2014 IEEE International Conference on Data Mining

Scalable SVM-based Classification in Dynamic Graphs

Yibo Yao and Lawrence Holder
School of Electrical Engineering and Computer Science
Washington State University, Pullman, WA 99164-2752
Email:{yibo.yao, holder}@wsu.edu

***Abstract*—With the emergence of networked data, graph classification has received considerable interest during the past years. Most approaches to graph classification focus on designing effective kernels to compute similarities for static graphs. However, they become computationally intractable in terms of time and space when a graph is presented in a incremental fashion with continuous updates, i.e., insertions of nodes and edges. In this paper, we examine the problem of classification in large-scale and incrementally changing graphs. To this end, a framework combining an incremental Support Vector Machine (SVM) with the Weisfeiler-Lehman (W-L) graph kernel has been proposed to study this problem. By retaining the support vectors from each learning step, the classification model is incrementally updated whenever new changes are made to the subject graph. Furthermore, we design an entropy-based subgraph extraction strategy to select informative neighbor nodes and discard those with less discriminative power, to facilitate an effective classification process. We demonstrate the advantages of our learning techniques by conducting an empirical evaluation on two large-scale real-world graph datasets. The experimental results also validate the benefits of our subgraph extraction method when combined with the incremental learning techniques.**

I. Introduction

In recent years, networked data has gained popularity in the data mining community, due to a great amount of information becoming available in the form of social networks, hyperlinked web documents, chemical compounds, and communication networks. Thus there has been a growing interest in developing algorithmic techniques for performing supervised learning classification tasks on these graph datasets. The aim of graph classification is to learn a discriminative model from the given training graphs and predict the class labels for the testing graphs. The problem of classification in a large-scale graph usually involves classifying nodes [1] or subgraphs [2] into suitable categories. In a binary classification scenario, a node/subgraph is assigned to either a positive class or a negative class. For example, authors in a co-authorship network can be classified into two classes: those who are prolific and those who are not. Similarly, chemical compounds can be classified into two classes: those which are active and those which are not. A detailed investigation of various graph mining and classification algorithms as well as their applications can be found in [3].

Graph classification is itself a challenging task due to the rich and arbitrary structure of graphs. Most graph classification approaches are designed based on kernel machines, which aim to compute similarities between graphs by enumerating their common substructures, e.g., walks [4], subtrees [5], and subgraphs [6]. Conventionally, they assume that the graph data is limited in size and thus can be stored in memory or local storage, which makes it possible to do multiple-scans during the learning process within reasonable time constraints. However, in many real-world applications, graphs are presented in a streaming fashion with the rapid appearance of nodes or edges in the underlying networks. The structures of such networks are highly dynamic and frequently updated over time. For example, social networks (e.g., Facebook), are continuously formed by the increasing social interactions among entities. Due to the dynamic nature of those networks, classical graph kernel methods will be incapable of calculating similarities effectively between graphs for the following reasons.

- With the increasing volume of graph data, it is impossible to hold all information about the underlying network structure in memory or local storage. When the graph is evolving over time, old information regarding nodes or edges must be eliminated from storage in order to maintain a modest speed of accessibility, which makes it infeasible to compute the global kernel matrix for all seen data.
- When the sets of nodes and edges are becoming larger, enumerating the substructures (e.g., paths, subtrees) will result in longer training time. Furthermore, the structures of many real-world networks may constantly evolve over time. As a result, a classification model needs to be effectively updated whenever new structural information becomes available. Unfortunately, traditional graph kernels built on batch mode will not be able to scale well because the resources needed for learning similarity matrices will increase dramatically.
- There is always noisy structural information inside large-scale networks. The existence of these irrelevant features may deteriorate the classification performance or introduce unexpected structural complexity during the learning progress. However, no effective strategy exists to select informative structural information to facilitate the classification progress in a dynamic graph.

In order to address the above challenges, we report in this paper, an incremental graph kernel framework based on Support Vector Machine (SVM) to investigate the problem of classification on large-scale and dynamically changing graphs. The main idea of our incremental classification framework is to construct a classifier based on the data of the current batch and

1550-4786/14 $31.00 © 2014 IEEE
DOI 10.1109/ICDM.2014.69

the support vectors retained from the previous batch, and then use this model to predict the class labels of the future batch. The classification model is incrementally trained upon the arrival of a new batch of data. This methodology has favorable properties regarding time and space issues through summarizing historic data by preserving the support vectors. In case that the entire network cannot be loaded completely into main memory, we adopt a windowing method on the incremental SVM classification algorithm. The sliding window maintains recent data which is available for training and discards all older information from memory. Since the problem of subgraph classification can be transformed into node classification by introducing a virtual node for each subgraph, we focus on node classification in this paper. But our techniques can be easily extend onto subgraph classification. While classifying nodes in a single graph, it is natural to take advantage of the information from neighbor nodes. Therefore, for a target node to be classified, we design an entropy-based scheme to extract a subgraph surrounding this node, in which the informative neighbor nodes are included through discriminative links and the irrelevant ones are filtered out. To the best of our knowledge, this is the first work to leverage graph kernel to classify nodes inside a large-scale dynamic graph. We empirically test our algorithms on two real-world dynamic graph datasets. The experimental results clearly demonstrate the benefits of our entropy-based subgraph extraction strategy, as well as the impressive performance of the proposed incremental learning techniques when compared to state-of-the-art methods.

The rest of this paper is organized as follows: Section II reviews the related work. Some notations and the problem of interest are defined in Section III. In Section IV, the proposed framework including the detailed methods is described. Experimental results are given in Section V. Some conclusions are drawn in Section VI.

II. Related Work

The problem of graph classification has often been studied on static data. There are a large number of effective graph kernels that have been developed for classifying graphs. Most of them follow the same principle of enumerating common substructures to quantify the similarity between graphs. Some typical substructures that have been used to build graph kernels include random walks [4], shortest paths [7], subtrees [5], graphlets [6], and frequent subgraphs [8]. In this paper, our incremental technique is designed based on one of the fastest graph kernels, namely the Weisfeiler-Lehman graph kernel [9].

