Intelligent Data Analysis 20 (2016) 825–852
DOI 10.3233/IDA-160834
IOS Press

Incremental SVM-based classification in dynamic streaming networks

Yibo Yao and Lawrence B. Holder*
School of Electrical Engineering and Computer Science, Washington State University, Pullman, WA, USA

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 an 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. We propose a framework combining an incremental support vector machine (SVM) with the Weisfeiler-Lehman (W-L) graph kernel. By retaining the support vectors from each learning step, the classification model is incrementally updated whenever new changes are made to the graph. We design an entropy-based subgraph extraction strategy, that selects informative neighbor nodes and discards those with less discriminative power, to facilitate the classification of nodes in a dynamic network. We validate the advantages of our learning techniques by conducting an empirical evaluation on several large-scale real-world graph datasets in comparison with other graph classification methods. The experimental results also validate the benefits of our subgraph extraction method when combined with the incremental learning techniques.

Keywords: Graph classification, dynamic graph, graph stream, incremental learning, graph kernel

1. Introduction

In recent years, a great amount of information has become available in the form of graphs, such as social networks, hyper-linked web documents, chemical compounds, and communication networks. As a result, networked data has gained popularity in the data mining community due to its superiority in describing the relations among entities. Thus there has been a growing interest in developing algorithmic techniques for performing supervised learning classification tasks on these graph datasets. The objective of graph classification is to classify graph-structured data instances into several categories. Generally, the problem of classification in a large-scale graph involves classifying nodes [18] or subgraphs [15] into suitable categories. In a binary classification scenario, an object (node or subgraph) is assigned to either a positive 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 [9].

Graph classification is itself a challenging task due to the rich and arbitrary structure of graphs. Traditional machine learning algorithms assume that data instances are represented in fixed-size numerical

*Corresponding author: Lawrence B. Holder, School of Electrical Engineering and Computer Science, Washington State University, Box 642752, Pullman, WA 99164, USA. E-mail: holder@wsu.edu.

vectors. But this assumption makes the algorithms incapable of catching the relations among entities for graph data and thus fail to classify the data effectively. Kernel methods, e.g., Support Vector Machine (SVM), provide an elegant solution to handling graph-structured data. The similarities among data instances are measured by implicitly mapping them into a high-dimensional space and then computing their inner products. 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 [12], subtrees [23], and subgraphs [24]. 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. 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 available information in memory or local storage. When the volume of graph data becomes extremely large, old information regarding nodes or edges must be eliminated from storage in order to maintain a modest speed of accessibility. In most cases, node or edge streams can be accessed only once, which makes it infeasible to compute the global kernel matrix.
- 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 will not be able to scale well because the resources needed for learning similarity matrices and the response time for any query (e.g., pattern match) 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 process. However, no effective strategy exists to select informative structural information to facilitate the classification progress in a dynamic graph.

Contribution: In order to address the above challenges, we report in this paper, an incremental graph kernel framework to investigate the problem of classification on large-scale and dynamically changing graphs. This framework encompasses two graph classification scenarios. In the *graph transaction* scenario, independent graphs are streaming in over time, and we seek to classify each graph. In the *one large graph* scenario, new nodes and edges are streaming in over time, and we seek to classify the new nodes, typically by extracting a subgraph around the neighborhood of the new node, thus transforming to the first scenario. We assume the data arrives in batches, but the batch size can vary. The main idea of our framework is to construct a classifier based on the data of the current batch and 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 only the support vectors. In case that the entire data cannot be loaded completely into main memory, we adopt a windowing scheme on the incremental SVM classification algorithm. The sliding window maintains recent data which is available for training and discards all older information from memory. 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 kernels to classify nodes inside a large-scale dynamic graph. We empirically test our algorithms on four real-world dynamic graph datasets covering the two classification scenarios. The experimental results clearly demonstrate the benefits of our entropy-based subgraph extraction strategy, as well as the superior performance of the proposed incremental learning techniques when compared to state-of-the-art methods.

The rest of this paper is organized as follows: In Section 2, we discuss the research work related to graph classification. Section 3 describes some definitions and notations. We present our framework in Section 4. In Section 5, we evaluate the proposed methods on four real-world datasets. Section 6 concludes the paper with a discussion on the completed work and possible lines of further improvements and investigations.

2. Related work

Graph classification techniques have often been conducted by comparing graphs using meaningful similarity measures. Graph kernels are a set of fundamental and powerful tools for quantifying the similarity between graphs. In recent years, there are a large number of graph kernels that have been developed, see [27] and references therein. Most of them follow the same principle of enumerating common substructures. Some typical substructures that have been used to build graph kernels include random walks, shortest paths, subtrees and small subgraphs.

In [12,14], the authors develop random walk-based graph kernels, which count the number of common labeled walks existing in two graphs. However, computing the kernel value between two graphs has the worst-case time complexity of $O(N^6)$, where N denotes the number of nodes in these two graphs. It is shown in [6] that the complexity of these walk-based kernels can be reduced to $O(N^3)$. A path-based kernel is introduced in [5], which computes the length of shortest-paths between all pairs of nodes and then counts pairs with similar attributes and lengths. The authors also prove that the k-shortest path is computationally tractable with time complexity $O(N^4)$.

A drawback of these walk- or path-based kernels is that the structures chosen to express the similarities among graphs are too simple. However, it is shown in [12] that computing graph kernel values by counting common subgraphs is known to be NP-hard. Thus, another direction in developing an efficient graph kernel is to employ graphlets (small subgraphs with order $k \in \{3, 4, 5\}$) to characterize graphs when computing their similarities [24]. The authors show that for a graph with bounded degree d, the connected graphlets can be enumerated in $O(Nd^{k-1})$. Ramon and Gärtner [22] propose a subtree kernel based on common subtrees of height h rooted at any pair of nodes from two different graphs. The recently proposed Weisfeiler-Lehman (W-L) kernel [25] is a fast subtree 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 [28], which counts the matching multiset labels of the entire neighborhood of each node up to a given distance h. The W-L kernel has $O(Mh)$ complexity, where M is the number of edges in the graphs.

On the other hand, with the emergence of streaming data, there also exists some work related to developing classification techniques on dynamic streaming graphs. In [2], 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 [1], 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 [8] to classify graph streams by detecting discriminative cliques and mapping them onto a fixed-size common feature space. The authors in [17] use their presented Nested Subtree Hash (NSH) algorithm based on the W-L kernel to project different subtree patterns from graph streams onto a set of common low-dimensional feature spaces, and construct an ensemble of NSH kernels for large-scale graph classification over streams. Both of the two aforementioned methods aim to find common feature (subtree or clique) patterns across the data stream and map those patterns onto a lower dimensional feature space using random hashing techniques. However, they are likely to lose some discriminative substructure patterns by compressing the feature patterns into a fixed-size feature space during the hashing process. In addition, they are not applicable to a single large-scale dynamic graph without a subgraph extraction process.

The node classification problem in networked data has also been studied in the context of label propagation. In real networks (e.g., social networks, communication networks), links between nodes usually indicate some form of relationship between the corresponding entities. In particular, an edge can indicate some degree of similarity between the linked entities. Therefore, it may be helpful to classify a node of interest based on information of the nodes close to that target node. Label propagation [3,30,31] is a popular semi-supervised learning technique to assign class labels to the unlabeled nodes by spreading the information of labeled nodes in the graph. It aims to learn a global labeling function over the graph with convergence guarantees and has been shown to be equivalent to propagating the labels by performing random walks on the graph [4]. Although label propagation utilizes the global structure of a network to help predict labels, it is usually described in terms of finding the inverse of a matrix representing the network structure (e.g., adjacency matrix, weight matrix, graph Laplacian). Therefore, it becomes computationally expensive to apply this technique on a large dynamic network in which the number of nodes grows unboundedly.

The framework presented in this paper addresses the novel task to learn a classification model incrementally on a large-scale and time-evolving graph. In our previous work [29], we proposed a scalable SVM-based learner for classifying nodes in dynamic graphs with only insertions of nodes and edges. In this paper, we extend that work to include the classification for individual graphs in a graph stream and provide more detailed explanation to the presented algorithms. We also experiment with more real-world datasets to demonstrate the performance of our methods.

3. Preliminaries

We first introduce some notations and definitions. For the rest of this paper, we use network and graph interchangeably.

Definition 1. A *labeled graph* is represented by a 4-tuple, i.e., $G = (\mathcal{V}, \mathcal{E}, \mathcal{L}, l)$, where

(1) $\mathcal{V} = \{v_1, \ldots, v_{|\mathcal{V}|}\}$ is a set of nodes,
(2) $\mathcal{E} \subseteq \mathcal{V} \times \mathcal{V}$ is a set of edges,
(3) $\mathcal{L}$ is a set of labels,
(4) $l : \mathcal{V} \cup \mathcal{E} \rightarrow \mathcal{L}$ is a function assigning labels to nodes and edges.

Definition 2. Let $G = (\mathcal{V}, \mathcal{E}, \mathcal{L}, l)$ and $G' = (\mathcal{V}', \mathcal{E}', \mathcal{L}', l')$ 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. $\forall v \in \mathcal{V}, l(v) = l'(v)$,
3. $\mathcal{E} \subseteq \mathcal{E}'$,
4. $\forall (u, v) \in \mathcal{E}, l(u, v) = l'(u, v)$.

In many application domains, including social networks, communication networks and biological networks, the graph structures are subject to streaming changes, such as insertions or deletions of nodes and edges. These changes complicate the learning tasks (i.e., classification, clustering) to be performed on the network. However, we have considered the networks with only insertions of nodes and edges in our research.

Definition 3. An *update*, denoted by U, is a node or an edge to be inserted into the graph.

Definition 4. A *dynamic streaming graph*, denoted as $\mathcal{G}$, is a sequence of updates $\{\ldots, U_i, U_{i+1}, \ldots\}$, where each U_i is applied to the object graph in a streaming fashion. $\mathcal{G}^t = \{\bigcup_{i=0}^{i=t} U_i\}$ denotes the object graph at time t, and $\mathcal{G}^0 = \emptyset$ is the initial graph without any nodes or edges.

We assume that each update U_i can be denoted in the form of an edge, i.e., $e_i = \langle v_{i1}, v_{i2}, \ell_i \rangle$, where v_{i1}, v_{i2}, ℓ_i denote the two endpoints and the label of e_i. If the graph is a directed graph, then e_i has a direction pointing from v_{i1} to v_{i2}. Applying U_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 $v_{i_1}, v_{i_2} \notin \mathcal{G}$.

