduction [23] and classification on multi-label graph data [24]. Hilbert-Schmidt Independence Criterion (HSIC) has been designed to measure statistical dependency based on the covariance operators in kernel space, which we will use as the evaluation criterion for dependency in Eq. 1. The details of HSIC and its empirical estimate can be found in [25]. Based on the representation of HSIC, we will have the following graph PU learning framework:

$$\begin{aligned} &\max_{T,Y} \qquad tr(K_T H L_Y H) \\ &s.t. \quad |T| \leq t, \ T \subseteq S, \ Y \in \{-1, 1\}^n. \end{aligned} \quad (2)$$

where tr(.) is the trace of a matrix, and matrices K_T, H, $L_Y \in R^{n \times n}$. The matrix K_T is the inner product of the graph vector representations based on the selected subgraph features T, which can also be considered as the kernel matrix of graphs with the kernel function $K(G_i, G_j) = \langle D_T x_i, D_T x_j \rangle$. D_T here is a diagonal matrix representing which features are selected in the feature set T. In details, $D_{Tii} = 1$ iff graph feature g_i is selected in T; otherwise $D_{Tii} = 0$. $H = [H_{ij}]_{n \times n}$ is the "centering" matrix, and $H_{ij} = \delta_{ij} - 1/n$, where δ_{ij} is the indicator function which equals to 1 when $i = j$ and 0 otherwise. $L_Y = [L_{Yij}]_{n \times n}$ is the kernel matrix for graph labels including the estimated negative graphs. The kernel function can be written as $L(y_i, y_j) = \langle C_{ii} y_i, C_{jj} y_j \rangle$, where the matrix C indicates the confidence levels of graph label assignments.

It is clear that there are two sets of variables in the optimization framework, which are the selected subgraph features T and a list of assigned labels Y. Both sets of variables can dynamically change and need to be optimized to maximize Eq. 2. In addition they are also coupled. Therefore,

Algorithm *optimize_features* (Graph Data Set: $\mathcal{D}$,
Labels: Y,
Number of Selected Subgraph Features: t,
Minimum Support: θ);
begin
$T = \emptyset$;
Recursively visit the DFS Code Tree in gSpan:
g = currently visited subgraph in DFS Code Tree;
Compute the value of $q(g)$ using Eq. 4;
if $|T| < t$, then:
$T = T \cup \{g\}$;
else if $q(g) > \min_{g' \in T} q(g')$, then:
remove the feature with the lowest quality from T;
$T = T \cup \{g\}$;
if $freq(g) \geq \theta$ and $\hat{q}(g) \geq \min_{g' \in T} q(g')$:
Depth-first search subtree rooted from node g;
return T.
end

Figure 3. the Algorithm to Optimize Features

optimizing them together can be extremely difficult. We will divide the optimization framework into two subproblems, which are:

- Given the graph data set and labels, how to find the optimal subgraph features.
- Given the graph data set and a set of features, how to accurately estimate the class labels.

Our approach is to integrate the above two subproblems and optimize the whole framework concurrently. In the following part, we will first show how to optimize the above two subproblems, and then integrate them to solve the graph PU learning problem.

1) Optimize Subgraph Features: We will first describe how to optimize the selected subgraph features T by assuming the label assignments are given. We can rewrite the optimization Eq. 2 as:

$$\begin{aligned} & tr(K_T H L_Y H) \\ = \; & tr(X^\top D_T D_T X H C Y^\top Y C H) \\ = \; & tr(D_T X H C Y^\top Y C H X^\top D_T) \\ = \; & \sum_{g_i \in T} (f_{g_i}^\top H C Y^\top Y C H f_{g_i}) \\ = \; & \sum_{g_i \in T} (f_{g_i}^\top M f_{g_i}) \end{aligned}$$

where $M = HCY^\top YCH$ and f_{g_i} is the indicator vector for subgraph feature g_i. Here $f_{g_i} = [f_{g_i}^1, ..., f_{g_i}^n]^\top$, and $f_{g_i}^k = 1$ when the graph object G_k has the subgraph feature g_i for $k = 1, 2, ..., n$. Let the function $h(g_i, M)$ denote the value of $f_{g_i}^\top M f_{g_i}$. We can observe that in Eq. 2, $tr(K_T H L_Y H)$ equals to the sum of $h(g_i, M)$ over all graph feature $g_i \in T$. Therefore the problem of maximizing the value of $tr(K_T H L_Y H)$ is equal to finding a subset of features that can maximize the sum of $h(g_i, M)$, which can be represented as:

$$\max_T \quad \sum_{g_i \in T} h(g_i, M) \qquad s.t. \;\; |T| \leq t, T \subseteq S. \tag{3}$$