On the other hand, node classification of graphs has been studied in the context of exploiting the linkage structures to improve classification accuracy [1], [10]. However, only a few works [11]–[14] are related to the problem considered in this paper. In [12], the authors propose a random walk approach combined with the textual content of nodes in the network to improve the robustness and accuracy in classifying nodes in a dynamic content-based network. In [11], a hash-based probabilistic approach is proposed for finding discriminative subgraphs to facilitate the classification on massive graph streams. A 2-dimensional hashing scheme has been designed to compress and summarize the continuously presented edge streams, and explore the relation between edge pattern co-occurrence and class label distributions. Hashing techniques have also been used in [13] to classify graph streams by detecting discriminative cliques and mapping them onto a fixed-size common feature space. The authors in [14] use their presented Nested Subtree Hash Kernel (NSHK) algorithm to project different subtree patterns from graph streams onto a set of common low-dimensional feature spaces, and construct an ensemble of NSHKs for large-scale graph classification over streams. The approaches proposed in [13], [14] are closely related to the problem considered in this paper. Both of their methods aim to find common feature (subtree or clique) patterns across the data stream and map those increasing patterns onto a lower dimensional feature space using random hashing techniques. However, those two methods are likely to lose some discriminative substructure patterns by compressing the expanding feature patterns into a fixed-size feature space during the hashing process. Meanwhile, they are not directly applicable to a single large-scale dynamic graph without a subgraph extraction process.

The framework presented in this paper addresses the novel task to learn a classification model incrementally on a large-scale and time-evolving graph. We adopt an incremental learning strategy to incorporate the discriminative substructure patterns (i.e., the support vectors) from previous batches into the current training batch, instead of projecting them onto a lower-dimensional space. Additionally, the proposed subgraph extraction strategy helps to improve the classification performance by filtering out irrelevant information. Our methods achieve impressive performance in terms of classification effectiveness and learning efficiency as well as avoiding the limitations discussed in Section I.

III. Preliminaries

We first introduce some important notations and definitions. In this paper, we focus on the dynamic graphs which are subject to incremental changes, i.e., continuous insertions of new nodes and edges. Other types of changes, e.g., deletions of nodes or edges, modifications of attributes, are beyond the scope of this paper. For the rest of this paper, we use network and graph interchangeably.

Definition 1: A **labeled graph** is denoted as $G = (\mathcal{V}, \mathcal{E}, \mathcal{L})$, where $\mathcal{V}$ is a set of nodes $\mathcal{V} = \{v_1, \ldots, v_{|\mathcal{V}|}\}$, $\mathcal{E} \subseteq \mathcal{V} \times \mathcal{V}$ is a set of directed/undirected edges, and $\mathcal{L}$ is a set of labels that can be assigned to nodes or edges to indicate their unique identifiers or attributes.

Definition 2: Let $G = (\mathcal{V}, \mathcal{E}, \mathcal{L})$ and $G' = (\mathcal{V}', \mathcal{E}', \mathcal{L}')$ denote two labeled graphs. G' is said to be a **subgraph** of G, i.e., $G' \subseteq G$, if and only if $(1)\mathcal{V}' \subseteq \mathcal{V}$,$(2)\mathcal{E}' \subseteq \mathcal{E}$, and $(3)\mathcal{L}' \subseteq \mathcal{L}$.

Definition 3: An **update**, denoted by $\mathcal{U}$, is an operation that inserts nodes or edges into the underlying graph.

Definition 4: A **incremental dynamic graph** $\mathcal{G}$ is defined over a sequence of updates $\{\ldots, \mathcal{U}_i, \mathcal{U}_{i+1}, \ldots\}$, where each $\mathcal{U}_i$ contains a specified operation that is to be applied to the object graph in a streaming fashion.

We assume that each update operation $\mathcal{U}_i$ can be represented in the form of an edge $\mathcal{E}_i =< v_{i_1}, v_{i_2} >$, where v_{i_1}, v_{i_2} denote the two endpoints. If the underlying dynamic graph is a directed graph, the edge $\mathcal{E}_i$ has a direction pointing from v_{i_1}

to v_{i_2}. Applying $\mathcal{E}_i$ to the current dynamic graph $\mathcal{G}$ will result in the following cases:

- *case 1:* insertion of a new node, if $v_{i_1} \notin \mathcal{G}$ and $v_{i_2} = \emptyset$.
- *case 2:* insertions of a new node and a new edge between this new node and an old node, if $v_{i_1} \notin \mathcal{G}$ and $v_{i_2} \in \mathcal{G}$, or vice versa.
- *case 3:* insertion of a new edge between two old nodes, if $v_{i_1}, v_{i_2} \in \mathcal{G}$.
- *case 4:* insertions of two new nodes and a new edge between these two nodes, if $v_{i_1}, v_{i_2} \notin \mathcal{G}$.

In this paper, we assume that the updates are received in the form of batches $\{B_t\}_{t=1}^{\infty}$, and these batches contain various numbers of insertion operations. By using the batch notation, an incremental dynamic graph can also be denoted as a collection of batches $\mathcal{G} = \{B_1, \ldots, B_t, \ldots\}$. See Fig.1 for an example.

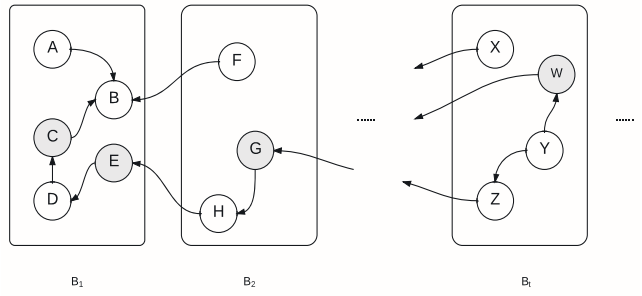


Fig. 1. An example of a dynamic graph. Updates are received in the form of batches. For example, when B_2 comes in, *case 2* happens since two new nodes F, H are connected to two old nodes B, E; *case 4* also happens because a new edge connecting two new nodes H, G is inserted.

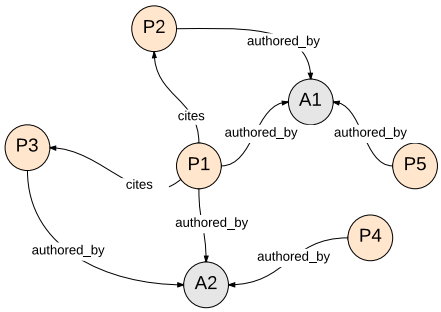


Fig. 2. An instance of a citation network. P_i represents a paper of interest while A_i represents an author associated to a paper. An edge label represents the relationship between the two nodes it connects.

Definition 5: We define a node to be classified as a **central node** which denotes a **central entity** from the original data, and a node as a **side node** if it is not a central one. Furthermore, we assume that each side node must be connected only to central nodes in our graph representation. In other words, we do not allow edges between any two side nodes in the graph representation in this paper.

Fig.2 shows one instance of a citation network in which all papers of interest are marked as central nodes while authors as side nodes. Such a network would support the classification of papers, e.g., papers that will receive a high number of citations versus papers that will not.

Given the aforementioned setting, we now formulate the problem that we aim to tackle in this paper as follows: Given a dynamic graph with central and side nodes indicated in its representation, and each central node v_i has an associated class label $y_i \in \{+1, -1\}$, the goal is to learn a classifier using the available information up until the current time t, i.e., $\cup_{i=1}^{t} B_i$, and to predict the class labels of new central nodes from the next batch B_{t+1}.