It is common that a real-world network consists of different types of nodes and their relationships (e.g., a paper-author network), and users are often interested in categorizing a certain type of node (e.g., paper nodes in a paper-author network) with help of the other types of nodes (e.g., author nodes in a paper-author network).

Definition 5. A node to be classified is defined as a *central node* which denotes a central entity from the original data, and a node is defined as a *side node* if it is not a central one.

Figure 1 shows one instance of a citation network in which all papers P_i are marked as central nodes while authors A_i are 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 of classification in a dynamic streaming graph. The following two scenarios are considered in our framework.

- *Central node classification*: Given a dynamic graph with central and side nodes indicated in its representation, and each central node v_i is associated with a class label $y_i \in \{+1, -1\}$, the aim is to learn a classifier using the available information up until time t, and to predict the class membership of any new central nodes arriving at time $t + 1$. See Fig. 2 for an example.
- *Isolated-graph classification*: Here, we take an entire small graph as an instance to be classified. Each isolated graph is independent from the others and is associated with a class label $y_i \in \{+1, -1\}$. The goal is to build a classification model on the instances up until time t, and to predict the class labels of the instances in the next batch. See Fig. 3 for an example. This scenario is also referred to as a graph transaction.

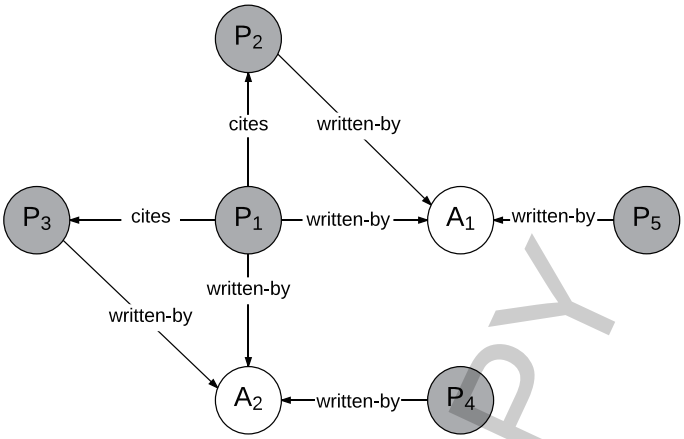


Fig. 1. An instance of a citation network. P_i represents a paper while A_i represents an author associated to a paper. An edge label represents the relationship between the two nodes it connects.

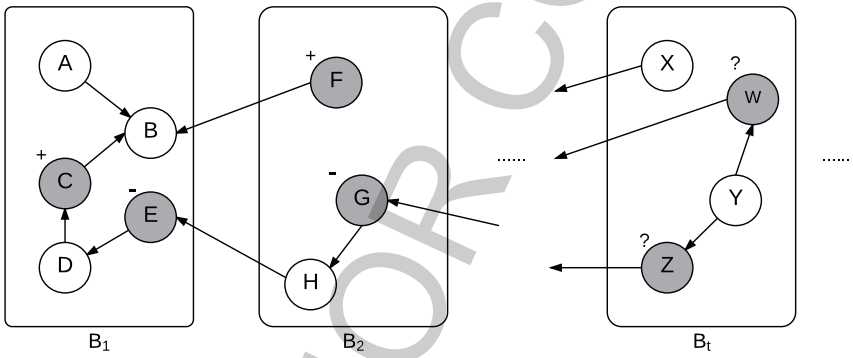


Fig. 2. Central node classification. The shaded nodes denote the central nodes, and the symbols $\pm$ 1 indicate their class labels. B_t denotes the batch of updates received at time t. When B_2 is received, *case 2* occurs since two new nodes F and H are connected to two old nodes B and E; *case 4* also occurs as a new edge connecting two new nodes H and G is inserted.

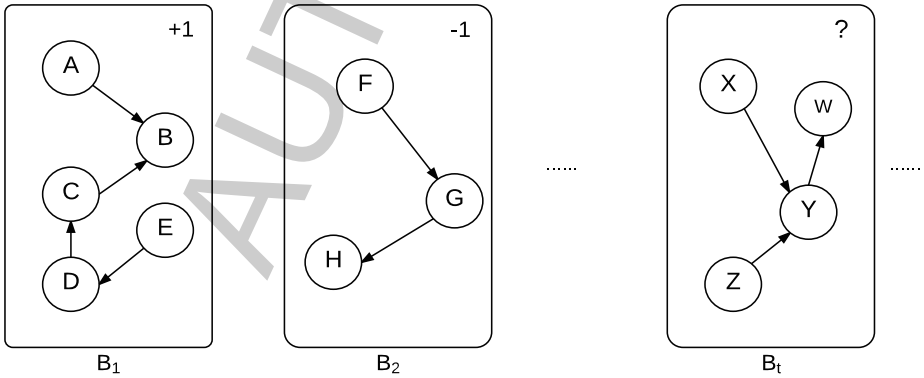


Fig. 3. Isolated-subgraph classification. The symbols $\pm$ 1 indicate these isolated subgraphs' class labels. B_t represents the set of updates received at time t, which contains a small graph to be streamed into the underlying large graph.

Definition 6. A *batch* B_t is a set of updates that are applied to the object graph at time t, i.e., $B_t = \{U_t^1, U_t^2, \ldots, U_t^n\}$.

In our research, we assume that the updates are received in the form of batches, and these batches contain various numbers of updates. By using the batch notation, an alternative representation to denote a dynamic streaming graph is as follows:

$$\mathcal{G} = \bigcup_{t=1}^{\infty} B_t$$

3.1. Weisfeiler-Lehman graph kernel

The Weisfeiler-Lehman (W-L) graph kernel [25] 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 [28]. In this algorithm, each node's label is augmented by a sorted set of node labels of neighbor nodes, and the augmented labels are compressed into new labels using mapping functions. These steps are then repeated for r iterations. However, the structure of the graph remains unchanged during the process. Let G_i denote the augmented graph at the ith iteration (G_0 is the initial graph with original node labels), then the W-L kernel with r iterations between two graphs can be calculated as

$$K^{(r)}(G, G') = \sum_{i=0}^{i=r} \kappa(G_i, G_i')$$

where $\kappa(G_i, G_i') = \langle \phi(G_i), \phi(G_i') \rangle$ and $\phi(G_i)$ is a numeric vector whose elements record the numbers of occurrences of node labels in G_i.

However, the W-L kernel based on batch mode will suffer from memory and time issues as described in Section 1 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.

3.2. 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 [10]. In order to make SVMs suitable for incremental learning, the data stream is partitioned into batches and each batch contains a small number of examples so that it can be loaded into memory. Then, at each learning step, the support vectors which define the decision boundary are retained as a representation of the data seen so far and incorporated with the new incoming batch of data to serve as the training data [26]. This incrementally built model is expected to be an approximation to the model built with the complete data, since it makes use of the essential class boundary information which contributes significantly to generate the classifier at the consecutive step.

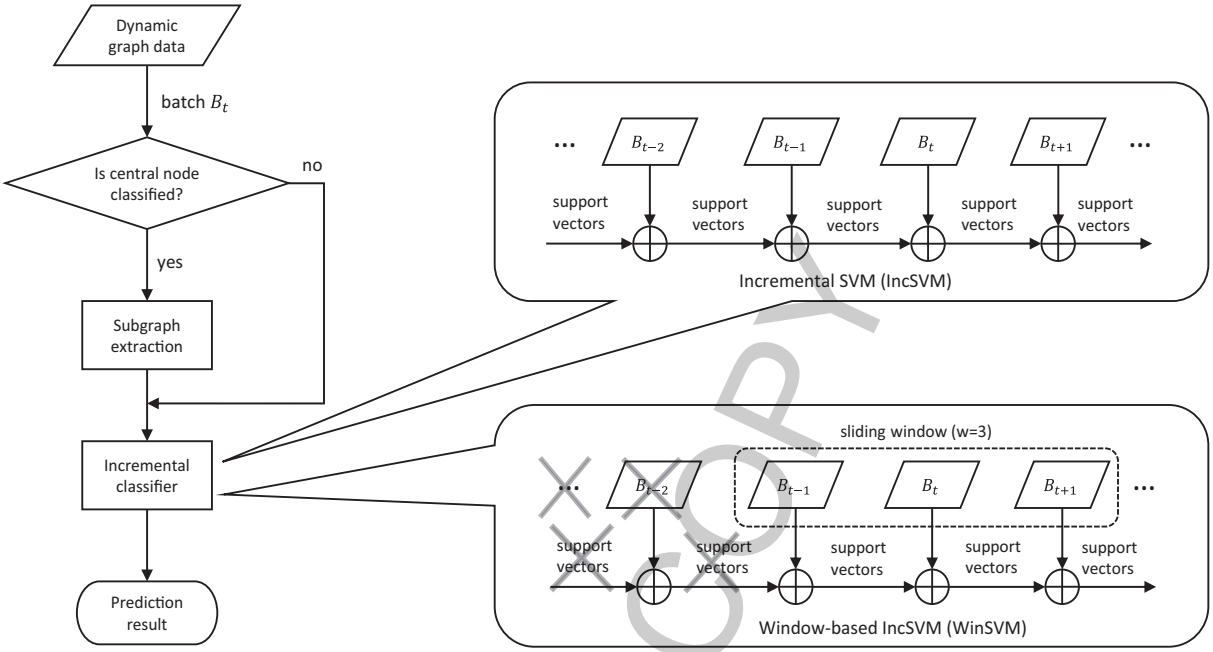


Fig. 4. Working flow of the proposed incremental classification framework.

4. Framework

In this section, we introduce the three key components included in our graph classification technique: (1) extracting a 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 incremental learning algorithm works as follows:

1. If it is a central node classification scenario, 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. Train a new model on this set to predict the class labels of the subgraphs (and the central nodes from which they are built) in B_{t+1}.

Figure 4 depicts our proposed incremental classification framework. In the following subsections, we first introduce a subgraph extraction scheme for central nodes, then describe the incremental classification method and the window-based incremental method.