DEFINITION *3:* (Feature Quality Measurement) Given a graph data set $\mathcal{D} = \{G_1, G_2, ..., G_n\}$, its corresponding labels y_i for graph G_i and the label confidence level matrix C, the quality of a feature g_i can be measured by:

$$q(g_i) = h(g_i, M) = f_{g_i}^\top M f_{g_i} \tag{4}$$

where $M = HCY^\top YCH$.

In the above definition, a higher value of feature quality measurement $q(g_i)$ represents larger dependency between this feature and the labels. In other words, good subgraph features should have high $q(g_i)$ values. The solution to the subgraph feature optimization can be done by first computing the feature quality measurement $q(g_i)$ on each subgraphs in S, and then selecting the top t subgraph features which have the highest values of $q(g_i)$. Before we introduce the algorithm to optimize features, we can derive the following upper bound of $q(g_i)$:

Lemma 1: Given any two subgraphs g, $g' \in S$, g' is a supergraph of g ($g' \supseteq g$). The feature quality measurement of g' is bounded by $\hat{q}(g)$ (i.e., $q(g') \leq \hat{q}(g)$):

$$\hat{q}(g) = f_g^\top \hat{M} f_g \tag{5}$$

where $\hat{M}$ is defined as $\hat{M}_{ij} = \max(M_{ij}, 0)$.

Proof:

$$q(g') = f_{g'}^\top M f_{g'} = \sum_{i,j:G_i,G_j \in \Omega(g')} M_{ij}$$

where $\Omega(g') = \{G_i | g' \subseteq G_i, 1 \leq i \leq n\}$. Since g is a subgraph of g', we have $\Omega(g) \supseteq \Omega(g')$.

$$\begin{aligned} q(g') \;\; = \;\; & \sum_{i,j:G_i,G_j \in \Omega(g')} M_{ij} \leq \sum_{i,j:G_i,G_j \in \Omega(g')} \hat{M}_{ij} \\ \leq \;\; & \sum_{i,j:G_i,G_j \in \Omega(g)} \hat{M}_{ij} = \hat{q}(g) \end{aligned}$$

Therefore, for any $g' \supseteq g$, $q(g')$ is bounded by $\hat{q}(g)$. ■

In order to obtain the subgraph feature set S, we use a well-known depth-first search algorithm *gSpan* [17] to enumerate frequent subgraphs. The key idea of *gSpan* is to first assign a unique minimum DFS code to each graph, and then discover all frequent subgraphs by a pre-order traversal of the tree. The minimum DFS codes of two graphs are the same iff they are isomorphic. We extend this idea and utilize *Lemma* 1 to efficiently prune the DFS code tree with a *branch-and-bound* strategy.

We first initialize the set of selected subgraph features T to be empty. In the depth-first search through the DFS code tree, we compute the quality of each current visited subgraph g using Eq. 4. If the number of features in T is less than or equal to t, we will add the current subgraph

Algorithm *optimize_labels* (Graph Data Set: $\mathcal{D}$,
Labels: Y,
Subgraph Features: T);
begin
$P = \{G_i | G_i \in \mathcal{D}, y_i = 1\}$;
$N = \{G_i | G_i \in \mathcal{D}, y_i = -1\}$;
Build an SVM classifier M_{SVM} using P and N;
for each graph $G_i \in \{G_j | G_j \in \mathcal{D}, y_j \neq 1\}$:
Predict class label of G_i by applying classifier M_{SVM};
if(G_i is not negative), $y_i = 0$, then continue;
$y_i = -1$;
C_{ii} = probability estimate from classifier M_{SVM};
end

Figure 4. the Algorithm to Optimize Labels

feature g into T. Otherwise we will compare the quality of subgraph g with the lowest quality subgraph in T. If $q(g)$ is higher which means the current visited subgraph g is a more useful feature than some of subgraph features in T, we will replace the subgraph in T which has the lowest quality measure with the subgraph feature g.

