Classification in Dynamic Streaming Networks

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

***Abstract*—Traditional network classification techniques will become computationally intractable when applied on a network which is presented in a streaming fashion with continuous updates. In this paper, we examine the problem of classification in dynamic streaming networks, or graphs. Two scenarios have been considered: the *graph transaction* scenario and the *one large graph* scenario. We propose a unified framework consisting of three components: a subgraph extraction method, an online version of an existing graph kernel, and two kernel-based incremental learners. We demonstrate the advantages of our framework via empirical evaluations on several real-world network datasets.**

I. Introduction

In recent years, network data has gained much popularity due to a great amount of information becoming available in the form of social networks, hyper-linked websites, chemical compounds, and so on. A research focus has been to extend traditional machine learning techniques to perform supervised classification tasks on these network, or graph, datasets. A detailed investigation of various graph mining and classification algorithms as well as their applications can be found in [1].

Most graph classification approaches use kernel machines, which compute similarities between graphs by mapping them to vectors in a high dimensional space and then computing their inner products. Conventionally, graph kernels assume that the graph data is limited in size and can be stored in memory or local storage, which allows the methods to scan the data multiple times within reasonable time constraints. However, this assumption has been violated since the recent growth in the sizes of real-world graphs. These graph datasets are measured in terabytes and heading toward petabytes. Furthermore, they are generated in a streaming fashion with frequent updates (e.g., insertions/deletions/modifications of nodes/edges). For example, social networks are continuously formed by the increasing social interactions among entities. Due to the dynamic nature of those networks, graph classification methods which involve enumerating substructures (e.g., graph kernel methods) are incapable of calculating similarities effectively between graphs for the following reasons.

- It is impossible to hold all information in memory or local storage. In order to maintain a modest speed of accessibility, old information must be removed from storage, which makes it infeasible to compute the global kernel matrix.
- With the increasing volume of nodes and edges streamed in, enumerating the substructures (e.g., paths, subtrees) for graph kernels and computing their kernel matrices will become extremely expensive or even impractical.
- The existence of noisy information may deteriorate the classification performance or introduce unnecessary structural complexity during the learning progress.

In order to address the above challenges, we propose a unified framework to investigate the problem of classification in dynamic streaming graphs. It encompasses two classification scenarios. In the *graph transaction* scenario, independent graphs are being received through a stream 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. We design an entropy-based scheme to extract a subgraph surrounding the node to be classified. Through the extraction step, we transform the classification of nodes in a single large graph into the classification of the extracted independent subgraphs, thus reverting to the graph transaction scenario. We then propose an online version of an existing fast graph kernel, namely Weisfeiler-Lehman [2], to enable the kernel computation to be performed in an online fashion rather than the traditional batch mode. Once the kernel values are obtained, we propose two incremental classification techniques based on SVM and perceptron, which take the kernel values as inputs and predict the class memberships of the independent graphs (in the graph transaction case) or the target nodes (in the one large graph case). The framework in this paper addresses the novel task to learn a classification model incrementally on large streaming graphs. In our previous work [3]–[5], we proposed an SVM-based and a perceptron-based learners for classifying large dynamic streaming networks. In this paper, we put these two learning models in one unified framework and provide more experimental evaluation to show the advantages of them.

The specific contributions of this paper are:

- Develop an online version of an existing graph kernel and make it capable of computing kernel values for a graph stream.
- Combine the online graph kernel and two kernel-based learning methods (SVM and perceptron) to learn classification models for streaming graphs.

The rest of this paper is organized as follows: Section II introduces some related research work. Section III formulates

IEEE/ACM ASONAM 2016, August 18-21, 2016, San Francisco, CA, USA 978-1-5090-2846-7/16/$31.00 ©2016 IEEE

the problem definition. The unified learning framework is presented in Section IV. In Sections V and VI, we establish the experiment settings and evaluate the proposed learning methods. Section VII concludes the paper.

II. Related Work

A. Graph Kernels

Traditional machine learning algorithms assume that data instances are represented in numerical vectors. However, this assumption makes the algorithms incapable of catching the relations among entities for graph data and thus fail to classify the data effectively. Graph kernels provide an elegant solution to classifying graph data by implicitly mapping them into a high-dimensional space and then computing the inner products. Most graph kernels aim to calculate similarities between graphs by enumerating their common substructures like random walks [6], shortest-paths [7], subtrees [8], graphlets [9], and frequent subgraphs [10]. The recently proposed Weisfeiler-Lehman (WL) kernel [2] is a fast subtree kernel whose runtime scales linearly in the number of edges of the graphs. It counts the matching multiset labels of the entire neighborhood of each node up to a given distance h. Because of its superiority in time complexity, we use the WL kernel in our framework and combine it with two kernel-based learning algorithms to facilitate the classification in dynamic graphs. We propose an online version of the WL kernel and make it capable of computing kernel values incrementally.