4.1. 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

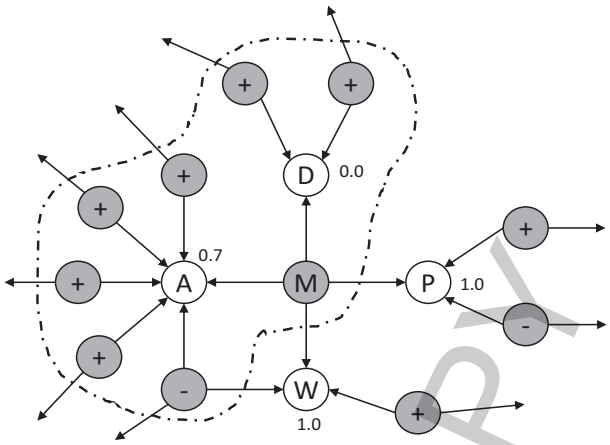


Fig. 5. Entropy-based subgraph extraction from a large movie network. The shaded nodes denote the movie nodes, while the nodes labeled A, D, P, W denote the actor, director, producer, and writer associated to these movies. The shaded node labeled M is the target movie node. The other shaded nodes are the movies share the same actor, director, producer or writer with M, and the labels $\pm$ imply their class memberships. The entropy values for the 4 side nodes are indicated using real numbers. In this setting, if we set the entropy threshold $\theta \leqslant 0.8$, then the subgraph surrounded by the dash line is the one that will be 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 lose discriminative information for classification, thus reducing the classification performance.

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 $v_i \in \mathcal{G}$ (v_i can be a central or side node), 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:

$$p_1 = \frac{n_{pos}}{n_{pos} + n_{neg}} \text{ and } p_2 = \frac{n_{neg}}{n_{pos} + n_{neg}}$$

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

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

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 discriminative 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. Moreover, we assume that each side node must be connected only to central nodes in our graph representation. In other words, we do not allow interconnections between any two side nodes in the representation. In most domains, this constraint does not impose undue limitations on the representation of information, because most side nodes represent attributes of the central node, and most relationships of interest in the graph are between central nodes. The main idea of our extraction method is that we start from v_c and keep extracting neighbor nodes whose entropy values $\leqslant \theta$ 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. Algorithm 1 shows the detailed procedure for extracting a subgraph surrounding a central node from a graph. Figure 5 illustrates a movie subgraph extracted from a large movie network.

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

The proposed extraction scheme will only extract, at most, all the nodes connected to v_c and all the central nodes connected to the side nodes of v_c. 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. 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 efficient 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.

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 Subsection 4.3). There is a possibility that a certain update in the current batch B_t may refer to older nodes which have been inserted at batches previous to B_t but deleted from memory because they are outside the sliding window. In our extraction scheme, we allow these older nodes to be re-inserted into the underlying graph when new incoming edges refer to them.

4.2. 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. Some traditional learning algorithms, e.g., decision tree [11], SVM [10,26], have been extended to their incremental versions to build the classification models for high-speed data streams. Since graph kernel methods with SVM have shown good performance when classifying static graph-structured data, we combine an incremental SVM with one of the fastest graph kernels, namely W-L kernel, to tackle the problem of classification in dynamic graphs. 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. The input includes a dynamic graph, a set of support vectors retained from the previous model, the current batch of updates, and the threshold for extracting subgraphs. The output is a new SVM model including its support vectors. First, the current batch of updates, B_t, is applied to the dynamic graph $\mathcal{G}$. If it is a central node classification scenario, SubExtract is called for extracting a set of subgraphs for the central nodes in B_t (lines 3–5). For isolated-graph scenario, we do not need to call SubExtract (line 7). A new training set is constructed by combining the support vectors from the previous learner and the extracted subgraphs (line 9). Then a new SVM model is built using the W-L kernel matrix of the training set (line 10). Finally, the newly built SVM model is returned for predicting the instances (i.e., central nodes or isolated-graphs) in the next batch (line 11). In our experiments, IncSVM has shown improved performance for classifying nodes/subgraphs in large-scale dynamic graphs when compared to state-of-the-art methods. See Section 5 for the detailed performance results.

4.3. 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

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:

$\mathcal{M}_t$: A SVM classification model for prediction

1: $Sub_{B_t} = \emptyset$
2: **if** *central node classification* **then**
3: **for** each central node v_c in B_t **do**
4: $Sub_{B_t} = Sub_{B_t} \cup \{\text{SubExtract}(\mathcal{G}, v_c, \theta)\}$
5: **end for**
6: **else if** *isolated-graph classification* **then**
7: $Sub_{Bt} = B_t$
8: **end if**
9: construct a training set $TR_t = SV_{t-1} \cup Sub_{B_t}$
10: learn a classifier $\mathcal{M}_t$ on TR_t using W-L kernel
11: **return** $\mathcal{M}_t$ (including its support vectors SV_t)

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 only retaining support vectors, some other strategies are needed to remove old information which is not valuable in deciding the separating hyperplane.

Windowing approaches have been designed to enable machine learning algorithms to process data streams with strict constraints on space and time. A fixed-size sliding window is an easy and straightforward way of limiting the amount of processed data. It maintains the most recent data and discards all older information for any learning algorithm. However, the window size is usually determined through empirical tests. A narrow window usually produces an accurate representation of the recent state but will not generalize well. A wide window results in more stable results but fails to react quickly to a changing environment. The landmark windowing scheme [32] tracks the data statistics over all data points starting from a particular time point called the landmark. But the window size will grow arbitrarily large when the data keeps streaming in. The damped windowing model [32] assigns weights to the data points according to their ages. The weights of data decrease exponentially into the past. This guarantees that the recent data has a more important influence for building the up-to-date learning model than the old data.

In this paper, we have adopted the easiest windowing scheme, i.e., fixed-size sliding window, to maintain a limited amount of graph data in memory. We present another incremental SVM learner with a sliding window. 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. Our window-based incremental SVM (WinSVM) is described in Algorithm 3.

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. As a result, the support vectors form B_{t-W} are also removed

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($\mathcal{G}, SV_{t-1}, B_t, \theta$)

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. Referring to Fig. 1 as a toy example, suppose one 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 no longer has discriminative power. On the other hand, W depends to some extent on the batch size $|B|$. A larger window size will generally produce higher classification accuracy but need more learning time. In practice, when $|B|$ is relatively large, we should avoid setting W arbitrarily large, in order to keep the accessibility and runtime at a tractable level. When $|B|$ is relatively small, we should avoid using a small W since a learner trained using fewer examples will produce higher generalization error. The batch size is typically chosen based on the frequency with which data is generated in the particular domain. In the experiments (refer to Section 5), we demonstrate that WinSVM can clearly outperform the compared state-of-the-art methods in terms of classification effectiveness. We also run WinSVM with different values for $|W|$ and show that WinSVM can achieve comparable classification accuracy with IncSVM but perform more efficiently.

4.4. 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 $|B|$ and the subgraph extracted for each central node in a batch has no more than $|\mathcal{V}_B|$ nodes and $|\mathcal{E}_B|$ edges. Then the computational complexity of the W-L graph kernel for one batch is $O(r|B||\mathcal{E}_B|)$ [25] where r is the number of iterations used in the W-L isomorphism test. The value of r is usually chosen by using cross-validation on the training data [25]. But for practical problems, it is recommended to set $r < 10$ [17,23,25]. In our experiments (see Section 5

for details), we have set $r = 5$. Therefore, the time complexity for computing the W-L graph kernel in our experiments can be reduced to $O(|B||\mathcal{E}_B|)$.

For any central node, we assume the maximum numbers of central nodes and side nodes connected to it are n_c and n_s respectively, and the maximum degree of any side node is d_s. So the maximum degree of a central node is bounded by $O(n_c + n_s)$. SubExtract takes $O(n_c)$ and $O(n_s)$ to extract the central nodes and side nodes from the neighbor of a central node. For each extracted side node, it will need $O(d_s)$ to extract the surrounding nodes in order to reach the same type of nodes as the central node. Therefore, the overall time complexity for SubExtract is $O(n_c + n_s d_s)$. For some particular cases, there is no connection between any two central nodes (e.g., the IMDb Network we have used in Subsection 5.1). Then the time complexity of SubExtract can be further simplified to $O(n_s d_s)$ since $n_c = 0$ under these cases.

For IncSVM, the worst case is that the set of support vectors retained from the 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 indicates that the complexity for IncSVM is $O(t|B||\mathcal{E}_B|)$. 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 $O(W|B||\mathcal{E}_B|)$ (W is the size of a sliding window) time to compute a kernel matrix.

5. Experimental evaluation

In this section we apply our dynamic graph classification algorithms on graph-structured data from practical application domains. In particular, we evaluate the effectiveness of the entropy-based subgraph extraction method by setting different values for the entropy threshold. We validate the proposed incremental learning techniques on four real-world dynamic graph datasets by comparing them with two of the state-of-the-art algorithms related to the considered problem in our research. All these datasets are publicly available.

5.1. Benchmark data

5.1.1. NCI graph stream

The National Cancer Institute (NCI) datasets[1] contain information on anticancer activities and are frequently used as a benchmark for graph classification [16,17]. In our experiment, we use five cancer datasets consisting of 21,058 chemical compounds in total. Each chemical compound is represented as a graph with nodes denoting atoms and edges denoting bonds between atoms. Since each dataset is a bioassay task for anticancer activity prediction, a graph is labeled as a positive example if its chemical compound is active against the corresponding cancer type. Figure 6 summarizes the five datasets. We concatenate these datasets sequentially to simulate a large-scale dynamic chemical compound network with 6.5×10^5 nodes and 7.1×10^5 edges. This network falls under the isolated-graph classification scenario, where each graph is independent and isolated from the others.

5.1.2. 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, box-office receipts, etc. In our experiments, we

[1] http://pubchem.ncbi.nlm.nih.gov.
[2] http://www.imdb.com.

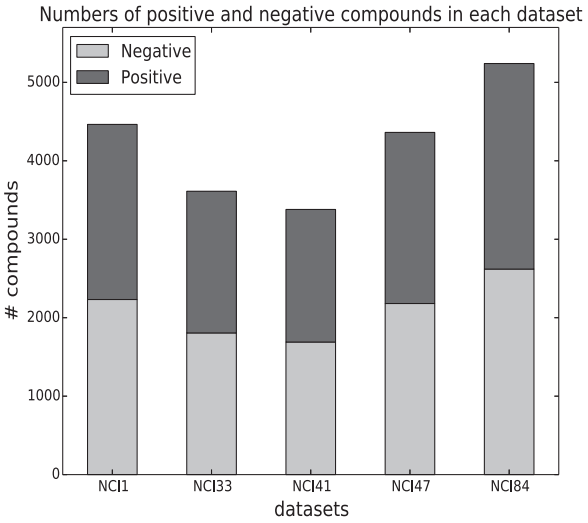


Fig. 6. The numbers of positive and negative chemical compounds in each cancer dataset.