A. Weisfeiler-Lehman Graph Kernel

Graph kernels have been designed to express the similarity between graphs by mapping graph structures into a high-dimensional Hilbert space and then calculating their inner product through kernel functions. The Weisfeiler-Lehman (W-L) graph kernel [9] is a state-of-the-art graph kernel whose runtime scales linearly in the number of edges of the graphs. It is based on the one-dimensional variant of the Weisfeiler-Lehman isomorphism test [15]. In this algorithm, each node's label is updated by incorporating a sorted set consisting of its neighbor nodes' labels for a certain number of iterations. However, the W-L kernel based on batch mode will suffer from memory and time issues as described in Section I when applied to dynamic graphs since multiple-scans and holding all data in memory are not realistic in this situation. The kernel matrix has to be recomputed whenever a new batch of central nodes or subgraphs arrives in order to perform classification. As a result, the computational complexity grows dramatically as new batches of data continuously stream in.

B. Incremental Support Vector Machines

SVMs have been successfully applied as classification tools in a variety of domains. The data points which specify the separating hyperplane, namely, support vectors, convey more information for classification than those that lie far away from the hyperplane. This fact indicates a compression scheme for saving more space as well as an incremental learning technique for classification on large datasets [16]. In order to make SVMs suitable for learning with data streams, the support vectors which define the decision boundary are retained as a representation of the data seen so far and treated as part of the training data [17].

IV. Framework

In this section, we introduce the three key components included in our framework in detail: (1) extracting subgraph for a central node from a large dynamic network, (2) retaining the support vectors from past data for incremental learning, (3) sliding a window on the data stream to keep data stored in memory at a moderate size. When a new batch B_t arrives, our overall framework works as follows:

1) A subgraph extraction procedure is called to extract subgraphs for the central nodes in B_t.

2) If a sliding window is specified, then delete the old information which is outside the current window. Go directly to the next step if no window is set.
3) Combine the support vectors of the classification model learned from B_{t-1} with the subgraphs from B_t as a new training set. And train a new model M_t on this set to predict the class labels of the central nodes from B_{t+1}.

In the following subsections, we first propose a subgraph extraction scheme for central nodes, then describe the incremental classification method and the window-based incremental method.

A. Subgraph Extraction for Central Entities

For classifying nodes in a graph, it is always desirable to use the linkage structures that encode relationships between nodes. Therefore, it becomes natural to assign class labels to the nodes by measuring the similarities between subgraphs surrounding those nodes. There are many approaches for extracting a subgraph surrounding a node, e.g., 1-edge hops, random walks. However, a star-like subgraph extracted using 1-edge hops may not be discriminative enough since it contains less structural information. On the other hand, a subgraph extracted by random walks may make the classification process more complicated.

In order to classify a central node in a dynamic graph, we design an effective strategy to extract a subgraph for it by selecting the informative neighbor nodes and discarding those with less discriminative power. For a node $v_i \in \mathcal{G}$[1], if it is connected to any central nodes, then we can define the entropy value for v_i. Let $\mathcal{N}(v_i)$ be the neighbor nodes of v_i, and let n_{pos} and n_{neg} denote the numbers of central nodes with positive class labels and negative class labels in $\mathcal{N}(v_i)$ respectively. The probabilities of positive and negative instances in $\mathcal{N}(v_i)$ can then be estimated as follows:

$$p_1 = \frac{n_{pos}}{n_{pos} + n_{neg}} \text{ and } p_2 = \frac{n_{neg}}{n_{pos} + n_{neg}} \quad (1)$$

Thus the entropy computation for v_i can be explicitly written as:

$$E(v_i) = -p_1 \log_2 p_1 - p_2 \log_2 p_2 \quad (2)$$

The entropy value of v_i expresses the discriminative power of v_i with respect to the two classes (positive and negative) during the classification process. The lower $E(v_i)$ is, the more power v_i has.

To obtain a subgraph for a target central node to be classified, denoted as v_c, a threshold parameter θ needs to be set for selecting the discriminative neighbor nodes. The main idea of our extraction method is that we start from v_c and keep extracting neighbor nodes whose entropy values $\leq \theta$ until we meet other central nodes of the same type as v_c. We then induce a subgraph from the whole large graph, in which we include all the interconnections between the extracted nodes. It is worth noting that when computing entropy values for each node in $\mathcal{N}(v_c)$, the class label of v_c should not be counted because v_c is the target node to be classified. And if a node $v_s \in \mathcal{N}(v_c)$ has no other connections to central nodes except v_c (this is the case that v_s is attached to the subject graph through v_c), then this node will also be included in extraction. The subgraph extraction scheme aims to serve two purposes: (1) select informative neighbor nodes for classification, and (2) reduce the structural complexity of the subgraph to facilitate the computation of the kernel matrix in the learning steps. This method allows us to utilize the discriminative information included in these neighbor nodes, and has been proven to perform effectively in our experiments. Algorithm 1 shows the detailed procedure for extracting a subgraph surrounding a central node from a graph.

[1]Note that v_i can be central or side node.

Algorithm 1 Subgraph Extraction (SubExtract)

Input:
$\mathcal{G}$: A graph
v_c: A target central node
θ: A threshold for selecting discriminative nodes
Output:
Sub_{v_c}: A subgraph surrounding v_c

```
1: I(v_c) = {v_c}
2: N_v = N(v)
3: while N_v ≠ ∅ do
4:    pop a node v' from N_v and compute its entropy E(v')
5:    if v' is not visited and E(v') ≤ θ then
6:       I(v_c) = I(v_c) ∪ {v'}
7:       mark v' as visited in G
8:       if v' is not the same type as v_c then
9:          N_v = N_v ∪ N(v')
10:      end if
11:   end if
12: end while
13: induce a subgraph Subg_{v_c} from G with the nodes in I(v_c)
14: return Subg_{v_c}
```

When dealing with a dynamic graph, the subgraph extraction becomes complicated. In order to control the size of the dynamic graph stored in memory, we propose to use a sliding window to retain the most recent data batches and discard the oldest ones (see the following subsection IV-C). There is a possibility that a certain update in current batch B_t may refer to old nodes which have been inserted at batches previous to B_t (i.e., *case 2* or *case 3* mentioned in Section III occurs). We then check the existence of these old nodes. If they have been removed from memory, we will ignore that update. This is reasonable because the target concept is likely to change in data streams. And we want to classify the future data using the latest model constructed base on recent batches.