B. Graph Stream Classification

With the emergence of streaming data, there also exists some work on supervised classification techniques for dynamic streaming graphs. Aggarwal and Li [11] 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 [12], 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.

Li et al. [14] use their presented Nested Subtree Hash (NSH) algorithm based on the Weisfeiler-Lehman 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. The aforementioned methods aim to find common feature (subtree or clique) patterns across the data stream and map this increasing number of patterns onto a lower dimensional feature space using random hashing techniques. However, they 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 applicable to a single large-scale dynamic graph without a subgraph extraction process.

Yang et al. [15] propose an active learning approach to classification in streaming networked data, which reduces the need for classified instances. They also introduce a network sampling strategy that removes data minimizing the loss function when exceeding a reservoir size. Their network model is constrained to relationships between feature-vector instances to be classified; whereas our approach does not constrain the types of edges.

III. Problem Formulation

A. Preliminaries

Definition 1: A **labeled graph** is represented by a 4-tuple, i.e., $G = (\mathcal{V}, \mathcal{E}, \mathcal{L}, l)$, where (1)$\mathcal{V} = \{v_1, \dots, v_{|\mathcal{V}|}\}$ is a set of nodes, (2)$\mathcal{E} \subseteq \mathcal{V} \times \mathcal{V}$ is a set of (directed/undirected) edges, (3)$\mathcal{L}$ is a set of labels, and (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)$\mathcal{E} \subseteq \mathcal{E}'$, and (4)$\forall (u, v) \in \mathcal{E}, l(u, v) = l'(u, v)$.

Definition 3: A **subtree** is a subgraph of a graph, with a designated root node but no cycles. The **height** of a subtree is the maximum distance between the root and any leaf node in the subtree.

Definition 4: Let G be a graph and $v \in V$ be a node. The **neighborhood** of v, denoted $\mathcal{N}(v)$, is the set of nodes to which v is connected by an edge in $\mathcal{E}$, i.e., $\mathcal{N}(v) = \{u|(u, v) \in \mathcal{E}\}$.

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. In this paper, we focus on streaming networks with only insertions of nodes or edges.

Definition 5: An **update** (denoted by U_t) on a graph is an operation that inserts a node or an edge into the graph at time t. A **batch** (denoted by B_t) is a set of updates that are applied to the graph at time t, i.e., $B_t = \{U_t^1, U_t^2, \cdots, U_t^{n_t}\}$, where n_t is the number of updates in the tth batch.

Definition 6: A **dynamic streaming graph**, denoted $\mathcal{G}$, is formed by batches of updates $\mathcal{G} = \bigcup_{t=1}^{\infty} B_t$. Here, t denotes the timestamp of the corresponding update received by the graph. Typically, $\mathcal{G}^t = \bigcup_{i=1}^{t} B_i$ denotes the graph at time t, and $\mathcal{G}^0 = \emptyset$ represents the initial empty graph.

A real-world network usually consists of different types of nodes and their relationships (e.g., a paper-author network). Users are particularly interested in categorizing a certain type of node (e.g., paper nodes in a paper-author network) with help from the other types of nodes in the graph (e.g., author nodes in a paper-author network).

Definition 7: A node to be classified is defined as a **central node**, which denotes an entity that is going to be classified from the original data, and a node is defined as a **side node** if it is not a central one.

B. Problem Definition

We now formulate the problem of classification in a dynamic streaming graph. The following two scenarios are considered in our study.

- **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 $t-1$, and to predict the class membership of any new central nodes arriving at time t. See Fig. 1 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 $t-1$, and to predict the class labels of the instances in the next batch. See Fig. 2 for an example. This scenario is also referred to as the graph transaction scenario.