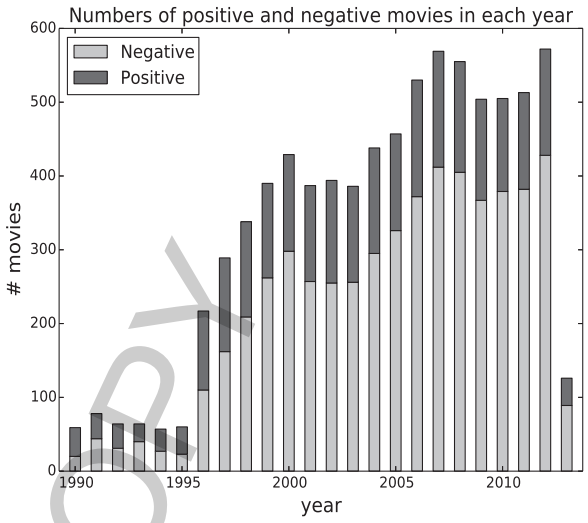


Fig. 7. The numbers of positive and negative movies in each year for IMDb dataset.

have identified a sample consisting of 7,535 movies[3] released between 1990 and 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, acted by an actor A_1, written by a writer W_1, and produced by a producer P_1, there is a directed edge with label *directed-by* from M_1 to D_1, a directed edge with label *acted-by* from M_1 to A_1, a directed edge with label *written-by* from M_1 to W_1, and a directed edge with label *produced-by* from M_1 to P_1, respectively. The dynamic movie network is formed by continuous insertions of aforementioned nodes and edges chronologically when new movies appear along with their actors, directors. 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) [19] when it is released. Among all the identified movies, 2,524 of them (whose opening weekend revenue $\geqslant$ \$2 million) are labeled as positive while the others are labeled as negative. Figure 7 plots the numbers of positive and negative movies released in each year between 1990 and 2013. Note that for 2013, the original data contains the movies released only between January and March.

5.1.3. DBLP network

DBLP[4] 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 [21], our classification task is to predict which of the following two fields a paper belongs to: DBDM (database and data mining: published in conferences VLDB, SIGMOD, PODS, ICDE, EDBT, SIGKDD, ICDM, DASFAA, SSDBM, CIKM, PAKDD, PKDD, SDM and DEXA) and CVPR (computer vision and pattern recognition: published in conferences CVPR, ICCV, ICIP, ICPR, ECCV, ICME and ACM-MM). We have identified

[3] http://www.aviz.fr/Teaching2013/Datasets.
[4] http://arnetminer.org/citation.

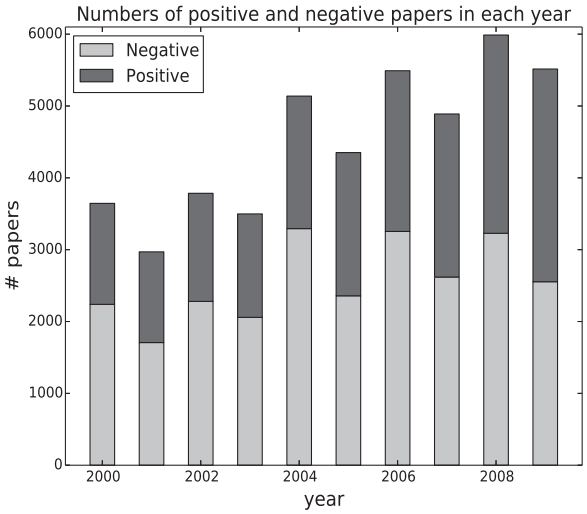


Fig. 8. The numbers of positive and negative papers in each year for DBLP dataset.

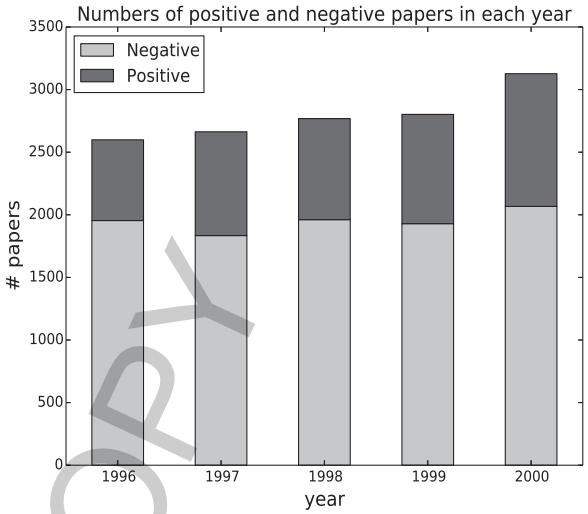


Fig. 9. The numbers of positive and negative papers in each year for HEPTH dataset.

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. Figure 8 plots the numbers of positive and negative papers published in each year between 2000 and 2009.

5.1.4. HEPTH network

The HEP-TH dataset was derived from the arXiv archive, which was originally released in KDD Cup 2003 [13]. It presents information on papers which were related to theoretical high-energy physics and submitted to arXiv during the period from January 1992 to April 2003. This dataset was later consolidated by UMass KDL into an XML format.[5] In our experiments, we extract 13,959 papers which have been submitted to arXiv between January 1996 and December 2000 and their authors and journals, to construct a large dynamic citation network. We denote that (1) each paper ID is a central node while each author ID or journal ID is a side node; (2) if a paper P_1 cites another paper P_2, there is a directed edge with label *cites* pointing from P_1 to P_2; (3) if a paper P_1 is written by an author A_1, there is a directed edge with label *authored_by* pointing from P_1 to A_1; (4) if a paper P_1 is published in a journal J_1, there is a directed edge with label *published_in* from P_1 to J_1. The learning task is to predict citation counts of a paper within one year after its submission to arXiv. All papers are categorized into two classes: a positive label implies a paper that receives at least 5 citations during the year after its submission, while a negative label implies fewer than 5 citations. Figure 9 shows the numbers of positive and negative papers in every year. The final graph consists of 2.5×10^4 nodes and 2.3×10^5 edges. This dynamic citation network is continuously updated with the insertions of nodes and edges when new papers appear along with their authors, references, and journals.

[5] https://kdl.cs.umass.edu/display/public/HEP-Th.

5.2. 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)* [17] is an ensemble learning framework which consists of W 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 and

$$w_i = \sum_{y\in\{\pm\}} P_t(y)(1-P_t(y))^2 - \frac{1}{|B_t|}\sum_{n=1}^{|B_t|}(0.5(1-y_n^t f_i(x_n^t)))^2$$

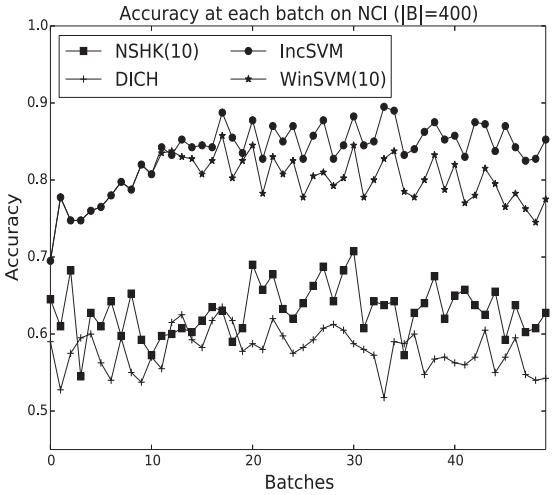
is the weight for f_i measured by the mean square errors related to f_i and the class distribution, and $P_t(y)$ denotes the class distribution in B_t. Each classifier of the ensemble is constructed using the W-L kernel. During the W-L isomorphism test, hashing techniques are used to map the unlimited subtree patterns into low-dimensional feature spaces. In our experiments, the ensemble size has been fixed to 10. We set the number of iterations for W-L isomorphism test as 5, and the dimensionality of the hashed feature space for the 5 iterations are {500, 1000, 5000, 10000, 50000}, respectively.

– *Discriminative Clique Hashing (DICH)* [8] uses a random hashing technique to compress infinite edge space onto a fixed-size space, applies a fast clique detection algorithm to detect frequent discriminative cliques to build a pattern-class table from a graph stream, and then constructs a rule-based classifier. 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.

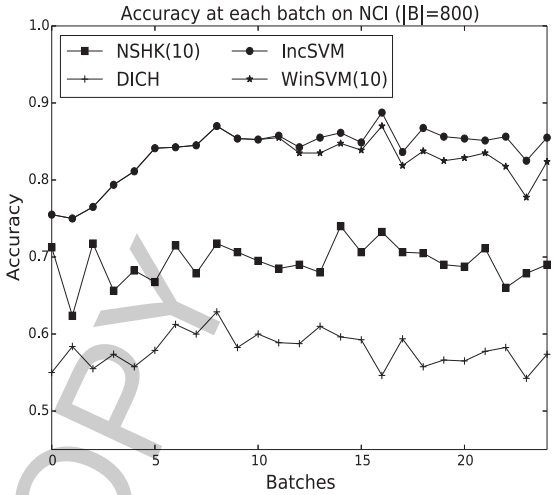
All SVM-based classification tasks (including NSHK, IncSVM, WinSVM) are conducted using LIBSVM [7]. To make a fair comparison, we set 5 as the maximum number of iterations for the W-L isomorphism test in IncSVM and WinSVM. Unless specified otherwise, we use the following settings for the parameters of these methods: batch size $|B_t| = \{200, 400\}$ for IMDb and $\{400, 800\}$ for NCI, DBLP and HEPTH, subgraph extraction threshold $\theta = \{0.2, 0.4, 0.6, 0.8, 1.0\}$, and window size $W = \{6, 8, 10\}$.

5.3. Performance evaluation

The effectiveness of these methods 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.