B. Incremental Classifier

To perform classification tasks on large dynamically changing graphs, the major challenge is to learn a model on currently available data and incrementally update the model upon the arrival of new batches of streaming data. Typically, learning a classifier based on all historic data seen so far will help reduce the generalization error. However, the learning process itself will become computationally intractable in terms of memory and CPU time when the amount of available data tends to be huge. Moreover, the graph data is constantly generated at a

rapid rate, which makes it impossible to store all data in main memory or scan the data multiple times.

Incremental learning techniques address those issues by condensing the historic information and combining the condensed information with the current batch of data for training purpose. This approach has shown good performance when incorporated into the SVM learning process [16], [17]. At each learning step, the support vectors from the previous batch are retained as a compact summary of the past data, and they are combined with the current batch to comprise a new training set for the next step. This scheme allows us to throw away old examples that play a negligible role in specifying the decision boundary in classification.

When dealing with a dynamic graph, we consider every batch B_t and the support vectors retained from the model learned based on the previous batch B_{t-1} as the current training set, and build a SVM classification model using the W-L kernel. It is possible that the set of support vectors retained from B_{t-1} may include some support vectors from the batches previous to B_{t-1}. This is because these support vectors are still identified as important examples when defining the classification decision boundary on B_{t-1}. The new model is then used to predict the class labels of those central nodes or subgraphs in batch B_{t+1}. In this setting, we enable the classification to be updated incrementally when new batches of data stream in continuously at a rapid rate. Algorithm 2 shows the pseudocode of the incremental SVM (IncSVM) method.

Algorithm 2 Incremental SVM (IncSVM)

Input:
$\mathcal{G}$: A graph
SV_{t-1}: A set of support vectors
B_t: The current batch
θ: The threshold for selecting neighbor nodes
Output:
M_t: A SVM classification model for prediction

1: $Sub_{B_t} = \emptyset$
2: **for** each central node v_c in B_t **do**
3: $Sub_{B_t} = Sub_{B_t} \cup \{\text{SubExtract}(\mathcal{G}, v_c, \theta)\}$
4: **end for**
5: construct a training set $TR_t = SV_{t-1} \cup Sub_{B_t}$
6: learn a classifier M_t on TR_t using W-L kernel
7: **return** M_t (including its support vectors SV_t)

C. Window-based Incremental Classifier

Although the incremental SVM method will potentially utilize less memory by discarding old data points which are not identified as support vectors, it is still possible that the algorithm will not scale up effectively when: 1) A huge amount of support vectors tend to be retained. For example, the learning problem is a hard one where almost all training data points are on the decision boundary of the constructed SVM; 2) The large dynamic graph itself is continuously changing with increasing volume of nodes and edges, which needs more memory. Furthermore, the target concept of the data may change with time so that the old examples may not be a promising predictor for future data. Besides retaining support vectors, some other strategies are needed to remove old information which is not valuable in deciding the separating hyperplane.

We present another incremental SVM learner with a sliding window which maintains a limited amount of data in memory. Using a sliding window enables us to keep a large dynamic network covering the nodes and edges that arrive recently. When the window is full, the oldest batch of nodes or edges will be removed from the network so that we can maintain the network as a moderate size in memory.

The important parameter for this method is the size of the window W which stores the recent W batches of data. At time t when the new batch B_t arrives, if the window is full, the old nodes and edges inserted at batch B_{t-W} will be deleted from the underlying network. As a result, the support vectors form B_{t-W} are also removed from the retained support vector list SV_{t-1}. However, in the central node classification scenario, it is possible that some support vectors remaining in SV_{t-1} may connect to nodes from B_{t-W}, especially the central nodes of support vectors from B_{t-W}. In that case, we modify these support vectors by deleting all the old nodes which come from B_{t-W} and their connections. Take Fig.2 as a toy example, a support vector in a citation network is a subgraph surrounding a central paper node P_1 and it contains another central node P_4 which is an old support vector from B_{t-W}. We will retain this subgraph of P_1 as a support vector by deleting P_4 and the edges related to P_4. And the subgraph of P_1 will be eliminated from the support vector list in the next learning step if it has no discriminative power any more. Our window-based incremental SVM (WinSVM) is described in Algorithm 3.

Algorithm 3 Window-based IncSVM (WinSVM)

Input:
$\mathcal{G}$: A graph
SV_{t-1}: A set of support vectors
B_t: The current batch
θ: The threshold for extracting subgraphs
W: Window size
Output:
M_t: A SVM classification model for prediction

1: **if** the window is full **then**
2: delete the batch B_{t-W} from $\mathcal{G}$
3: check SV_{t-1} and **do**
4: a. delete the support vectors from B_{t-W}
5: b. modify the support vectors which contain nodes from B_{t-W}
6: **end if**
7: **return** IncSVM($\mathcal{G}$,SV_{t-1},B_t, θ)

D. Complexity Analysis

In order to analyze the time complexity of our proposed methods, we assume that each batch has the same number of central nodes, denoted by $|B|$, to be streamed into the underlying large dynamic network. Furthermore, we assume the subgraph extracted for each central node in a batch has no more than $|\mathcal{V}|$ nodes and $|\mathcal{E}|$ edges. Then the computational

complexity for the W-L graph kernel is $O(r|B||\mathcal{E}|+r|B|^2|\mathcal{V}|)$ [9] where r is the number of iterations used in W-L isomorphism test.

For IncSVM, the worst case is that the set of support vectors retained from previous batch contains all the training examples seen so far. Therefore, at time t, we have $t|B|$ subgraphs (consisting of the previous $t-1$ batches and the current batch), which indicate that the complexity for IncSVM is $O(rt|B||\mathcal{E}|+rt^2|B|^2|\mathcal{V}|)$. Similar results can be drawn for WinSVM. The worst situation is that it retains all training examples from all previous batches in a window as support vectors, which causes $O(rW|B||\mathcal{E}|+rW^2|B|^2|\mathcal{V}|)$ (W is the size of a sliding window) time to compute a kernel matrix.

V. Experiments

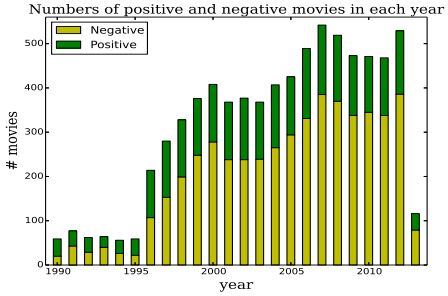
In this section, we evaluate the effectiveness of the entropy-based subgraph extraction method and test the proposed incremental learning techniques on two real-world dynamic graph datasets.

A. Benchmark Data

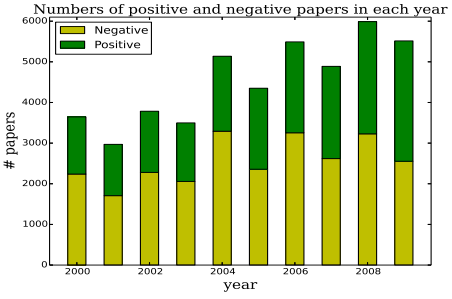
The following datasets are used in our experimental validation.