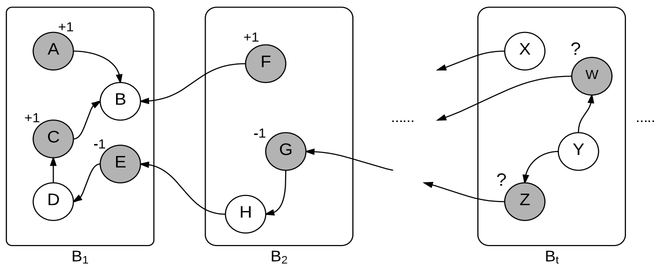


Fig. 1. Central node classification. The shaded nodes denote the central nodes of interest, and the symbols ± 1 indicate their class labels. B_t denotes the batch of updates received at time t.

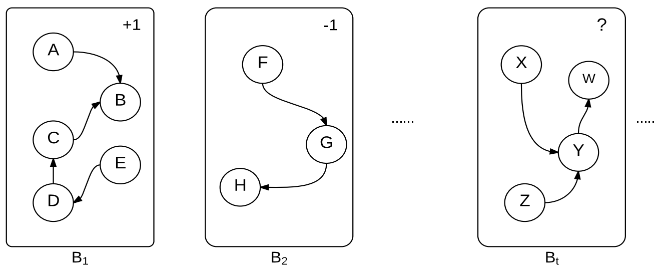


Fig. 2. Isolated-graph classification. The symbols ± 1 indicate these isolated graphs' class labels. B_t represents the set of updates received at time t, which contains a small graph to be streamed into the underlying large graph.

IV. Framework

A. Subgraph Extraction for Central 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 lose discriminative information for classification, thus deteriorating the classification performance.

We design an effective strategy to extract a subgraph for the node 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:

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

The entropy computation for v_i can be explicitly written as:

$$EN(v_i) = -p_1 \log_2 p_1 - p_2 \log_2 p_2 \qquad (2)$$

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 $EN(v_i)$ is, the more discriminative power v_i has.

Algorithm 1 Subgraph Extraction (SubExtract)

```
Input: $\mathcal{G}$: A graph
       $v_c$: A target central node
       $\theta$: A threshold for selecting discriminative nodes
Output: $Sub_{v_c}$: A subgraph surrounding $v_c$
 1: $I(v_c) = \{v_c\}$
 2: $N_v = \mathcal{N}(v_c)$
 3: while $N_v \neq \emptyset$ do
 4:     pop a node $v'$ from $N_v$
 5:     if $v'$ is not visited then
 6:         compute the entropy $EN(v')$
 7:         if $EN(v') \leq \theta$ then
 8:             $I(v_c) = I(v_c) \cup \{v'\}$
 9:             mark $v'$ as visited in $\mathcal{G}$
10:             if $v'$ is not the same type as $v_c$ then
11:                 $N_v = N_v \cup \mathcal{N}(v')$
12: induce a subgraph $Subg_{v_c}$ from $\mathcal{G}$ with the nodes in $I(v_c)$
```

To obtain a subgraph for a target central node to be classified, denoted as v_c, an entropy 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 with 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. Algorithm 1 shows the detailed procedure for extracting a subgraph surrounding a central node from a graph.

The 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. It 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.

B. Online WL Kernel

Almost all graph kernel methods fail to scale efficiently on data streams with unlimited number of graphs due to the following drawbacks [14]: (1) the feature space keeps expanding with emerging subgraph patterns when new graphs are continuously fed in; (2) the memory and runtime to find the desired subgraph patterns become prohibitive as the size of the graph set grows; (3) multiple scans are required in order to calculate the global kernel matrix for all graphs seen so far.

The Nested Subtree Hashing (NSH) kernel [14] addresses the above issues. At each iteration, NSH uses a random hash function, $h : str \to \mathbb{N}$, to project the strings representing subtree patterns onto a set of common low-dimensional feature spaces. The function takes a string str and an integer d as inputs, and maps str to a random integer which is no greater than d. They prove that their NSH kernel is an unbiased estimator of the original WL kernel. One parameter for the NSH kernel is the hashing dimension $D = \{d^0, \cdots, d^R\}$, where d^i represents the size of the subtree feature space at the ith iteration. All subtree patterns encountered at the ith iteration will be mapped to integers no greater than d^i. The output of the NSH kernel is a multi-resolution feature vector set $\{\mathbf{x}^i\}_{i=0}^R$, where $\mathbf{x}^i$ is a d^i-length vector and records the occurrences of subtree patterns encountered at the ith WL test.