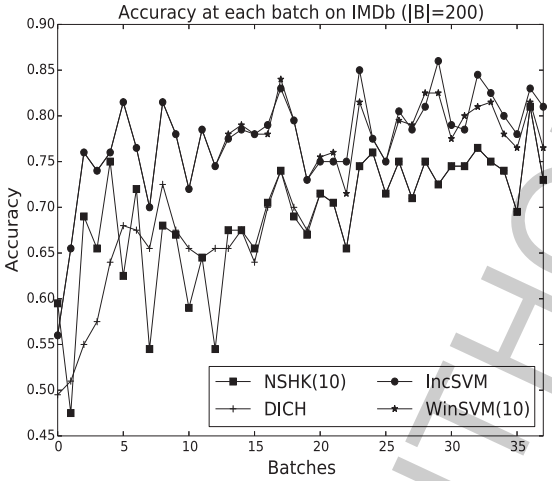


(a) $|B| = 400$

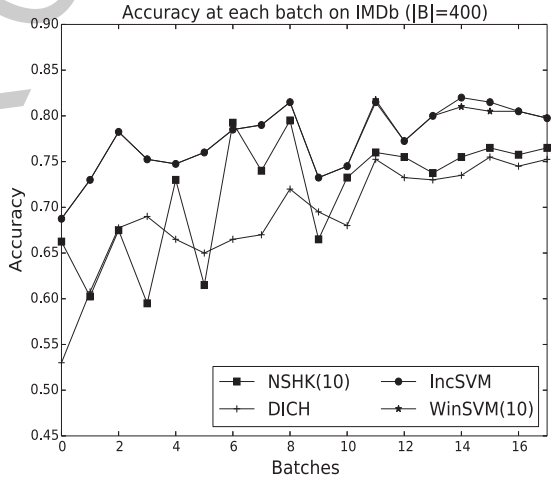


(b) $|B| = 800$

Fig. 10. Accuracy at each batch on NCI.



(a) $|B| = 200$

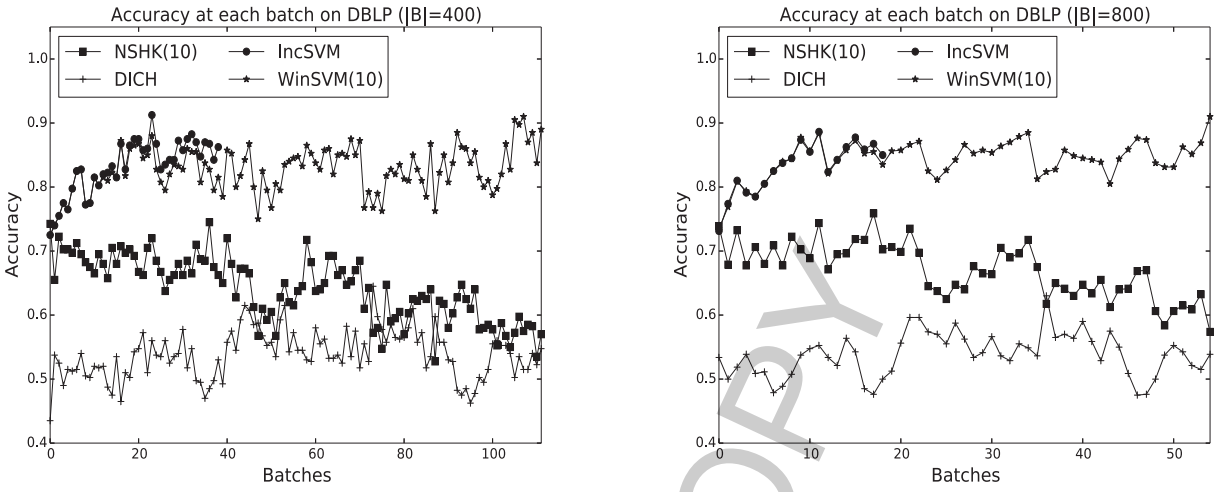


(b) $|B| = 400$

Fig. 11. Accuracy at each batch on IMDb.

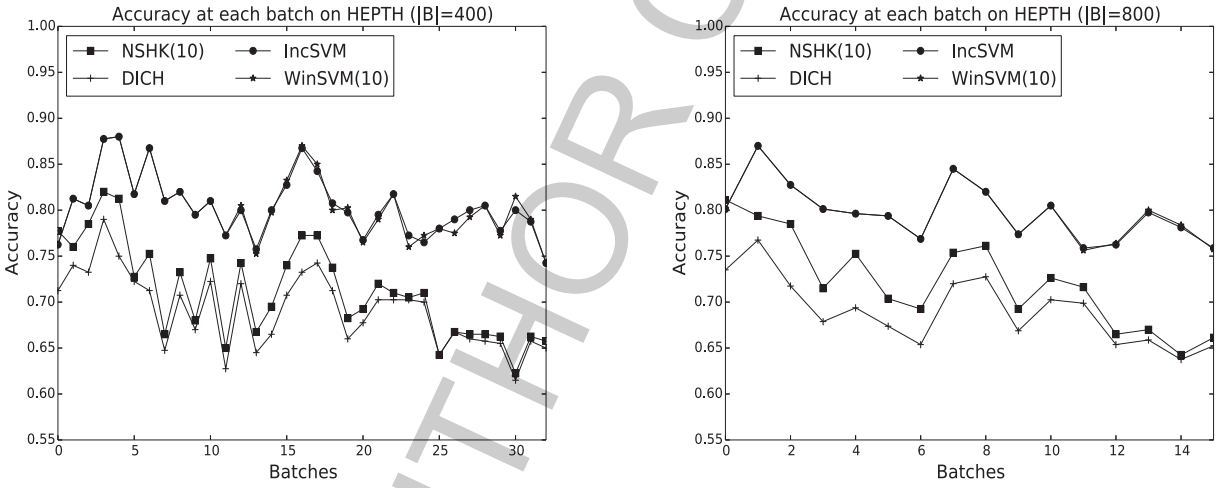
5.3.1. Classification performance analysis

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 IMDb, DBLP and HEPTH networks in order to classify the central nodes in them. Nevertheless, we do not apply the subgraph extraction method on NCI data since each subgraph is an isolated small graph to be classified. 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. Figure 10 through Fig. 13 show the accuracy values at different learning steps for the four datasets. We vary the number of subgraphs in each batch and fix the



(a) $|B| = 400$ (b) $|B| = 800$

Fig. 12. Accuracy at each batch on DBLP.



(a) $|B| = 400$ (b) $|B| = 800$

Fig. 13. Accuracy at each batch on HEPTH.

window size $W = 10$ for WinSVM.[6] IncSVM does not scale well on DBLP in this experiment because almost all training examples tend to be support vectors, so we only show the accuracy values of IncSVM for the first few batches. We can clearly see that our incremental learning techniques consistently outperform their peers across all batches of those 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 shown in Fig. 14 demonstrate this fact. We find that the average accuracy values are insensitive to $|B|$,

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

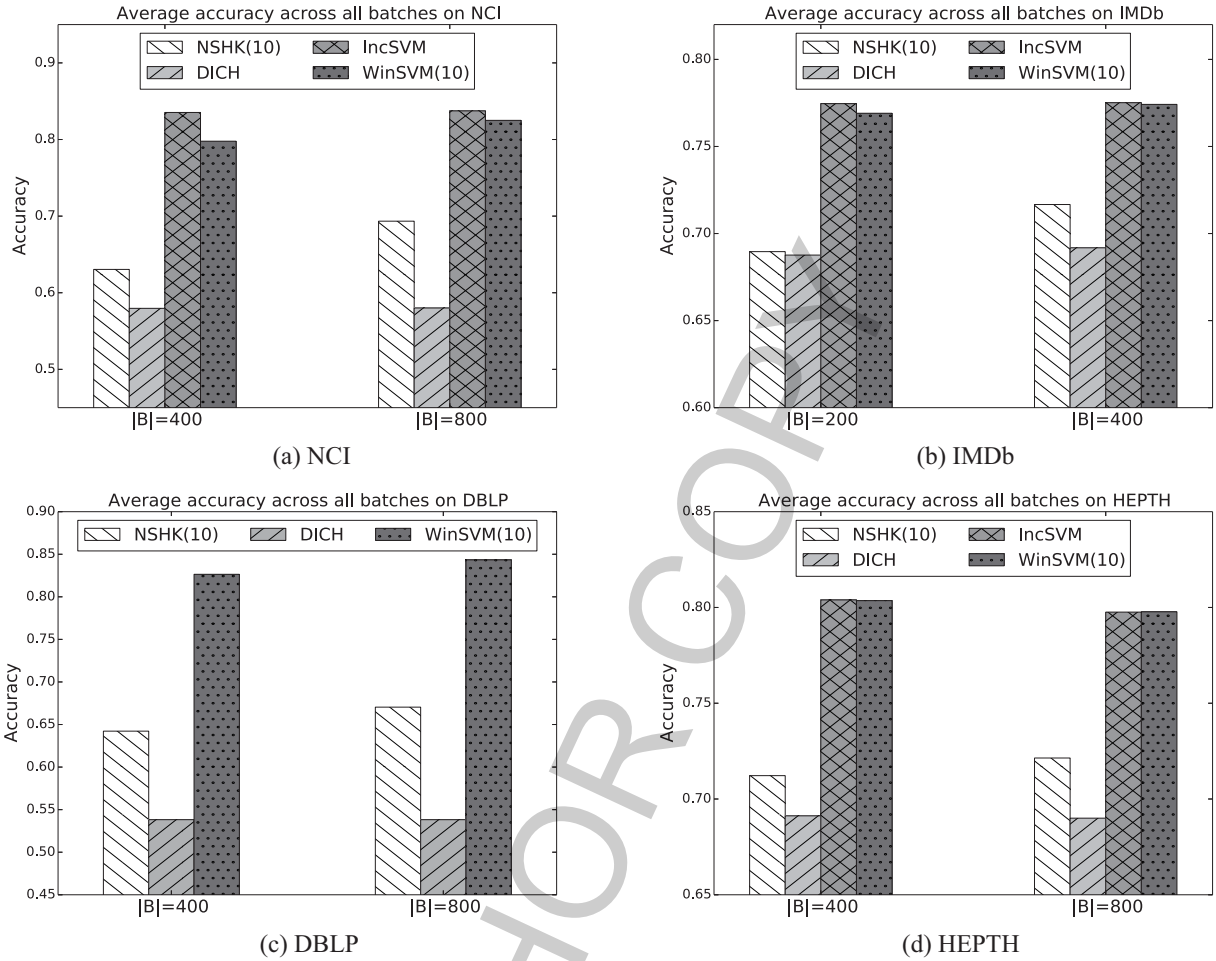


Fig. 14. Average accuracy across all batches.

which implies the stability of our techniques. Another observation is that WinSVM can generate impressive accuracy comparable to that of IncSVM on NCI, IMDb and HEPTH, which implies 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 across all batches for running these algorithms on the four datasets. The results are shown in Fig. 15. 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. IncSVM takes a little more time than NSHK on NCI and HEPTH when $|B| = 400$. But when the batch size is larger, IncSVM needs less learning time than NSHK on these two datasets. Contrastingly, IncSVM takes much less time than NSHK with $|B| = 200, 400$ on IMDb. We find that, the learning time of IncSVM tends to decrease when the batch size $|B|$ is increased. On the other hand, WinSVM always performs better than NSHK and IncSVM in terms of accumulated learning time on all the four datasets.