(1) IMDB Network: The Internet Movie Database (IMDB)[2] consists of data related to movies. Each movie in IMDB is associated with rich information such as actors, directors, budget, box-office receipts, etc. In our experiments, we focus on movies released in the United States from 1990 to 2013 and build a dynamic movie network by extracting its associated actors, directors, writers and producers for each movie. The IDs of movies, actors, directors, writers and producers are represented as nodes and the relationships between them are represented by various types of edges. More specifically, in our representation, we denote that (1) each movie ID is a central node; (2) all other types of nodes are side nodes; (3) if a movie M_1 is directed by an director D_1, there is a directed edge with label *directed-by* from M_1 to D_1; (4) if a movie M_1 is acted by an actor A_1, there is a directed edge with label *acted-by* from M_1 to A_1; (5) if a movie M_1 is written by a writer W_1, there is a directed edge with label *written-by* from M_1 to W_1; (6) if a movie M_1 is produced by a producer P_1, there is a directed edge with label *produced-by* from M_1 to P_1. We have identified 7535 movies released between 1990 and 2013. The dynamic movie network is formed by continuous insertions of aforementioned nodes and edges chronologically when new movies appear along with their actors, directors, writers and producers. The final network contains over 2.0×10^5 nodes and 3.4×10^5 edges. Our goal is to predict whether a movie will be successful (the opening weekend box-office receipts exceed $2 million) [18] when it is released. Among all the identified movies, 2524 of them (whose opening weekend revenue $\geq$ $2 million) are labeled as positive while the others are labeled as negative. Fig.3(a) shows the numbers of positive and negative movies released in each year between 1990 and 2013.

(2) DBLP Network: DBLP[3] is a database containing millions of publications in computer science. Each paper is associated with abstract, authors, year, venue, title and references. Similar to the work in [19], our classification task is to predict which of the following two fields a paper belongs to: DBDM (database and data mining: VLDB, SIGMOD, PODS, ICDE, EDBT, SIGKDD, ICDM, DASFAA, SSDBM, CIKM, PAKDD, PKDD, SDM and DEXA) and CVPR (computer vision and pattern recognition: CVPR, ICCV, ICIP, ICPR, ECCV, ICME and ACM-MM). However, we could not recreate the complete settings as described in [19], which used 19,456 papers and 1,000 most frequent keywords to represent the graphs. We instead have identified 45,270 papers published between 2000 and 2009, and their references and authors. 19,680 of them are DBDM-related (positive) while 25,590 of them are CVPR-related (negative). The dynamic DBLP network is then formed by insertions of papers and authors and the relationships between these entities. In particular, we denote that (1) each paper ID is a central node while each author ID is a side node; (2) if a paper P_1 cites another paper P_2, there is a directed edge labeled with *cites* from P_1 to P_2; (3) if a paper P_1's author is A_1, there is a directed edge labeled with *written-by* from P_1 to A_1. The final graph contains about 1.2×10^5 nodes and 2.5×10^5 edges. Fig.3(b) shows the numbers of positive and negative papers published in each year between 2000 and 2009.



(a) IMDB



(b) DBLP

Fig. 3. The numbers of positive and negative examples in each year for IMDB and DBLP datasets.

B. Experimental Settings

Baseline Methods: To evaluate the classification performance of our learning framework, we compare the proposed methods with the following baseline methods.

- **Nested Subtree Hash Kernel (NSHK)** [14] is an ensemble learning framework which consists of W

[2]http://www.imdb.com

[3]http://arnetminer.org/citation

weighted classifiers built on the most recent W batches of data

$$f_E(\mathbf{x}) = \sum_{i=t-W+1}^{t} w_i f_i(\mathbf{x})$$

where f_i is a classification model learned from batch B_i, $w_i = \sum_{y\in\{\pm\}} P_t(y)(1 - P_t(y))^2 - \frac{1}{|B_t|}\sum_{n=1}^{|B_t|}(0.5(1 - y_n^t f_i(x_n^t)))^2$ is the weight for f_i measured by the mean square errors related to f_i and the class distribution, and $P_t(y)$ denotes the class distribution in B_t. Each classifier of the ensemble is constructed using W-L kernel. In our experiments, we fix the ensemble size as 10 and set the dimensionality for 5 resolutions as $\{500, 1000, 5000, 10000, 50000\}$.

- **Discriminative Clique Hashing (DICH)** [13] uses random hashing technique to compress infinite edge space onto a fixed-size one and applies a fast clique detection algorithm to detect frequent discriminative cliques as features from a graph stream, and constructs a simple classifier based on the detected cliques. We run DICH using the following parameters: frequent clique threshold $= \{0.01, 0.05, 0.1\}$, discriminative clique threshold $\{0.5, 0.6, 0.7\}$, and size of compressed edge set $= \{5000, 10000, 20000\}$. The experimental results of DICH are reported using one of these parameters' combinations with the highest classification accuracy.

All SVM-based classification tasks (including NSHK, IncSVM, WinSVM) are conducted using LIBSVM [20], as long as we compute the kernel matrices. To make a fair comparison, we set the maximum number of iterations for W-L isomorphism test in IncSVM and WinSVM as 5 (Note that this value should be consistent with the number of resolutions in NSHK). Unless specified otherwise, we use the following settings for the parameters of these methods: batch size $|B_t| = \{200, 400\}$ for IMDB and $\{400, 600\}$ for DBLP, subgraph extraction threshold $\theta = \{0.2, 0.4, 0.6, 0.8, 1.0\}$, and window size $W = \{6, 8, 10\}$. All experiments are conducted on a 1.40 GHz single core AMD Opteron machine with 132 GB memory size, running CentOS 6.4.

C. Performance Evaluation

We evaluate the classification effectiveness and efficiency with experiments on the two real-world datasets for each method. The effectiveness is measured by prediction accuracy, which is the proportion of correctly classified examples to the total number of examples in each batch. The efficiency is measured by recording accumulated system runtime (summation of training time and prediction time for each batch). The classification performance of our incremental learning framework is first compared to that of the baseline methods. Then we study the impact of varying the threshold θ for subgraph extraction and the window size for WinSVM.