Algorithm 2 Online WL (OWL)

```
Input: $\mathcal{D} = \{G_1, \cdots, G_j, \cdots\}$: a stream of graphs
       $D = \{d^0, \cdots, d^R\}$: the dimensions for hashing
Output: $K$: a kernel matrix
1: Initialize $\mathcal{F} = \emptyset$
2: for $t = 1, \cdots, j, \cdots$ do
3:     $\{x^i\}_{i=0}^R = \text{NSH}(G_t, D)$
4:     $\mathbf{x}_t = vec(\{x^i\}_{i=0}^R)$ [1]
5:     $\mathcal{F} = \mathcal{F} \cup \{\mathbf{x}_t\}$
6:     for $n = 1$ to $t - 1$ do
7:         $\mathbf{x}_n = \mathcal{F}(n)$
8:         $K(G_t, G_n) = \langle \mathbf{x}_t, \mathbf{x}_n \rangle$
```

One merit of the NSH algorithm is that the size of the feature space at each iteration is bounded. In this way, we are able to manage the emerging subtree patterns at each iteration and map them onto a fixed-size space. We no longer need to find the global set of distinct subtree patterns from all the graphs at each iteration. Based on the NSH kernel, we propose an online version of the WL kernel, namely OWL, which can be applied for calculating the similarity between graphs from a data stream in an online mode. The approach is that: once we receive a graph from the data stream, we call the NSH method to convert that graph into a set of feature vectors in which every feature vector records the numbers of occurrences of subtree patterns at the corresponding iteration. The similarity between graph G_i and all the previous graphs $\{G_1, \cdots, G_{i-1}\}$ can then be calculated by the inner products between G_i's feature vectors and G_j's ($j < i$) feature vectors. Since every graph is characterized by a set of feature vectors, we can store those vectors in memory and discard the graph itself. This is time-efficient since there is no need to fetch the original graph again and scan it for multiple iterations to obtain the subtree patterns. And it is also memory-efficient by limiting the bucket size for hashing, thus to avoid unboundedly emerging subtree patterns. Algorithm 2 shows the pseudocode of OWL.

[1]We define $vec(\cdot)$ as the vectorized operator which concatenates all the row vectors or column vectors together to form a larger row vector or column vector. $vec(\{x^i\}_{i=1}^R) = [x^1, \ldots, x^R]$ forms a new row vector with length $d_1 + \cdots + d_R$, where x^i is also a row vector and $|x^i| = d_i$.

C. Incremental Classifiers

1) Incremental SVM: Since graph kernel methods with SVM have shown good performance when classifying static graph-structured data, we combine an incremental SVM with the OWL 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 a new testing set is received, OWL is called to compute the kernel values between the training set and the testing set. Once the computation is done, the similarity matrix is fed into the SVM [16] algorithm.

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 WL 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 3 shows the pseudocode of the incremental SVM (**IncSVM**) method.

2) Online Perceptron: Despite its simplicity, the perceptron has produced good results in many real-world applications. And it becomes especially effective when combined with kernels due to the benefits generated by the kernel trick. A kernel-based perceptron can be written as a kernel expansion

$$f(G) = \sum_{i \in S_t} y_i K(G_i, G) \tag{3}$$

Algorithm 3 Incremental SVM (IncSVM)

```
Input: $\mathcal{G} = \{B_1, \cdots, B_t, \cdots\}$: a dynamic streaming graph
       $\theta$: the threshold for extracting subgraphs
Output: Classification of a batch $B_i (i = 1, \cdots, t, \cdots)$
1:  $\mathcal{M}_0 = 0$ (the initial classification model)
2:  $SV_0 = \emptyset$ (the initial support vector set)
3:  $TR_0 = \emptyset$ (the initial training set)
4:  $\mathcal{G}^0 = \emptyset$ (the initial graph structure with no nodes or edges)
5:  for $i = 1, \cdots, t, \cdots$ do
6:      $GB_i = \emptyset$
7:      if central node classification then
8:          $\mathcal{G}^i = \mathcal{G}^{i-1} \cup B_i$
9:          for each central node $v_c$ in $B_i$ do
10:             $GB_i = GB_i \cup SubExtract(\mathcal{G}^i, v_c, \theta)$
11:     else
12:         $GB_i = B_i$
13:     compute kernel values between $GB_i$ and $TR_{i-1}$
14:     classify $GB_i$ using $\mathcal{M}_{i-1}$
15:     fetch the support vectors of $\mathcal{M}_{i-1}$ as $SV_{i-1}$
16:     construct a training set $TR_i = SV_{i-1} \cup GB_i$
17:     delete nodes and edges not included in $TR_i$ from $\mathcal{G}^i$
18:     learn a SVM classifier $\mathcal{M}_i$ on $TR_i$
```