For each visited subgraph feature g, we also compute its upper bound $\hat{q}(g)$ before performing depth-first searching rooted from g. If $\hat{q}(g) < \min_{g' \in T} q(g')$, we can safely prune all branches originating from g. The reason is all supergraphs of g will have smaller values of $q(.)$ than any subgraph features in T. Thus skipping all supergraphs of g will not affect the quality of the final selected subgraph features but provide higher efficiency. By doing search space pruning, the searching can be quite efficient than the original *gSpan* since a large number of search spaces do not need to be visited. However, if $\hat{q}(g) \geq \min_{g' \in T} q(g')$, we cannot prune this branch since it is possible that there exists a supergraph $g' \supseteq g$ that is better than some subgraph features in T.

After all subgraphs in the DFS code tree are enumerated, the feature set T will be returned as the optimal set of subgraph features. The detailed algorithm is illustrated in Figure 3. We note that in order to continue to depth-first search the subgraphs rooted from g, g needs to satisfy two conditions: one is the frequency of feature g has to be greater than or equal to the minimum support, and the other is the upper bound of its supergraphs $\hat{q}(g)$ has to be greater than or equal to the quality measure of some selected features in T.

2) Optimize Class Labels: In this subsection, we will solve the second subproblem, which is how to obtain the class labels when the graph features T are given. The optimization can be approximated by building a classification model on the data set corresponding to the given features, and utilizing the classifier to predict the class labels of unlabeled examples. Any classifier which can output probability estimates should be sufficient for this purpose. Here we select Support Vector Machine (SVM) as the classifier.

Conventional PU learning algorithms expand the reliable negative set by iteratively adding new found negative examples. However, the challenge on the graph data are the quality of reliable negative set depends on the effectiveness of the graph features used. Presumably the quality of the initial graph features is low, but the feature set can be iteratively improved later. Therefore continuously growing a negative set generated from the low-quality initial graph features can bring more and more errors into the later iterations. In order to avoid this, at each iteration we reevaluate every unlabeled graph including those previously classified as negative ($G_i \in \{G_j | G_j \in \mathcal{D}, y_j = -1\}$), since some of them may be false negatives due to the previous ineffective graph features. Thus those graphs which are incorrectly classified as negative based on the previous graph features can have the opportunity to make their class labels corrected in the current run because of using a more accurate set of graph features.

Considering the positive graph examples are initially given in the data set with confidence 1.0, we only label estimated negative examples in order to avoid adding noise to the positive set. For those graphs classified as negative by classifier M_{SVM}, we use the method in [26] to compute their probability estimates by solving the following optimization problem:

$$\min_{p_1, p_{-1}} (r_{-1,1} p_1 - r_{1,-1} p_{-1})^2$$
$$s.t.\ p_1 \geq 0, p_{-1} \geq 0, p_1 + p_{-1} = 1. \quad (6)$$

where $r_{i,j}$ is the pairwise class probabilities of class labels i and j, *i.e.* the probability of an example belongs to class i given that it can only belong to class i or j, and p_i is the probability estimate of class label i. The value of p_{-1} is assigned to the corresponding value in the label confidence level matrix C for the negative graphs. The algorithm description on class label optimization can be found in Figure 4.

B. The GPU-Learning Algorithm

We can now make use of the solutions of the above two subproblems to optimize the evaluation criterion in Eq. 2. In order to start, we first set all unlabeled graphs to have label -1, use them together with the positive graphs and perform the initial feature selection by a depth-first search using the algorithm in Figure 3. It is clear that the quality of initial selected features is not optimal since it assumes all unlabeled examples are negative, but it provides a good starting point for the following optimization framework.