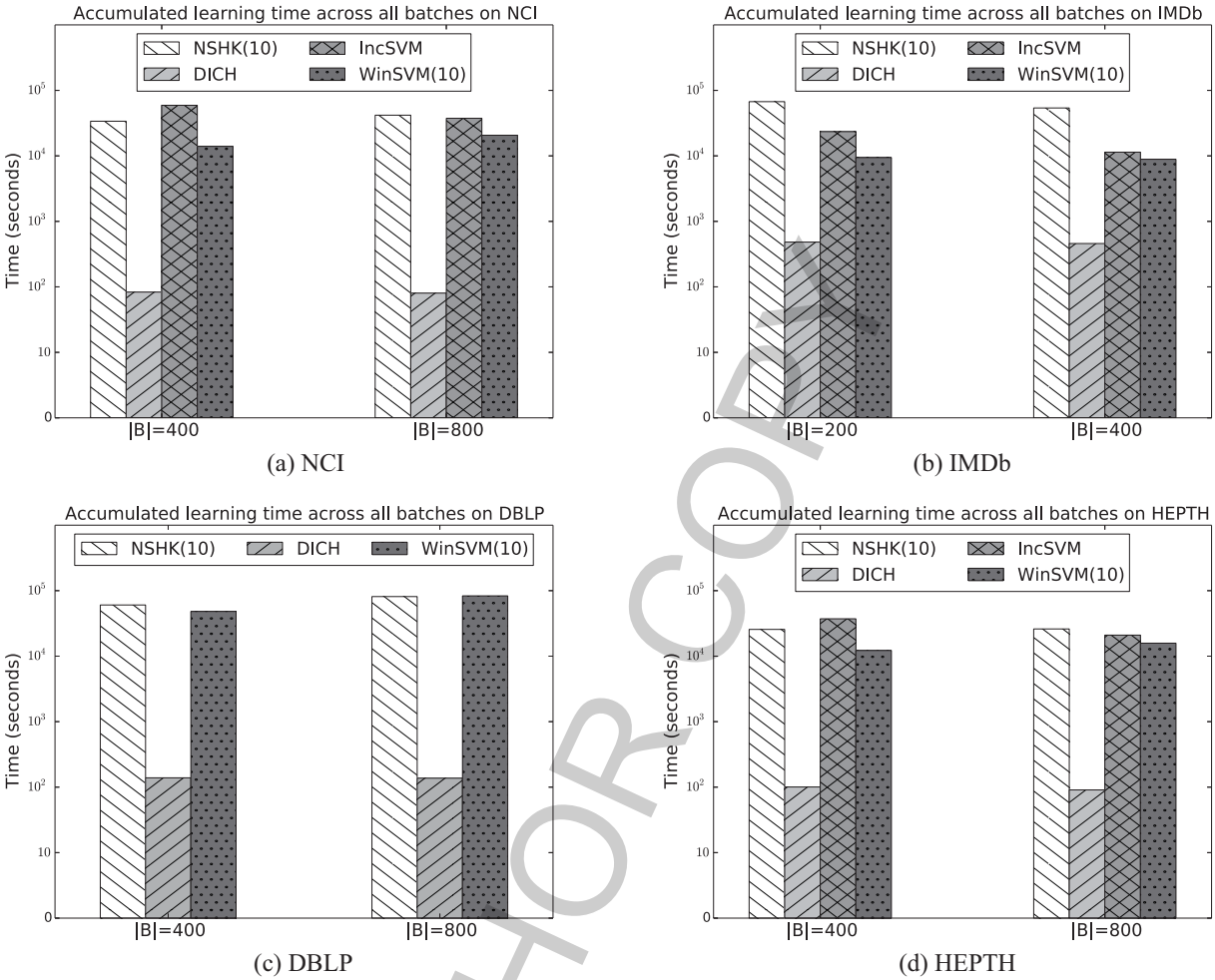


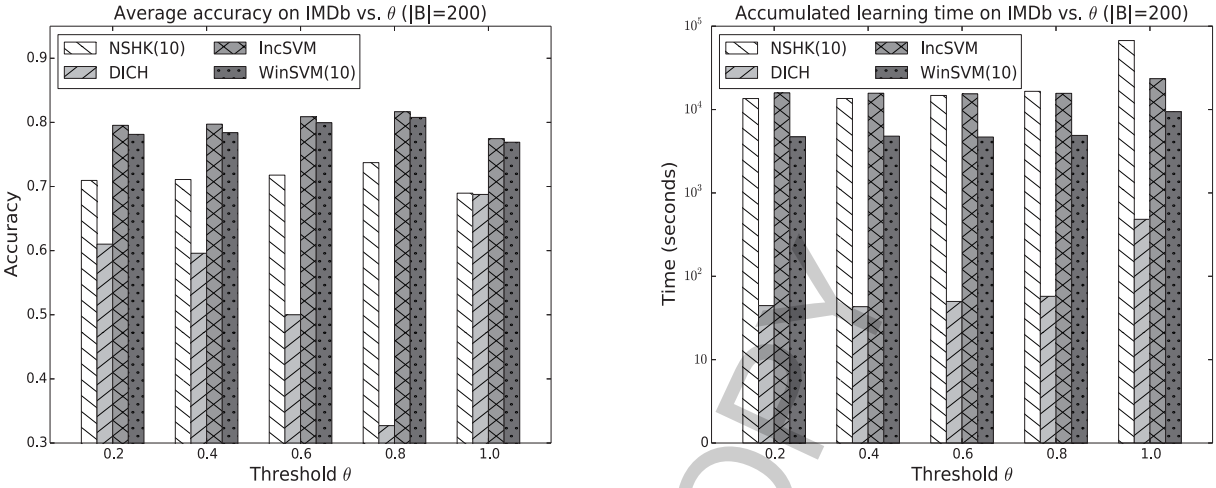
Fig. 15. Accumulated learning time across all batches.

Considering the higher accuracy that IncSVM and WinSVM achieve, we conclude that our incremental learning framework has the best classification performance. Furthermore, given the fact that WinSVM has much 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.

5.3.2. Impact of subgraph extraction

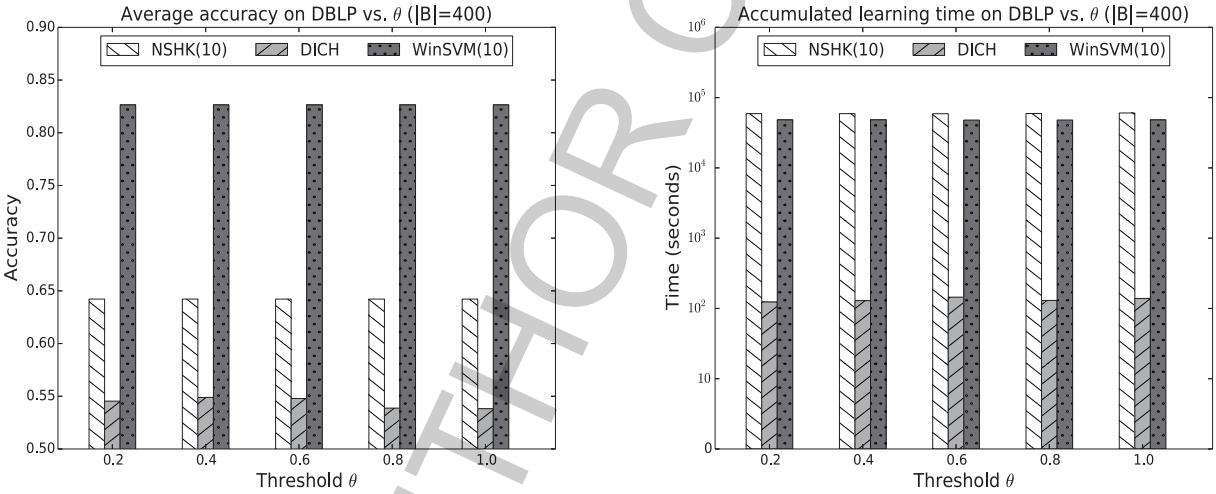
To investigate the impact of our entropy-based subgraph extraction scheme on performance, we set $|B| = 200$ for IMDb and $|B| = 400$ for DBLP and HEPTH, and report average classification accuracy and system runtime for these comparison algorithms by using different thresholds θ for extracting subgraphs. Figures 16–18 plot the average classification accuracy results across all batches and accumulated system runtime w.r.t. θ on IMDb, DBLP and HEPTH.

We observe from Figs 16(a) and 18(a) that the classification accuracy is sensitive to θ on IMDb and HEPTH. 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



(a) Average accuracy (b) Accumulated learning time

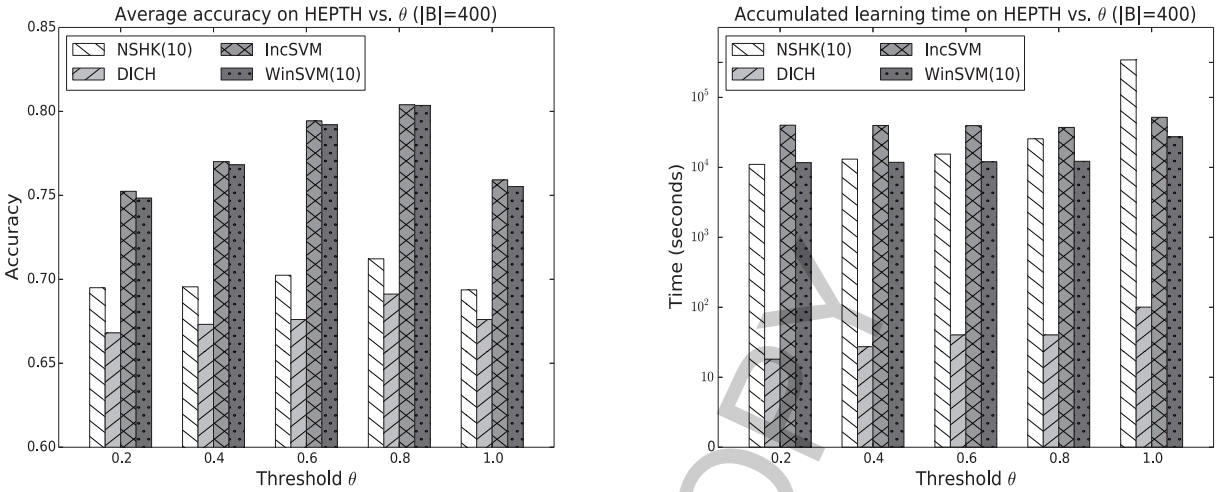
Fig. 16. Average accuracy and accumulated learning time across all batches on IMDb w.r.t. different values of θ.