Algorithm 4 Online Perceptron (OnPer)

```
Input: $\mathcal{G} = \{B_1, \cdots, B_t, \cdots\}$: a dynamic streaming graph
       $\theta$: the threshold for extracting subgraphs
Output: Classification of a batch $B_i (i = 1, \cdots, t, \cdots)$
1:  $SV_0 = \emptyset$ (the initial support vector set)
2:  $f_0 = 0$ (the initial classification model)
3:  $\mathcal{G}^0 = \emptyset$ (the initial graph structure with no nodes or edges)
4:  for $i = 1, \cdots, t, \cdots$ do
5:      $GB_i = \emptyset$
6:      if central node classification then
7:          $\mathcal{G}^i = \mathcal{G}^{i-1} \cup B_i$
8:          for each central node $v_c$ in $B_i$ do
9:              $GB_i = GB_i \cup SubExtract(\mathcal{G}^i, v_c, \theta)$
10:     else
11:         $GB_i = B_i$
12:     compute kernel values between $GB_i$ and $SV_{i-1}$
13:     $ERR_i = \emptyset$
14:     for each graph $G$ in $GB_i$ do
15:         $\hat{y} = \text{sgn}(f_{i-1}(G))$
16:         if $\hat{y} \neq y$ then
17:             $ERR_i = ERR_i \cup \{G\}$
18:     $SV_i = SV_{i-1} \cup ERR_i$
19:     delete nodes and edges not included in $SV_i$ from $\mathcal{G}^i$
20:     $f_i = \sum_{j \in S_i} y_j K(G_j, \cdot)$ (rebuild perceptron)
```

where $K(\cdot, \cdot)$ is the kernel value between two instances and S_t is the support set at time t. The prediction can be made by $\hat{y}_t = \text{sgn}(f(G_t))$. If the predicted result $\hat{y}_t$ is not equal to the true label y_t, G_t is added to the budget, that is, $S_t = \{i | \hat{y}_i \neq y_i, i < t\}$. Algorithm 4 lists the pseudocode of the online perceptron combined with OWL, namely **OnPer**. Unlike the incremental SVM, the online perceptron constrains memory usage by retaining only examples that fit within the budget.

D. *Window-based Incremental Classifier*

Although the incremental methods will potentially utilize less memory by discarding old data points which are not identified as support vectors, it is still possible that the algorithms will not scale up effectively when a large number of support vectors tend to be retained. In our research, we have adopted the easiest windowing scheme, i.e., fixed-size sliding window, to maintain a limited amount of graph data in memory. Using a sliding window enables us to keep the amount of data that arrives recently. When the window is full, the oldest batch of nodes or edges will be removed so that we can maintain the data as a moderate size in memory. In this way, we can always keep a set of instances which are closely related to the current instances in the time line, which makes the algorithm tolerant to potential concept drifts. At time t when the new batch B_t arrives, the oldest instances in the support set will be discarded if the window is full. As a result, the corresponding nodes and their associated edges will also be deleted from memory if they are not included in any existing instances in the budget. However, we allow the nodes which have been deleted to be re-inserted into the graph if new edges in B_t refer to them. The window-based versions of IncSVM and OnPer are referred to as **WinSVM** and **WinPer**.

V. Experiments

We apply our incremental classification algorithms on graph data from practical application domains. We evaluate the effectiveness of the entropy-based subgraph extraction method by setting different values for the entropy threshold. And we also validate the proposed incremental learning techniques on several real-world dynamic graph datasets by comparing them with two of the state-of-the-art algorithms.

A. *Benchmark Data*

(1) **DBLP Network:** DBLP[2] is a database containing millions of publications in computer science. Each paper is described by its abstract, authors, year, venue, title and references. Similar to the work in [17], 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 45,270 papers published between 2000 and 2009, and their references

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

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 3 plots the numbers of positive and negative papers published in each year between 2000 and 2009. Figure 4 shows a portion of the final DBLP network.

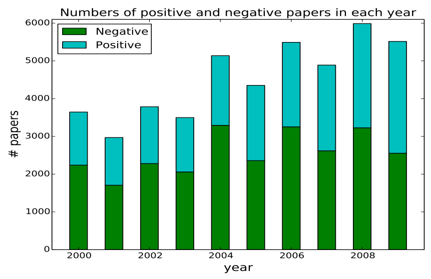


Fig. 3. The numbers of positive and negative examples in each year between 2000 and 2009 for DBLP dataset.