With the help of initial graph features, we use the algorithm in Figure 4 to classify unlabeled graph objects and compute their probability estimates. Since the initial features are obtained by assuming all unlabeled examples are negative, we only want to label the most possible negative graph objects. In details, we only select whose the probability estimates (p_{-1}) are above a particular threshold β as negatives and assign them class label -1. The value of

Algorithm GPU-Learning (Graph Data Set: $\mathcal{D}$, Labels: Y, Number of Selected Subgraph Features: t, Minimum Support: θ);
begin
  Set the labels of unlabeled graphs to be -1;
  while not(L changes in the previous run) **do**
  begin
    Update feature set:
      $T = optimize\_features(\mathcal{D}, Y, t, \theta)$;
    Update labels:
      $L = optimize\_labels(\mathcal{D}, Y, T)$;
      if (in the first iteration) **then**
        Compute the mean μ and standard deviation σ of the probability estimates of classified negative examples by M_{SVM};
        Do not update the labels of classified negative examples to be -1 if their probability estimates are less than or equal to $\mu - m \cdot \sigma$;
      end if
  end

  return a classifier built on $\mathcal{D}$, T and L;
end

Figure 5. The GPU-Learning Algorithm

β is picked to be m standard deviations[1] below the mean value of the probability estimates of all negative examples classified by M_{SVM}. The values of the label confidence level matrix C are also updated by the probability estimates correspondingly.

With the initial estimated negative examples, we optimize the graph features and label assignments by using the algorithms in Figure 3 and Figure 4 in an iterative way. Different from the traditional PU learning approaches, the graph features are dynamically selected at each iteration by using the function *optimize_features*, and each estimated negative graph is assigned a probability weight (confidence level) to assist the feature selection. We note that although the depth-first search for graph features need to be performed iteratively, only a small portion of the whole search tree is visited at each iteration because of the upper bound pruning in *Lemma* 1. Figure 5 shows the whole framework.

IV. Experiment Results

In this section, we will test our PU learning algorithm for graph classification on a number of real data sets. We will present experiment results for both effectiveness and efficiency of the proposed approach.

A. Data Sets

We used six real data sets in order to test our approach. The data sets used were from the following tasks:

- Toxicology prediction (PTC) task[2]: The first four data sets were collected from the toxicology prediction task, which contains the outcomes of biological tests for the carcinogenicity of chemicals using the graph structure of chemical compounds on different animal models. There are four data sets available corresponding to the animal model: (1) Female Mouse (**FM**), (2) Male Mouse (**MM**), (3) Female Rat (**FR**) and (4) Male Rat (**MR**). Each of these data sets contains more than 300 chemical compounds. The chemical compounds in the data sets are assigned with carcinogenicity labels for these animal models. On each animal model the carcinogenicity label is one of {CE, SE, P, E, EE, IS, NE, N}. We used {CE, SE, P} as the positive labels, and {NE, N} as the negative labels, similar to the setting in [24], [22], [27]. Each chemical compound is represented as a graph with an average of 25.7 vertices.
- Anti-cancer activity prediction (NCI1): We collected two more data sets from the PubChem website[3]. The data sets consist of anti-cancer activity records on chemical compounds against two types of cancers: breast (**MCF-7**) and leukemia (**MOLT-4**). For all chemical compounds, the one with an *active* label will be regarded as a positive example, whereas the *inactive* one will be treated as negative. The original data set are unbalanced, where the active label is around 5%. We randomly sampled 500 compounds which have balanced labels from each data set for classification.

Table I
Summary of Experimental Data Sets. "Pos%" denotes the percentage of positive graphs in each data set.

Name	Graphs#	Pos%	Description
FM	349	41.0	Female Mouse Toxicology
MM	336	38.4	Male Mouse Toxicology
FR	351	34.5	Female Rat Toxicology
MR	344	28.2	Male Rat Toxicology
MCF-7	27784	8.19	Breast Cancer
MOLT-4	39765	7.89	Leukemia Cancer

The summary of data sets is listed in Table I.

B. Methods

In order to show the effectiveness and efficiency of the proposed approach, we compared with a number of baseline approaches. We refer to our approach as the **GPU-Learning** method. Since there is no known method for PU learning for graph data, we used the standard PU learning approaches as baselines, in which the features are represented by top-k frequent subgraphs generated by *gSpan*. We compared with two variations of PU learning algorithms in [10]:

- **NB+SVM-I**: In the traditional PU learning two-step strategy setting, it uses Naive Bayesian technique in the first step to obtain a set of reliable negative examples from the unlabeled set. Then it runs SVM iteratively on the positive set and reliable negative set until converges.
- **Spy+SVM-I**: The difference between Spy+SVM-I and NB+SVM-I is in its initialization. Spy+SVM-I random-

[1]Here we pick this factor m to be 2.
[2]http://www.predictive-toxicology.org/ptc/
[3]http://pubchem.ncbi.nlm.nih.gov

ly samples a set of positive examples as "spies", and marks them as negative. Adding spies to the negative set allows the algorithm to understand the characteristics of the unknown positive examples in the unlabeled set.