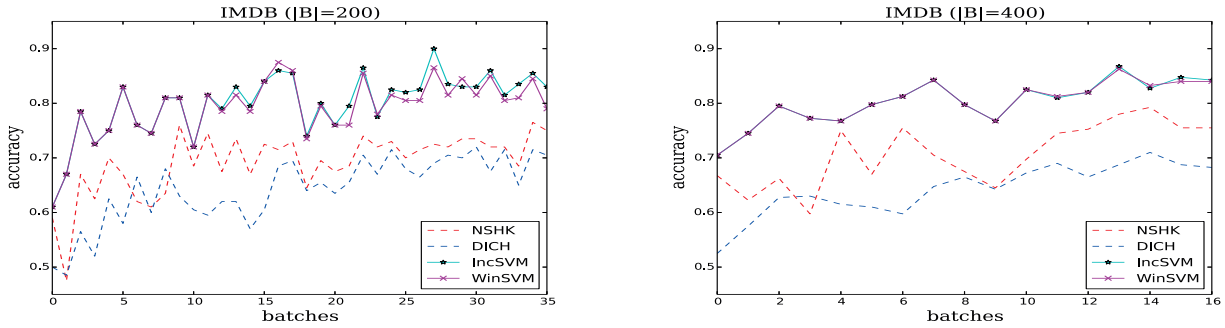
1) Classification Performance on Dynamic Graphs: Since NSHK and DICH are designed for classifying a graph stream which contains a sequence of independent graphs instead of learning from a single large-scale dynamic graph, we need to extract subgraphs from a dynamic graph for these two algorithms. In order to make a fair comparison, we extract the subgraphs for central entities by setting $\theta = 1.0$, which indicates a naive extraction where the subgraphs are generated by including all neighbor nodes for each entity. In Fig.4 and Fig.5 we report the classification accuracy at different learning steps on IMDB and DBLP. We vary the number of subgraphs in each batch and fix the window size $W = 10$ for WinSVM[4]. IncSVM does not scale well on DBLP in this experiment because almost all training examples tend to be support vectors, so we exclude its performance report on DBLP. We can clearly see that our incremental learning techniques constantly outperform their peers across all batches of both datasets. These results indicate that, by retaining the support vectors from previous batches, it is possible to learn a classifier on dynamic graphs which can achieve higher accuracy compared to state-of-the-art algorithms. The average accuracy values across all batches for both IMDB and DBLP shown in Fig.6 demonstrate this fact. The average accuracy values are insensitive to $|B|$, which implies the stability of our techniques. Another observation is that WinSVM can generate impressive accuracy comparable to that of IncSVM on IMDB, which indicates that we can obtain good accuracy results by setting a proper window size and discarding all old information outside the window. Overall, the incremental framework proposed in this paper is able to potentially reduce the number of training examples by compressing the historic data to facilitate a learning process, and maintain or improve classification performance.

Moreover, we record the accumulated system runtime[5] across all batches for running these algorithms on IMDB and DBLP. The results are shown in Fig.7. We find that DICH takes much less time than NSHK and our incremental techniques. This is mainly because DICH does not involve kernel matrix computation, which is actually the most time-consuming part for NSHK and IncSVM/WinSVM. However, IncSVM and WinSVM take less time than NSHK on IMDB. For IMDB, the accumulated learning time of IncSVM is only $28.34\%(|B| = 200)$ and $15.02\%(|B| = 400)$ of that of NSHK, while the accumulated learning time of WinSVM is only $12.60\%(|B| = 200)$ and $12.56\%(|B| = 400)$ of that of NSHK. Considering the higher accuracy that IncSVM and WinSVM achieve, we conclude that our incremental learning framework has the best classification performance on IMDB. On the other hand, WinSVM enables a runtime reduction by $29.39\%(|B| = 400)$ and $19.94\%(|B| = 600)$ of that of NSHK on DBLP. Given the fact that WinSVM has higher accuracy than both NSHK and DICH, we may conclude that WinSVM with a suitable window size is a promising classification method. Overall, the proposed learning framework is superior to the peers.

2) Impact of Subgraph Extraction: To investigate the impact of our entropy-based subgraph extraction scheme on performance, we set $|B| = 200$ for IMDB and $|B| = 600$ for DBLP, and report average classification accuracy and system runtime for the four algorithms by using different thresholds θ for extracting subgraphs from IMDB and DBLP. Fig.8 and Fig.9 plot the average classification accuracy results across all

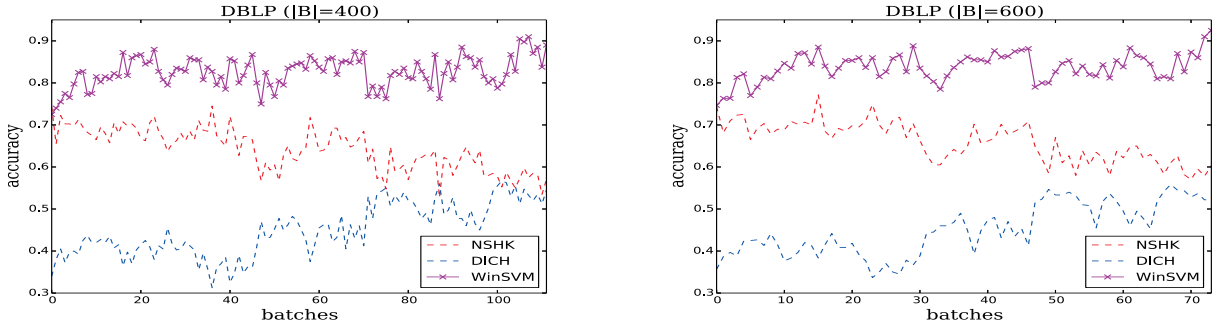
[4] W is set to 10 in order to be consistent with NSHK, whose training examples are the most recent 10 batches.

[5] Note that the time plots in Fig.7,8,9 are in 10-logarithmic scale.



(a) (b)

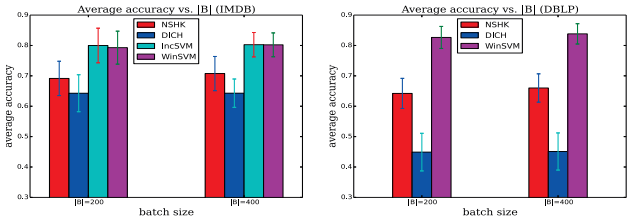
Fig. 4. Accuracy at each learning step w.r.t. different batch sizes on IMDB, with subgraph extraction threshold $\theta = 1.0$ and window size $W = 10$ for WinSVM. The number of subgraphs in each batch: (a)200 and (b)400.



(a) (b)

Fig. 5. Accuracy at each learning step w.r.t. different batch sizes on DBLP, with subgraph extraction threshold $\theta = 1.0$ and window size $W = 10$ for WinSVM. The number of subgraphs in each batch: (a)400 and (b)600.