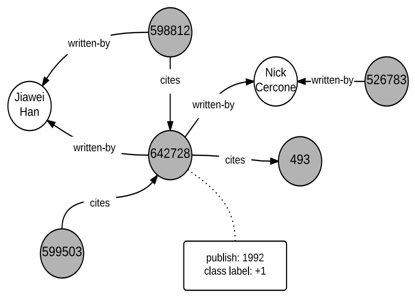


Fig. 4. A portion of the DBLP network. The shaded (central) nodes labeled with numeric IDs represent the papers of interest, while the non-shaded ones represent the authors. The label of the edge between two nodes indicates the relationship between them. The rectangular box describes the content stored in each paper node, including the publication year and the class membership.

(2) **NCI Graph Stream:** The National Cancer Institute (NCI) datasets[3] contain information on anticancer activities and are frequently used as a benchmark for graph classification [14]. 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 5 summarizes the five datasets. We concatenate these datasets sequentially and finally get a graph dataset with 6.5×10^5 nodes and 7.1×10^5 edges. This data falls under the isolated-graph classification scenario, where each graph is independent and isolated from the others.

[3]http://pubchem.ncbi.nlm.nih.gov

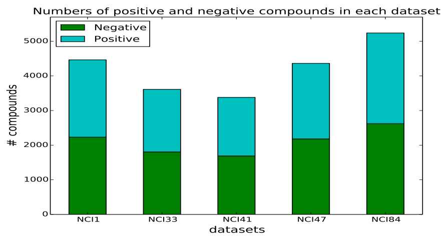


Fig. 5. Positive/negative chemical compounds in each cancer dataset.

B. Baseline Methods

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

1) Discriminative Clique Hashing (DICH): 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.

2) Nested Subtree Hash Kernel (NSHK): NSHK [14] is an ensemble learning framework which consists of W weighted classifiers built on the most recent W batches of data

$$f_E(\mathbf{x}) = \sum_{i=t-W+1}^{t} w_i f_i(\mathbf{x}) \tag{4}$$

where f_i is a classification model learned from batch B_i and

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

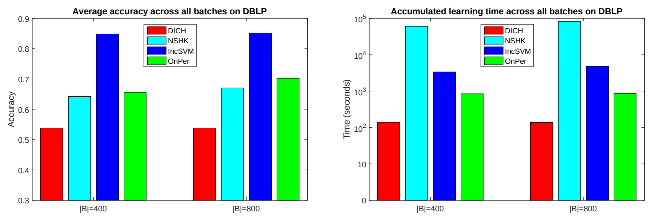
is the weight for f_i measured by the mean square errors related to f_i and the class distribution, and $P_t(y)$ denotes the class distribution in B_t. Each classifier of the ensemble is constructed using the WL kernel. During the WL isomorphism test, hashing techniques are used to map the unlimited subtree patterns into low-dimensional feature spaces.

Unless specified otherwise, we use the following settings for the parameters of our methods: batch size $|B_t| = \{400, 800\}$, window size $W = \{6, 8, 10\}$, and subgraph extraction threshold $\theta = \{0.2, 0.4, 0.6, 0.8, 1.0\}$.

VI. Evaluation

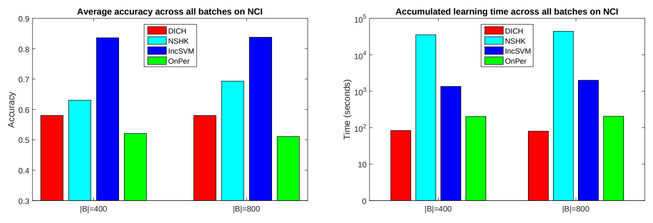
A. Batch Size

We first investigate the classification performance w.r.t. different batch sizes by setting the other three parameters to their default values. Fig. 6 and Fig. 7 show the accuracy values at different learning steps for the two datasets. We find that IncSVM consistently outperforms the two baseline methods on all the datasets. This result indicates that, by retaining the support vectors of a SVM built 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.



(a) Accuracy (b) Time

Fig. 6. Average accuracy and accumulated learning time across all batches on DBLP data w.r.t. different batch sizes.