In addition, it is also possible to solve the graph PU learning problem using a semi-supervised classification algorithm by assuming there are no negative examples:

- **gSSC+1-NN**: [22] proposed a method to classify semi-supervised graphs data. The idea is that the labeled graphs in different classes should be far from each other, while the ones in the same class should be close to each other and the unlabeled graphs should be well separated. The nearest neighbor (1-NN) classifier is used for classification while maintaining the above constraints.

C. Evaluation Metrics and Settings

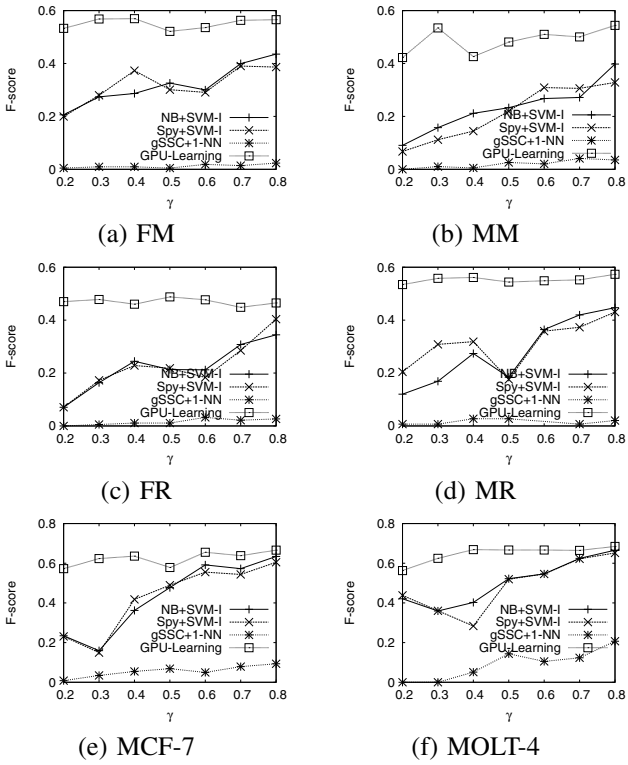
In order to evaluate the effectiveness of graph classification, we used the popular F-score on the positive class as the evaluation metric. F-score has also been widely used to measure the performance of PU learning techniques in the literature [16], [9], [10], [13]. F-score is defined as: $F = 2 \times \frac{p \times r}{p+r}$, where p is the precision and r is the recall of the test data set. F-score represents a harmonic mean between recall and precision, where F-score has a high value only if that both precision and recall are high.

For all data sets, we randomly selected 30% of the graphs as testing set in each run, and used the remaining graphs as training set. The baselines and GPU-Learning approaches were used to build classifiers on the training set, and we evaluated the performance by applying the classifiers on the testing set. In order to create a wide range of positive and unlabeled graph data for evaluation, we randomly sampled a portion with percentage γ (γ varies from 0.2 to 0.8) from the graphs that have positive labels, and used them as the positive set. The rest of the graphs that have positive labels with percentage $(1-\gamma)$ and all the graphs that have negative labels were treated as the unlabeled set.

Since the results of the PU classification can vary depending upon the randomly sampled test sets, we repeated each test 10 times with different random seeds and reported the average as the final score. Unless otherwise mentioned, the default value of γ is 0.5, the number of selected subgraph features is 30, and the minimum support of frequent subgraphs is 2% for *PTC* task and 5% for *NCI1* task.