(a) Average accuracy (b) Accumulated learning time

Fig. 17. Average accuracy and accumulated learning time across all batches on DBLP w.r.t. different values of θ.

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 the other three. Figures 16(b) and 18(b) show that the proposed subgraph extraction method can reduce accumulated learning time for all the four algorithms by setting $\theta < 1.0$. 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 has been observed on IMDb and HEPTH. 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



(a) Average accuracy (b) Accumulated learning time

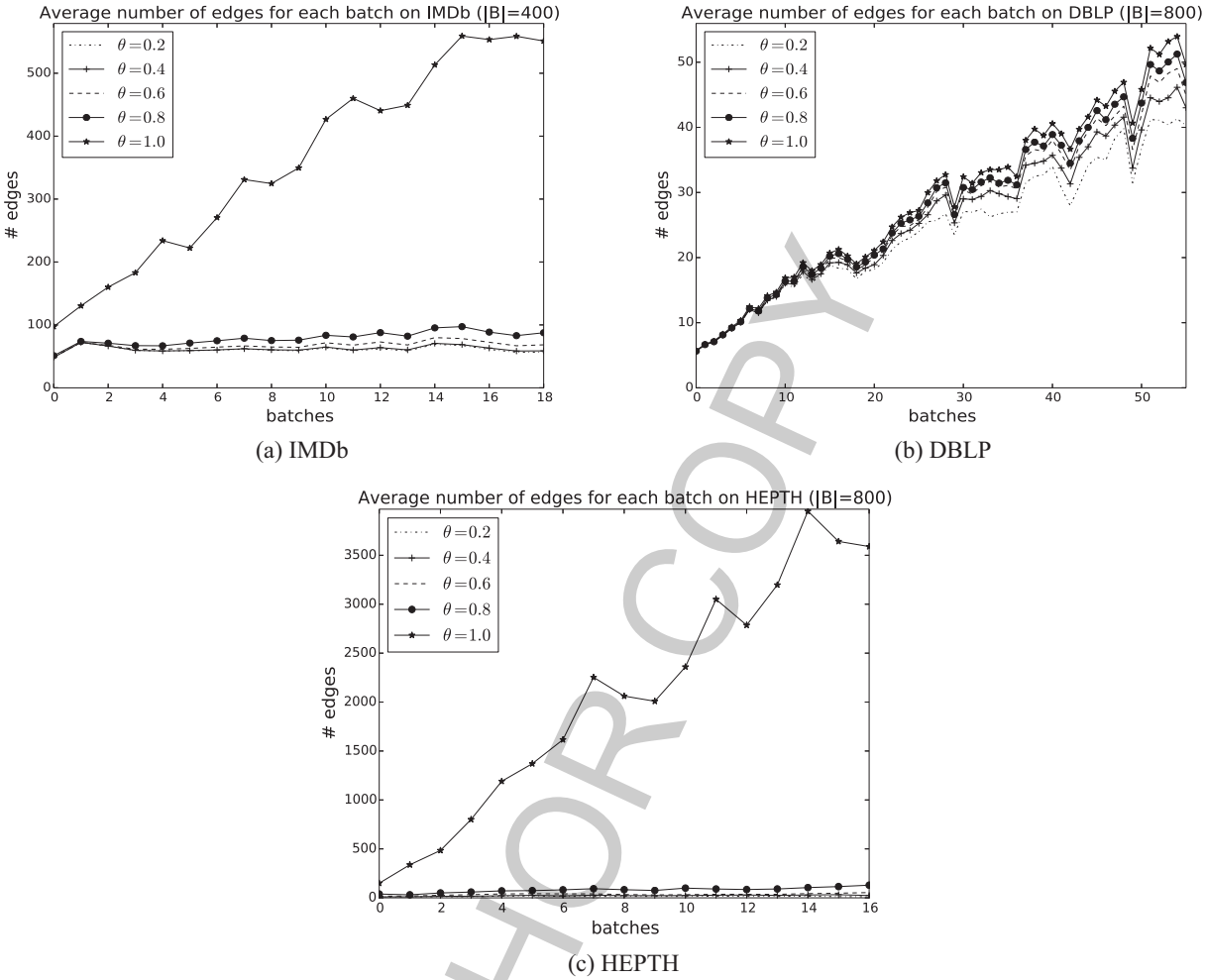
Fig. 18. Average accuracy and accumulated learning time across all batches on HEPTH w.r.t. different values of θ.

of runtime. These experimental results demonstrate clear benefits of our subgraph extraction method in terms of classification effectiveness and efficiency.

Another aspect for studying the impact of using different θ values is to investigate the structural complexity of the extracted subgraphs. As we mentioned in Section 3.1, the runtime of W-L graph kernel scales linearly in the number of edges. We use the average numbers of edges for subgraphs at each batch to measure the structural complexity, which also reflect the computational cost for the proposed incremental framework (see Section 4.4). From Fig. 19(a), 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 is structural complexity. Combining the results from Figs 16 and 19(a), we can see that our entropy-based subgraph extraction strategy is capable of retaining informative neighbor nodes and filtering irrelevant ones while inducing subgraphs for central entities in IMDb. A similar result can be inferred for HEPTH through the plots in Figs 18 and 19(c). On the other hand, Fig. 19(b) 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. 17.

5.3.3. Impact of window size

In this 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 $\theta = 0.8$ for performing the subgraph extraction process on IMDb, DBLP and HEPTH, and vary the window size $W = \{6, 8, 10\}$. We report the average classification accuracy across all batches in Fig. 20. 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. Figure 21 demonstrates this fact. Overall, the results show that WinSVM can achieve impressive classification performance with a proper window



(a) IMDb (b) DBLP

(c) HEPTH

Fig. 19. Average number of edges at each batch w.r.t. different values of θ. The number of subgraphs in each batch: $|B| = 400$ for IMDb, $|B| = 800$ for DBLP and HEPTH.

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.

5.4. Discussion

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 observed in the experiments of DBLP, IncSVM will fail to scale well on some 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.

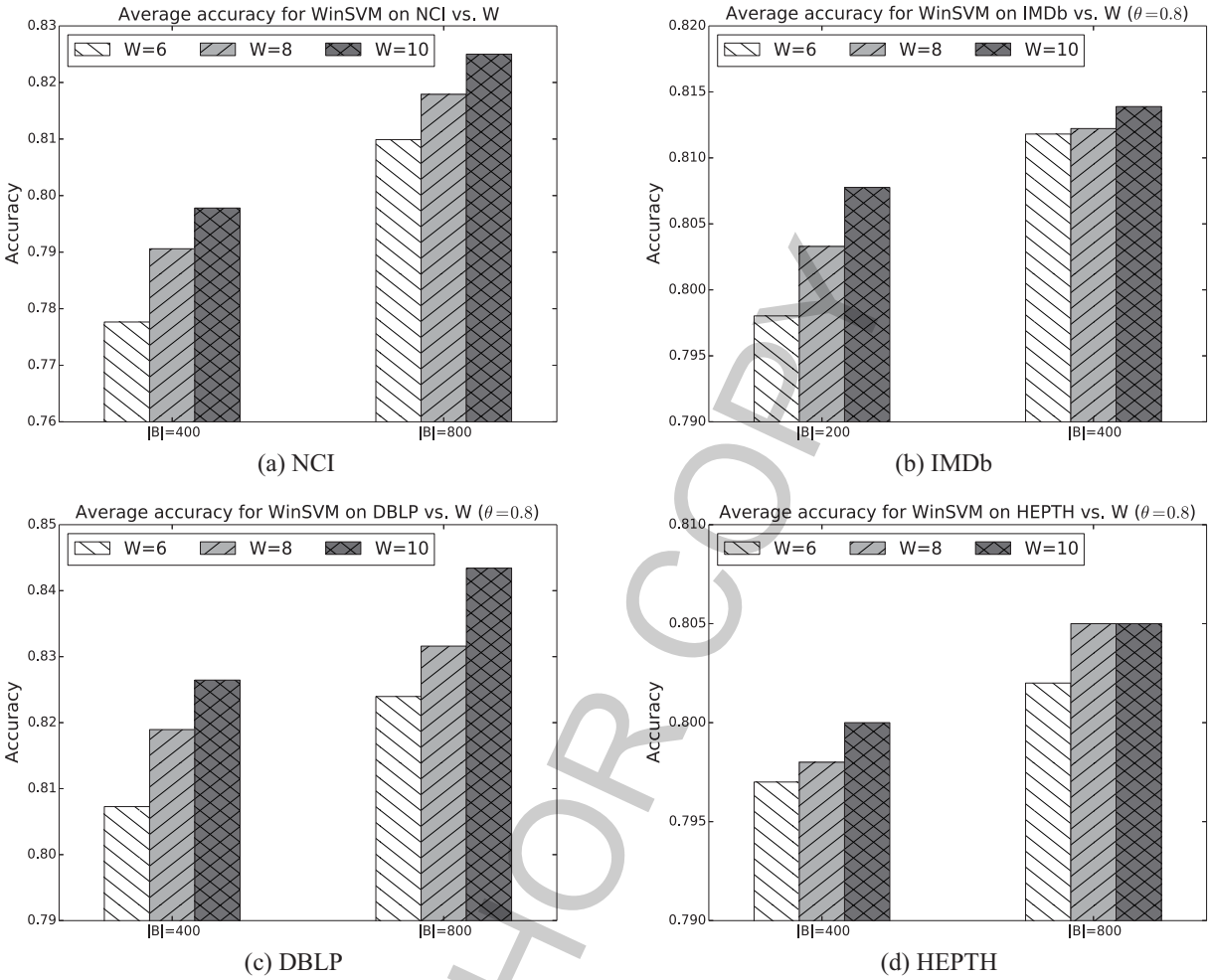


Fig. 20. Average classification accuracy across all batches w.r.t. different window sizes for WinSVM.

The proposed methodology is a general framework combining a fast graph kernel and an SVM. It is essential to note that the framework is independent from the types of graph kernels. In other words, any existing graph kernel can be incorporated into our framework to facilitate the classification in a dynamic streaming network. In this work, we have chosen the W-L kernel due to the fact that it scales linearly w.r.t. the number of edges, which makes it more efficient compared to other classical graph kernels such as random walk-based kernel and graphlet-based kernel.