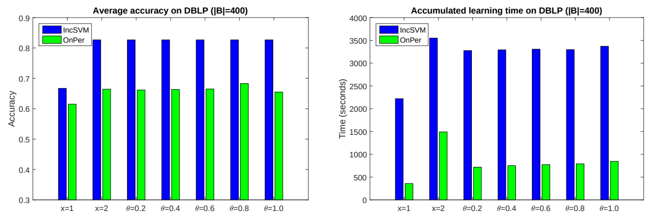
(a) Accuracy (b) Time

Fig. 7. Average accuracy and accumulated learning time across all batches on NCI data w.r.t. different batch sizes.

B. Entropy Threshold

To investigate the impact of our entropy-based subgraph extraction scheme on performance, we vary the subgraph extraction threshold $\theta = \{0.2, 0.4, 0.6, 0.8, 1.0\}$ for extracting subgraphs from DBLP, and report average classification accuracy and system runtime of IncSVM and OnPer. We also compare SubExtract with a naive subgraph extraction method, namely x-edge hop. The x-edge hop method extracts all nodes which are at most x edges away from the target central node and then induces a subgraph surrounding that central node. We have experimented with $x = 1$ and $x = 2$. The results are shown in Fig. 8.

For DBLP data, the naive extraction method with $x = 1$ generates a much lower accuracy than the other cases because it fails to extract more discriminative information compared to the case of $x = 2$ or SubExtract. However, it takes much less time to finish the learning process. Although the naive extraction method with $x = 2$ extracts more nodes and edges than SubExtract with $\theta = 1.0$, it has no superiority compared to SubExtract given that its learning time is longer. This indicates that including all information within two edges from a target central node will not improve the classification performance



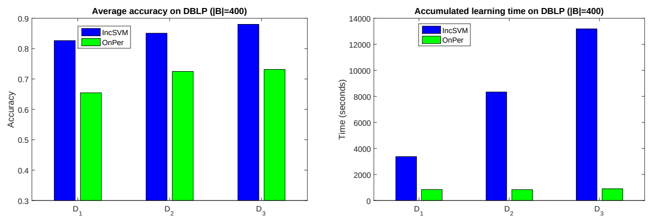
(a) Accuracy (b) Time

Fig. 8. Average accuracy and accumulated learning time across all batches on DBLP data w.r.t. different θs of SubExtract or different xs of the naive subgraph extraction.

on DBLP data. Overall, these experimental results demonstrate clear benefits of our subgraph extraction method in terms of classification effectiveness and efficiency.

C. Hashing Dimensions

To show how different hashing dimension settings affect classification error and runtime, we have chosen three sets of dimensions: $D_1 = \{500, 1000, 5000, 10000\}$, $D_2 = \{10000, 20000, 30000, 40000\}$, and $D_3 = \{50000, 60000, 70000, 80000\}$. Since more distinct subtree patterns will appear as the height of the subtrees increases, the value of dimensions increases for each dimension setting. Fig. 9 and Fig. 10 show the average classification accuracy and accumulated learning time for IncSVM and OnPer on the two datasets. An obvious finding is that the classification accuracy with large dimension values is superior to that with smaller dimension values for both IncSVM and OnPer. This is because large dimension values will avoid hash collision at a high probability while small dimension values bring a lot of hash collisions. As a result, the low dimension setting will cause much information loss deteriorating the classification accuracy. However, larger dimension values will take more time to compute the kernel values. Overall, these plots indicate that IncSVM and OnPer can scale well and achieve impressive classification results using a set of proper hashing dimensions.

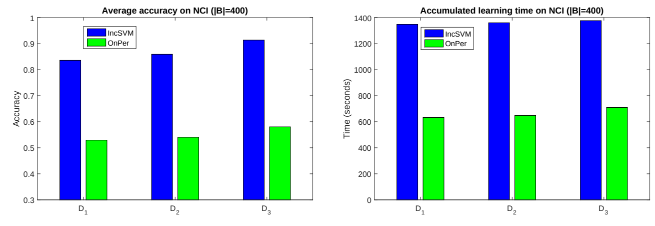


(a) Accuracy (b) Time

Fig. 9. Average accuracy and accumulated learning time across all batches on DBLP data w.r.t. hashing dimension settings.