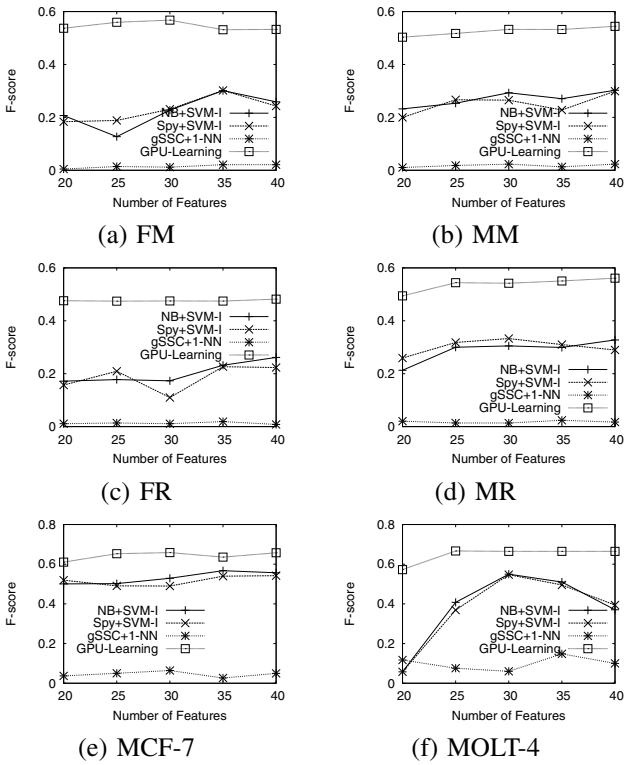
D. Effectiveness Results

We will first present the effectiveness results in terms of F-score for all six data sets. The effectiveness results on F-score for all baseline approaches and GPU-Learning with different γ settings are illustrated in Figure 6. The value of γ varies from 0.2 to 0.8 with an increment of 0.1, which is illustrated on the X-axis. The classification quality in terms of F-score with respect to the ground truth is illustrated



(a) FM (b) MM (c) FR (d) MR (e) MCF-7 (f) MOLT-4

Figure 6. F-score with Different γ



(a) FM (b) MM (c) FR (d) MR (e) MCF-7 (f) MOLT-4

Figure 7. F-score with Increasing Number of Features

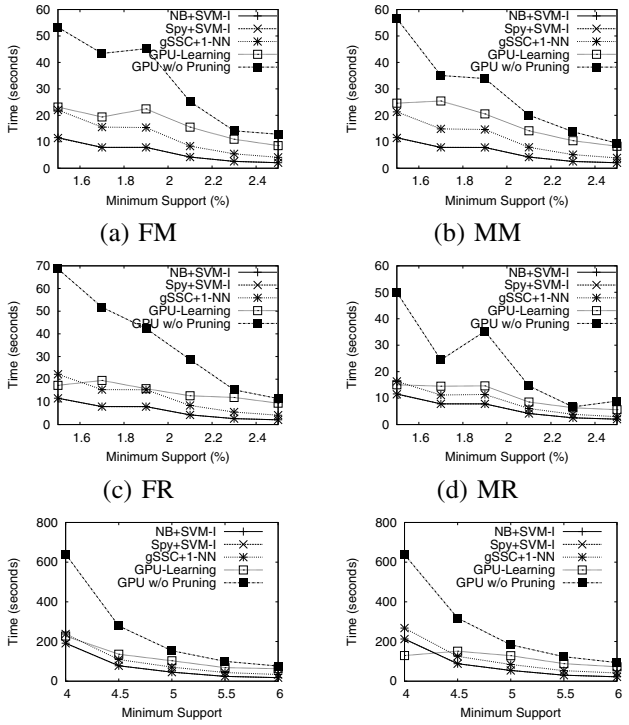
on the Y-axis. It is clear that GPU-Learning can achieve a much performance compared with other schemes. In details, gSSC+1-NN cannot achieve good performance since it is not designed for positive and unlabeled data. GPU-Learning improves the F-score by at least 0.1 up to 0.4 compared with NB+SVM-I and Spy+SVM-I in general. This is because many infrequent subgraphs can be very useful and a set of dynamically selected graph features can indeed improve the classification quality. The improvement on F-score keeps competitive throughout different settings of γ, which means the improvement of quality is not sensitive to the value of γ, even at some extreme cases. The GPU-Learning algorithm can work especially well when only a small portion of positive examples are labeled. This suggests the GPU-Learning framework is quite effective on the graph data over a wide range of percentage of labeled positive graphs. We further notice that the value of F-score will slightly increase when the γ increases on all six data sets. This is quite natural since larger number of positive examples provide more useful information on the characteristics of the graphs.