For the NCI graphs, it is possible that the data contains noisy information. Since NCI is an isolated-graph scenario, SubExtract cannot be applied on this dataset. Using graph mining techniques to find discriminative subgraph features seems to be a promising solution [20] for eliminating noisy data. Nevertheless, we have ignored this situation in our current experiments, but it is an important problem that we will explore as one of our future tasks. Also, in our experiments, we assume that the granularity level of the data remains unchanged over time. However, in many real domains, the data granularity changes frequently as time progresses. One approach is to treat this as the change of data representation, and rebuild a learning model in response to that change. In future work, we will explore more efficient methods for reacting to granularity change.

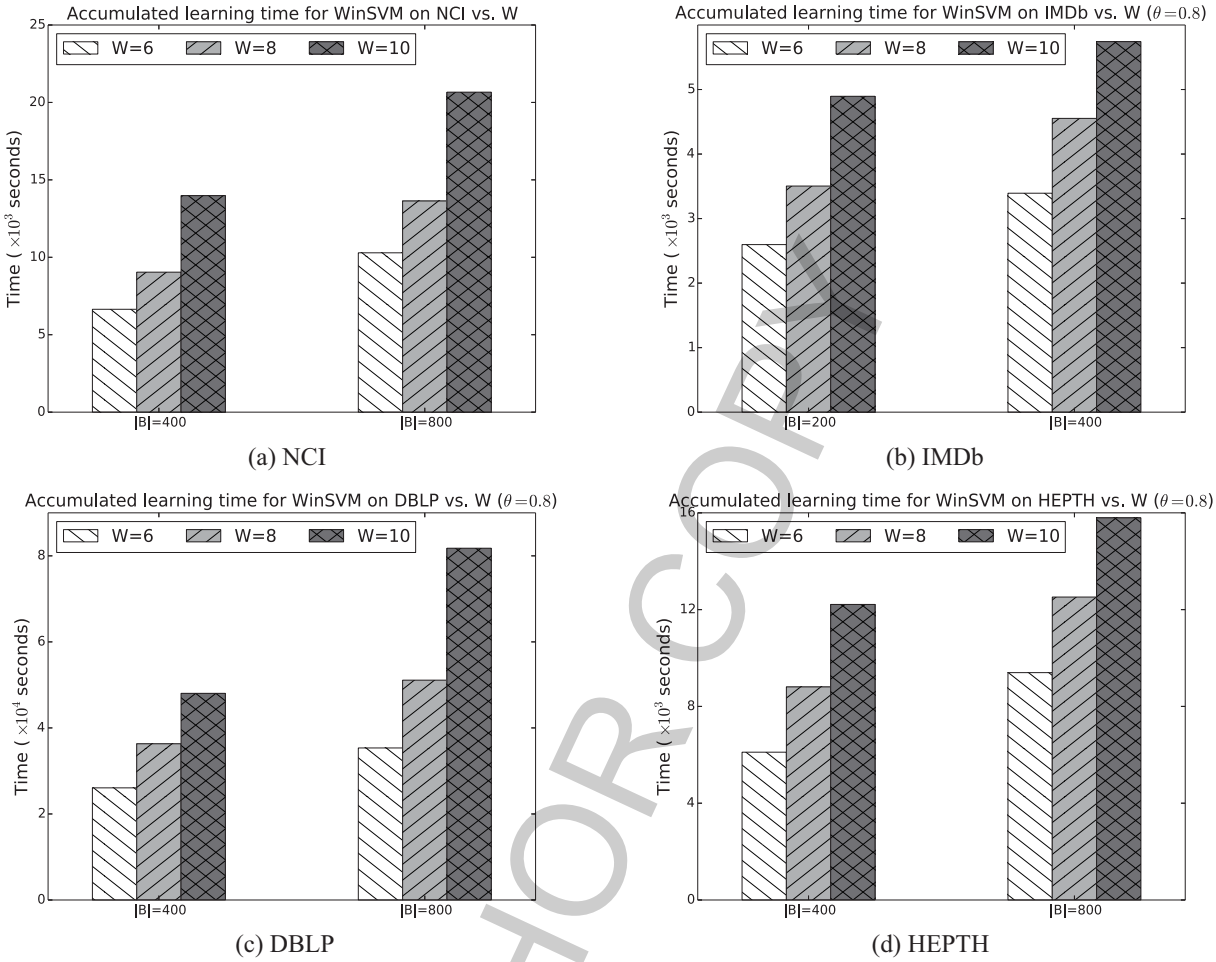


Fig. 21. Accumulated system runtime across all batches w.r.t. different window sizes for WinSVM.

6. Conclusions and future work

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 graph kernel to train a classification model and continuously 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 problem of granularity change in data and the problem of noisy information in the isolated-subgraph classification scenario. We will also explore the theoretical relation between the user-defined variables (i.e., window size W, subgraph extraction threshold θ) and the classification performance of the proposed algorithms. Developing algorithms that can be applied to a dynamic graph which is subject to other types of updates (e.g., deletion of nodes and edges) is another future direction.

Acknowledgment

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

References

[1] C.C. Aggarwal, On classification of graph streams, in: *Proceedings of SIAM International Conference on Data Mining*, SIAM (2011), 652–663.
[2] C.C. Aggarwal and N. Li, On node classification in dynamic content-based networks, in: *Proceedings of SIAM International Conference on Data Mining*, SIAM (2011), 355–366.
[3] A. Azran, The rendezvous algorithm: Multiclass semi-supervised learning with markov random walks, in: *Proceedings of the 24th International Conference on Machine Learning*, ACM (2007), 49–56.
[4] S. Bhagat, G. Cormode and S. Muthukrishnan, Node classification in social networks, in: *Social Network Data Analytics*, Springer (2011), 115–148.
[5] K.M. Borgwardt and H.-P. Kriegel, Shortest-path kernels on graphs, in: *Proceedings of IEEE International Conference on Data Mining*, IEEE (2005), 74–81.
[6] K.M. Borgwardt, N.N. Schraudolph and S. Vishwanathan, Fast computation of graph kernels, in: *Advances in Neural Information Processing Systems*, (2006), 1449–1456.
[7] C.-C. Chang and C.-J. Lin, Libsvm: A library for support vector machines, *ACM Transactions on Intelligent Systems and Technology* **2**(3) (2011), 27.
[8] 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), 225–236.
[9] D.J. Cook and L.B. Holder, *Mining Graph Data*, John Wiley & Sons, 2006.
[10] C. Domeniconi and D. Gunopulos, Incremental support vector machine construction, in: *Proceedings of IEEE International Conference on Data Mining*, IEEE (2001), 589–592.
[11] P. Domingos and G. Hulten, Mining high-speed data streams, in: *Proceedings of ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, ACM (2000), 71–80.
[12] T. Gärtner, P. Flach and S. Wrobel, On graph kernels: Hardness results and efficient alternatives, in: *Learning Theory and Kernel Machines*, Springer (2003), 129–143.
[13] J. Gehrke, P. Ginsparg and J. Kleinberg, Overview of the 2003 kdd cup, *ACM SIGKDD Explorations Newsletter* **5**(2) (2003), 149–151.
[14] H. Kashima, K. Tsuda and A. Inokuchi, Marginalized kernels between labeled graphs, in: *Proceedings of International Conference on Machine Leraning* **3** (2003), 321–328.
[15] 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).
[16] X. Kong, W. Fan and P.S. Yu, Dual active feature and sample selection for graph classification, in: *Proceedings of ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, ACM (2011), 654–662.
[17] B. Li, X. Zhu, L. Chi and C. Zhang, Nested subtree hash kernels for large-scale graph classification over streams, in: *Proceedings of IEEE International Conference on Data Mining*, IEEE (2012), 399–408.
[18] 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, New York, USA, 2004.
[19] J. Neville, D. Jensen, L. Friedland and M. Hay, Learning relational probability trees, in: *Proceedings of ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, ACM (2003), 625–630.
[20] S. Pan and X. Zhu, Graph classification with imbalanced class distributions and noise, in: *Proceedings of the Twenty-Third International Joint Conference on Artificial Intelligence*, AAAI Press (2013), 1586–1592.
[21] S. Pan, X. Zhu, C. Zhang et al., Graph stream classification using labeled and unlabeled graphs, in: *Proceedings of IEEE International Conference on Data Engineering*, IEEE (2013), 398–409.
[22] J. Ramon and T. Gärtner, Expressivity versus efficiency of graph kernels, in: *First International Workshop on Mining Graphs, Trees and Sequences*, (2003), 65–74.
[23] N. Shervashidze and K.M. Borgwardt, Fast subtree kernels on graphs, in: *Advances in Neural Information Processing Systems*, (2009), 1660–1668.
[24] N. Shervashidze, T. Petri, K. Mehlhorn, K.M. Borgwardt and S. Vishwanathan, Efficient graphlet kernels for large graph comparison, in: *Proceedings of International Conference on Artificial Intelligence and Statistics*, (2009), 488–495.
[25] N. Shervashidze, P. Schweitzer, E.J. Van Leeuwen, K. Mehlhorn and K.M. Borgwardt, Weisfeiler-lehman graph kernels, *The Journal of Machine Learning Research* **12** (2011), 2539–2561.

[26] N.A. Syed, S. Huan, L. Kah and K. Sung, Incremental learning with support vector machines, in: *Proceedings of International Joint Conference on Artificial Intelligence*, Citeseer (1999).
[27] S.V.N. Vishwanathan, N.N. Schraudolph, R. Kondor and K.M. Borgwardt, Graph kernels, *The Journal of Machine Learning Research* **11** (2010), 1201–1242.
[28] B. Weisfeiler and A. Lehman, A reduction of a graph to a canonical form and an algebra arising during this reduction, *Nauchno-Technicheskaya Informatsia* **2**(9) (1968), 12–16.
[29] Y. Yao and L. Holder, Scalable svm-based classification in dynamic graphs, in: *Proceedings of IEEE International Conference on Data Mining*, IEEE (2014), 650–659.
[30] D. Zhou, O. Bousquet, T.N. Lal, J. Weston and B. Schölkopf, Learning with local and global consistency, *Advances in Neural Information Processing Systems* **16**(16) (2004), 321–328.
[31] X. Zhu, Z. Ghahramani, J. Lafferty et al., Semi-supervised learning using gaussian fields and harmonic functions, in: *ICML* **3** (2003), 912–919.
[32] Y. Zhu and D. Shasha, Statstream: Statistical monitoring of thousands of data streams in real time, in: *Proceedings of the 28th International Conference on Very Large Data Bases*, VLDB Endowment (2002), 358–369.