D. Window Size

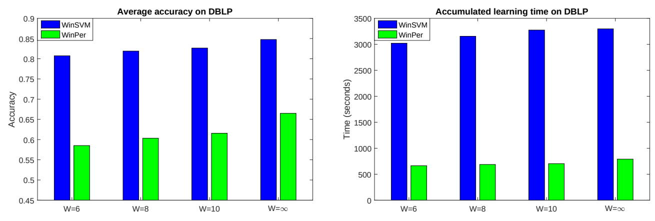
In this part, we investigate the classification performance w.r.t. different window sizes for the proposed window-based



(a) Accuracy (b) Time

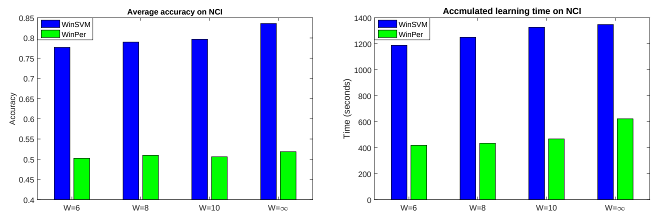
Fig. 10. Average accuracy and accumulated learning time across all batches on NCI data w.r.t. hashing dimension settings.

incremental learning techniques, namely WinSVM and WinPer. We vary the window size $W = \{6, 8, 10\}$ in our experiments, and also compare the classification performance of the window-based methods to IncSVM and OnPer (i.e., $W = \infty$). The average accuracy and accumulated learning time are plotted in Fig. 11 and Fig. 12. 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/WinPer to train a classifier. Overall, the results show that WinSVM/WinPer can achieve impressive classification performance with a proper window size.



(a) Accuracy (b) Time

Fig. 11. Average accuracy and accumulated learning time across all batches on DBLP data w.r.t. window sizes.



(a) Accuracy (b) Time

Fig. 12. Average accuracy and accumulated learning time across all batches on NCI data w.r.t. window sizes.

VII. Conclusion

We present a novel framework for studying the problem of classification in dynamic streaming graphs. The framework combines two incremental kernel-based methods and an online 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.

For future work, we will investigate the pros and cons of our incremental methods by conducting comparisons with state-of-the-art algorithms on more real-world dynamic networks, and explore the theoretical relationship between the user-defined variables (i.e., window size, edge extraction threshold) and the classification performance of the proposed algorithms. We would also like to investigate how to extend the proposed methods to study the problem of supervised learning on dynamic graphs with insertions, deletions and modifications of nodes/edges.

Acknowledgment

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

References

[1] D. J. Cook and L. B. Holder, *Mining graph data*. Wiley, 2006.
[2] 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.
[3] Y. Yao and L. Holder, "Scalable svm-based classification in dynamic graphs," in *ICDM*, 2014, pp. 650–659.
[4] ——, "Scalable classification for large dynamic networks," in *IEEE International Conference on Big Data*, 2015, pp. 609–618.
[5] ——, "Incremental svm-based classification in dynamic streaming networks," *Intelligent Data Analysis*, vol. 20(4), 2016.
[6] 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.
[7] K. M. Borgwardt and H.-P. Kriegel, "Shortest-path kernels on graphs," in *ICDM*, 2005, pp. 74–81.
[8] N. Shervashidze and K. M. Borgwardt, "Fast subtree kernels on graphs," in *NIPS*, 2009, pp. 1660–1668.
[9] N. Shervashidze, T. Petri, K. Mehlhorn, K. M. Borgwardt, and S. Vishwanathan, "Efficient graphlet kernels for large graph comparison," in *International Conference on AI and Statistics*, 2009, pp. 488–495.
[10] M. Deshpande, M. Kuramochi, N. Wale, and G. Karypis, "Frequent substructure-based approaches for classifying chemical compounds," *IEEE Transactions on Knowledge and Data Engineering*, vol. 17, no. 8, pp. 1036–1050, 2005.
[11] C. C. Aggarwal and N. Li, "On node classification in dynamic content-based networks." in *SDM*, 2011, pp. 355–366.
[12] C. C. Aggarwal, "On classification of graph streams," in *SDM*, 2011, pp. 652–663.
[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 *IEEE International Conference on Data Mining*, 2012, pp. 399–408.
[15] Z. Yang, J. Tang, and Y. Zhang, "Active learning for streaming networked data," in *CIKM*, 2014, pp. 1129–1138.
[16] C.-C. Chang and C.-J. Lin, "Libsvm: a library for support vector machines," *ACM Transactions on Intelligent Systems and Technology (TIST)*, vol. 2, no. 3, pp. 1–27, 2011.
[17] S. Pan, X. Zhu, C. Zhang, and P. S. Yu, "Graph stream classification using labeled and unlabeled graphs," in *ICDE*, 2013, pp. 398–409.