We also tested the effectiveness of three baseline approaches and GPU-Learning approach to the number of selected subgraph features. The number of selected subgraph features varies from 20 to 40. The results for six data sets are presented in Figure 7. The number of selected subgraph features are illustrated on the X-axis, whereas the quality measure F-score is illustrated on the Y-axis. From the figures, we can observe that the classification quality usually improves when the number of selected features increases. That is because larger number of selected features can provide more detailed structure information of graphs to the classifiers. It is clear that the GPU-Learning algorithm is consistently superior to all three baselines in terms of classification quality on various settings of number of selected subgraph features.

E. Efficiency Results

In this section, we present the running times for GPU-Learning and the three different baselines. In addition, we evaluate the efficiency of pruning by using the upper bound for feature quality in *Lemma* 1. In GPU-Learning algorithm, we use the upper bound for feature quality to prune the search spaces of subgraph enumerations. We compare with the same method but without the upper bound pruning. We denote this method as **GPU w/o Pruning**, which uses *gSpan* to obtain a set of frequent subgraphs and then selects the optimal set of subgraphs based on the feature quality measure. We note that *GPU w/o Pruning* generates the same classification results as *GPU-Learning*, since pruning does not change the set of selected subgraph features.

The efficiency results of the different methods with respect to the minimum support on six data sets are reported in Figure 8. The values of minimum support are illustrated on the X-axis, and the running time is illustrated on the Y-axis.



(a) FM (b) MM (c) FR (d) MR (e) MCF-7 (f) MOLT-4

Figure 8. Running Time

It is evident that all algorithms consume more running time when decreasing the minimum support, because they would need to explore larger subgraph search spaces. The running time of NB+SVM-I and Spy+SVM-I are almost identical due to the fact that they use the similar two-step strategy, which makes their curves in the figure overlap. Although GPU-Learning requires multiple searches over the DFS code tree whereas NB+SVM-I and Spy+SVM-I only need once, it is clear that the running time of the GPU-Learning approach is comparable with the baseline schemes because of the pruning technique adopted in the GPU-Learning framework. The running time of the proposed GPU-Learning approach is quite acceptable, considering the tremendous qualitative advantages over the baseline approaches.

We further observe that without pruning, the running time increases exponentially with the decrease of minimum support. This is because the sizes of subgraph search spaces increase exponentially when the minimum support decreases. It is interesting to see that GPU-Learning is at least twice more efficient than *GPU w/o Pruning*, and the running time of GPU-Learning does not increase as much throughout all data sets. The reason is that the upper bound of feature quality can effectively help prune the subgraph search spaces without decreasing the quality of classification. One can see that it is also possible that GPU-Learning consumes almost the same or less running time as the baseline approaches when the minimum support goes very low. Therefore the

pruning power of the proposed method is a clear advantage from the perspective of practical applications.

V. Conclusion

In this paper, we presented a new framework for positive and unlabeled learning on graph classification. While the problem of PU learning has been discussed in the literature, the currently available techniques are not designed for the graph domain. The lack of feature representations of graphs leads to the difficulty on identifying a set of reliable negative examples. In addition, the usefulness of features cannot be obtained when the negative examples are missing from the data set. We address these challenges by first deriving an evaluation criterion to estimate the dependency between features and labels, and then proposing an integrated approach that concurrently updates both graph feature selection and class label assignment. We present experimental results illustrating the proposed integrated framework significantly outperforms the previous methods.

VI. Acknowledgement

This work is supported in part by NSF through grants IIS-0905215, OISE-0968341, and DBI-0960443.

References

[1] H. Cheng, D. Lo, Y. Zhou, X. Wang, and X. Yan, "Identifying bug signatures using discrimative graph mining," in *ISSTA*, Chicago, IL, 2009, pp. 141–152.

[2] N. Jin, C. Young, and W. Wang, "GAIA: graph classification using evolutionary computation," in *SIGMOD*, Indianapolis, IN, 2010, pp. 879–890.

[3] X. Yan, H. Cheng, J. Han, and P. S. Yu, "Mining significant graph patterns by leap search," in *SIGMOD*, Vancouver, BC, 2008, pp. 433–444.

[4] M. Thoma, H. Cheng, A. Gretton, J. Han, H. peter Kriegel, A. Smola, L. Song, P. S. Yu, X. Yan, and K. Borgwardt, "Near-optimal supervised feature selection among frequent subgraphs," in *SDM*, Sparks, NV, 2009, pp. 1075–1086.