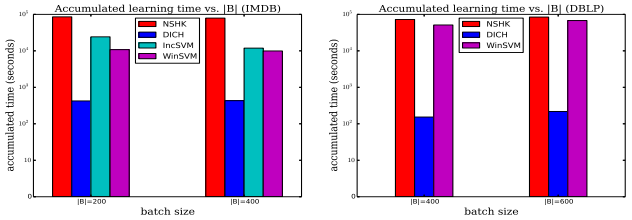
batches and accumulated system runtime w.r.t. θ on IMDB and DBLP. We observe from Fig.8(a) that the classification accuracy is sensitive to θ on IMDB. There is a performance improvement as θ increases from 0.2 to 0.8 and a deterioration from 0.8 to 1.0 for NSHK, IncSVM and WinSVM, which demonstrates that our subgraph extraction method is able to improve classification accuracy through discovering informative neighbor information during the learning process for graph kernel based learning algorithms. However, DICH is also influenced by its own parameters (frequent clique threshold and discriminative clique threshold [13]), so the results of DICH do not display the same phenomenon as other three do. Fig.8(b) shows that, our subgraph extraction method can reduce accumulated learning time for all the four algorithms by setting $\theta < 1.0$. Specifically, IncSVM gains an accuracy improvement by 5.23% and a runtime reduction by 35.16% from $\theta = 1.0$ to $\theta = 0.8$, while WinSVM gains an accuracy improvement by 5.15% and a runtime reduction by 48.04% from $\theta = 1.0$ to $\theta = 0.8$. On DBLP, the accuracy values and system runtime are more steady w.r.t. θ and there is no clear accuracy improvement or deterioration like what have been observed on IMDB. This is probably due to the fact that the discriminative nodes are extremely rare in the DBLP dynamic network. For all these three methods, the average classification accuracy values of $\theta = 0.8$ are approximately equivalent to those of $\theta = 1.0$, respectively. And they all gain slight reductions in terms of runtime. Specifically, the accumulated learning time of WinSVM gains a reduction by 8.39% from $\theta = 1.0$ to $\theta = 0.8$. These experimental results demonstrate clear benefits of our entropy-based subgraph extraction method in terms of classification effectiveness and efficiency.



(a) IMDB (b) DBLP

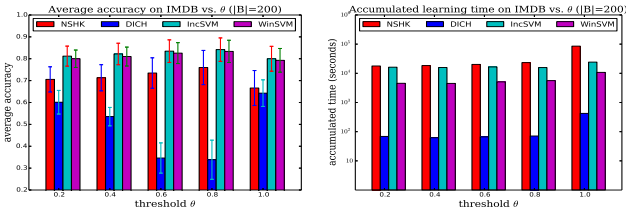
Fig. 6. Average accuracy across all batches w.r.t. different batch sizes on IMDB and DBLP.

Another aspect for studying the impact of using different θ is to investigate the structural complexity of the extracted subgraphs. We use the average numbers of nodes and edges for subgraphs at each batch to measure the structural complexity, which also reflect the computational cost for the proposed incremental framework (see Section IV-D). From Fig.10, we can find that, setting $\theta < 1.0$ for IMDB will reduce the structural complexity of extracted subgraphs to a great extent. The lower θ is, the lower structural complexity is. Combining the results from Fig.8 and Fig.10, we can see that our entropy-based subgraph extraction strategy is capable of retaining



(a) IMDB (b) DBLP

Fig. 7. Accumulated system runtime across all batches w.r.t. different batch sizes on IMDB and DBLP.

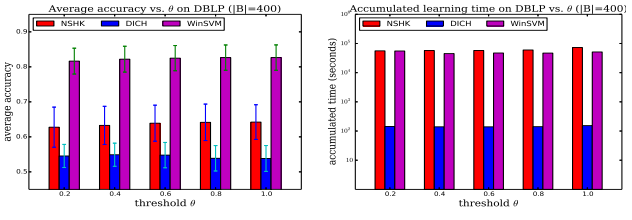


(a) Average accuracy (b) Accumulated learning time

Fig. 8. Average accuracy across all batches and accumulated learning time on IMDB w.r.t. different values of θ.

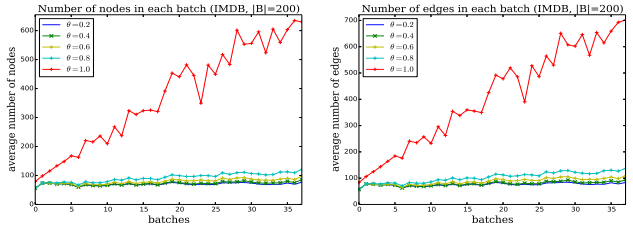
informative neighbor nodes and filtering irrelevant ones while inducing subgraphs for central entities in IMDB. Fig.11 shows the structural complexity of the extracted subgraphs at each batch on DBLP. We find that the structural complexity is insensitive to θ, which indicates that fewer neighbor nodes tend to be filtered out during the subgraph extraction process. Thus the classification performance on DBLP remains relatively steady w.r.t. different values of θ, which is reflected in Fig.9.

3) Impact of Window Size: In the last part, we investigate the classification performance w.r.t. different window sizes for one of the proposed learning techniques, namely WinSVM. According to the results that we have obtained from the previous subsections, we set subgraph extraction threshold $\theta = 0.8$ and vary the window size $W = \{6, 8, 10\}$. We report the average classification accuracy across all batches in Table I. Intuitively, the classification accuracy will increase at the expense of more space caused by enlarging the window size. This is mainly because more training examples will be retained inside a larger window, which will reduce the generalization error of the classifier. On the other hand, enlarging a window size will increase runtime because it will consume more time for WinSVM to train a classifier. Fig.12 demonstrates this



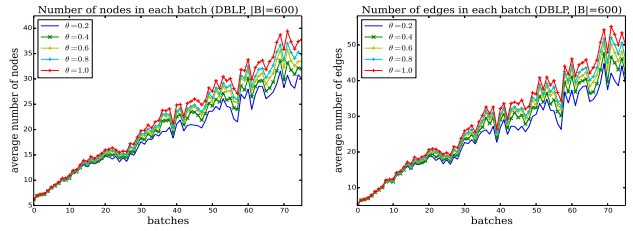
(a) Average accuracy (b) Accumulated learning time

Fig. 9. Average accuracy across all batches and accumulated learning time on DBLP w.r.t. different values of θ.



(a) Number of nodes (b) Number of edges

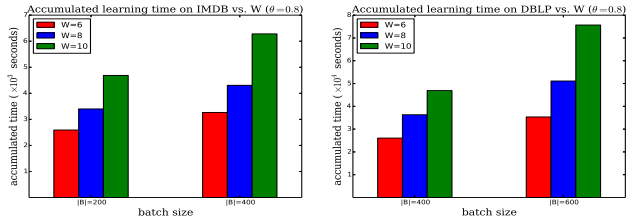
Fig. 10. Average numbers of nodes and edges at each batch on IMDB w.r.t. different values of θ. The number of subgraphs in each batch: $|B| = 200$.