[5] X. Yan, P. S. Yu, and J. Han, "Graph indexing: a frequent structure-based approach," in *SIGMOD*, Paris, France, 2004, pp. 335–346.

[6] X. Kong, W. Fan, and P. S. Yu, "Dual active feature and sample selection for graph classification," in *KDD*, San Diego, CA, 2011, pp. 654–662.

[7] H. Cheng, X. Yan, J. Han, and P. S. Yu, "Direct discriminative pattern mining for effective classification," in *ICDE*, Washington, DC, 2008, pp. 169–178.

[8] C. Elkan and K. Noto, "Learning classifiers from only positive and unlabeled data," in *KDD*, Las Vegas, NV, 2008, pp. 213–220.

[9] X. Li, P. S. Yu, B. Liu, and S.-K. Ng, "Positive unlabeled learning for data stream classification," in *SDM*, Sparks, NV, 2009, pp. 1075–1086.

[10] B. Liu, Y. Dai, X. Li, W. S. Lee, and P. S. Yu, "Building text classifiers using positive and unlabeled examples," in *ICDM*, Melbourne, FL, 2003, pp. 179–188.

[11] H. Yu, J. Han, and K. C.-C. Chang, "PEBL: positive example based learning for web page classification using SVM," in *KDD*, Edmonton, AB, 2002, pp. 239–248.

[12] F. Denis, "PAC learning from positive statistical queries," in *ALT*, London, UK, 1998, pp. 112–126.

[13] B. Liu, W. S. Lee, P. S. Yu, and X. Li, "Partially supervised classification of text documents," in *ICML*, San Francisco, CA, 2002, pp. 387–394.

[14] G. P. C. Fung, J. X. Yu, H. Lu, and P. S. Yu, "Text classification without negative examples revisit," *IEEE Transactions on Knowledge and Data Engineering*, vol. 18, pp. 6–20, 2006.

[15] F. Denis, R. Gilleron, and M. Tommasi, "Text classification from positive and unlabeled examples," in *IPMU*, Annecy, France, 2002, pp. 1927–1934.

[16] J. He, Y. Zhang, X. Li, and Y. Wang, "Naive bayes classifier for positive unlabeled learning with uncertainty," in *SDM*, Columbus, OH, 2010, pp. 361–372.

[17] X. Yan and J. Han, "gSpan: Graph-based substructure pattern mining," in *ICDM*, Maebashi City, Japan, 2002, pp. 721–724.

[18] M. Kuramochi and G. Karypis, "Frequent subgraph discovery," in *ICDM*, San Jose, CA, 2001, pp. 313–320.

[19] C. Borgelt and M. Berthold, "Mining molecular fragments: Finding relevant substructures of molecules," in *ICDM*, Maebashi City, Japan, 2002, pp. 211–218.

[20] J. Huan, W. Wang, and J. Prins, "Efficient mining of frequent subgraph in the presence of isomorphism," in *ICDM*, Melbourne, FL, 2003, pp. 549–552.

[21] S. Nijssen and J. Kok, "A quickstart in frequent structure mining can make a difference," in *KDD*, Seattle, WA, 2004, pp. 647–652.

[22] X. Kong and P. S. Yu, "Semi-supervised feature selection for graph classification," in *KDD*, Washington, DC, 2010, pp. 793–802.

[23] Y. Zhang and Z.-H. Zhou, "Multi-label dimensionality reduction via dependence maximization," in *AAAI*, Chicago, IL, 2008, pp. 1503–1505.

[24] X. Kong and P. S. Yu, "Multi-label feature selection for graph classification," in *ICDM*, Sydney, Australia, 2010, pp. 274–183.

[25] A. Gretton, O. Bousquet, A. Smola, and B. Schölkopf, "Measuring statistical dependence with hilbert-schmidt norms," in *ALT*, Singapore, 2005, pp. 63–77.

[26] T.-F. Wu, C.-J. Lin, and R. C. Weng, "Probability estimates for multi-class classification by pairwise coupling," *Journal of Machine Learning Research*, vol. 5, pp. 975–1005, 2004.

[27] T. Kudo, E. Maeda, and Y. Matsumoto, "An application of boosting to graph classification," in *NIPS*, Vancouver, Canada, 2004, pp. 729–736.