(a) Number of nodes (b) Number of edges

Fig. 11. Average numbers of nodes and edges at each batch on DBLP w.r.t. different values of θ. The number of subgraphs in each batch:$|B| = 600$.

fact. Overall, the results show that WinSVM can achieve impressive classification performance with a proper window size. Therefore, WinSVM is a promising technique and a potential alternative of IncSVM for doing classification on large-scale dynamic graphs with favorable properties in terms of processing time and memory.



(a) IMDB (b) DBLP

Fig. 12. Accumulated system runtime across all batches w.r.t. different window sizes for WinSVM on IMDB and DBLP.

D. Limitations

Overall, our experimental findings show the advantage of the proposed incremental techniques compared to the two baseline methods in terms of classification effectiveness and efficiency. However, as we mentioned in Section V-C, IncSVM will fail to scale well on those datasets in which almost all the training examples are support vectors. In this case, the incremental learner tends to retain all the training examples from the previous batches and make kernel computation infeasible. WinSVM addresses the scalability issue by discarding all old support vectors which are outside the window. Although this may reduce the accuracy in some cases, WinSVM is able to produce results comparable to the competing approaches in terms of accuracy and runtime.

TABLE I. AVERAGE ACCURACY ACROSS ALL BATCHES USING DIFFERENT WINDOW SIZES FOR WINSVM ($\theta = 0.8$).

IMDB			DBLP		
W	$\|B\| = 200$	$\|B\| = 400$	W	$\|B\| = 400$	$\|B\| = 600$
6	0.828 ± 0.048	0.837 ± 0.040	6	0.807 ± 0.036	0.824 ± 0.034
8	0.831 ± 0.050	$\mathbf{0.839 \pm 0.041}$	8	0.819 ± 0.036	0.832 ± 0.033
10	$\mathbf{0.834 \pm 0.051}$	$\mathbf{0.839 \pm 0.041}$	10	$\mathbf{0.826 \pm 0.036}$	$\mathbf{0.837 \pm 0.034}$

VI. CONCLUSION

In this paper, we present a novel framework for studying the problem of classification on large-scale dynamic graphs. Our techniques combine an incremental SVM and a fast Weisfeiler-Lehman graph kernel to train a classification model and constantly update it by preserving the support vectors at each learning step. Additionally, a sliding window strategy is incorporated into our framework in order to further reduce memory usage and learning time. The entropy-based subgraph extraction method is designed to discover informative neighbor information and discard irrelevant information when inducing a subgraph for a central entity to be classified. We validate the proposed methods for effective classification in large-scale dynamic graphs.

Our future work will include investigating the pros and cons of our incremental methods by conducting comparisons with state-of-the-art classification algorithms on more real-world dynamic networks. We will also explore the theoretical relation between the user-defined variables (i.e., window size, edge extraction threshold) and the classification performance of the proposed algorithms. Developing algorithms that can be applied to a dynamic graph which is subject to various types of updates, (i.e., insertions and deletions of nodes and edges) is another future direction.

ACKNOWLEDGEMENT

This material is based upon work supported by the National Science Foundation under Grant No.1318913.

REFERENCES

[1] S. Macskassy and F. Provost, "Simple models and classification in networked data," in *CeDER Working Paper 03-04*. Stern School of Business, New York University, 2004.

[2] N. S. Ketkar, L. B. Holder, and D. J. Cook, "Mining in the proximity of subgraphs," in *ACM KDD Workshop on Link Analysis: Dynamics and Statics of Large Networks*, 2006.

[3] D. J. Cook and L. B. Holder, *Mining graph data*. John Wiley & Sons, 2006.

[4] T. Gärtner, P. Flach, and S. Wrobel, "On graph kernels: Hardness results and efficient alternatives," in *Learning Theory and Kernel Machines*. Springer, 2003, pp. 129–143.

[5] N. Shervashidze and K. M. Borgwardt, "Fast subtree kernels on graphs." in *Proc. NIPS*, 2009, pp. 1660–1668.

[6] N. Shervashidze, T. Petri, K. Mehlhorn, K. M. Borgwardt, and S. Vishwanathan, "Efficient graphlet kernels for large graph comparison," in *Proc. AISTATS*, 2009, pp. 488–495.

[7] K. M. Borgwardt and H.-P. Kriegel, "Shortest-path kernels on graphs," in *Proc. ICDM*. IEEE, 2005, pp. 74–81.

[8] 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, 2005.

[9] N. Shervashidze, P. Schweitzer, E. J. Van Leeuwen, K. Mehlhorn, and K. M. Borgwardt, "Weisfeiler-Lehman graph kernels," *Journal of Machine Learning Research*, vol. 12, pp. 2539–2561, 2011.

[10] Q. Lu and L. Getoor, "Link-based classification," in *Proc. ICML*, vol. 3, 2003, pp. 496–503.

[11] C. C. Aggarwal, "On classification of graph streams." in *Proc. SDM*. SIAM, 2011, pp. 652–663.

[12] C. C. Aggarwal and N. Li, "On node classification in dynamic content-based networks." in *Proc. SDM*. SIAM, 2011, pp. 355–366.

[13] L. Chi, B. Li, and X. Zhu, "Fast graph stream classification using discriminative clique hashing," in *Advances in Knowledge Discovery and Data Mining*. Springer, 2013, pp. 225–236.

[14] B. Li, X. Zhu, L. Chi, and C. Zhang, "Nested subtree hash kernels for large-scale graph classification over streams," in *Proc. ICDM*. IEEE, 2012, pp. 399–408.

[15] B. Weisfeiler and A. Lehman, "A reduction of a graph to a canonical form and an algebra arising during this reduction," *Nauchno-Technicheskaya Informatsia*, vol. 2, no. 9, pp. 12–16, 1968.

[16] C. Domeniconi and D. Gunopulos, "Incremental support vector machine construction," in *Proc. ICDM*. IEEE, 2001, pp. 589–592.

[17] N. A. Syed, S. Huan, L. Kah, and K. Sung, "Incremental learning with support vector machines," in *Proc. IJCAI*. Citeseer, 1999.

[18] J. Neville, D. Jensen, L. Friedland, and M. Hay, "Learning relational probability trees," in *Proc. SIGKDD*. ACM, 2003, pp. 625–630.

[19] S. Pan, X. Zhu, C. Zhang, and P. S. Yu, "Graph stream classification using labeled and unlabeled graphs," in *Proc. ICDE*. IEEE, 2013, pp. 398–409.

[20] C.-C. Chang and C.-J. Lin, "LIBSVM: A library for support vector machines," *ACM Trans. on Intelligent Systems and Technology*, vol. 2, pp. 27:1–27:27, 2011, software available at http://www.csie.ntu.edu.tw/ cjlin/libsvm.