Incremental and Decremental Max-flow for Online Semi-supervised Learning

Abstract—Max-flow has been adopted for semi-supervised data modelling, yet existing algorithms were derived only for the learning from static data. This paper proposes an online max-flow algorithm for the semi-supervised learning from data streams. Consider a graph learned from labelled and unlabelled data, and the graph being updated dynamically for accommodating online data adding and retiring. In learning from the resulting non stationary graph, we augment and de-augment paths to update max-flow with a theoretical guarantee that the updated max-flow equals to that from batch retraining. For classification, we compute min-cut over current max-flow, so that minimized number of similar sample pairs are classified into distinct classes. Empirical evaluation on real-world data reveals that our algorithm outperforms state-of-the-art stream classification algorithms.

Index Terms—Online Semi-supervised Learning, Graph Mincuts, Max-flow, Augmenting path, Incremental Decremental Max-flow, Residual Graph.

1 Introduction

In big data era data volume and velocity increase fast. Labeling a small sample of data is the only feasible way to learn classification from real world big data. This results in that incoming data contain often a large portion of unlabelled instances. Semi-supervised learning is constructive in that unlabelled data can be utilized to facilitate machine learning and improve accuracy.

In semi-supervised learning, unlabelled data helps modify or reprioritize hypotheses obtained from labelled data alone [1], [2]. Provided with the same amount of labelled data, semi-supervised learning gives often better learning accuracy than supervised learning does when assumptions such as smoothness, cluster and manifold are met [3], [4]. On the other hand, labelled instances are often difficult, expensive, or time consuming to obtain in real world applications, as they require the efforts of experienced human annotators. In this sense, semi-supervised learning is capable of enhancing the learning effectiveness with less human efforts.

Graph mincuts [5] is a graph based semi-supervised learning algorithm featured by its non-parametric, discriminative, and transductive nature. The basic assumption of graph mincuts is simple but concrete: samples with smaller distance are more likely to be in the same class. This is called smoothness assumption in literature [2]. In graph mincuts, a graph is constructed from both labelled and unlabelled data according to samples closeness (i.e., similarity). For classification, min-cut is applied to split the graph into two isolated parts by removing an edge set with minimum total weight. Here, min-cut ensures that the number of similar sample pairs classified into distinct classes is minimized. Although graph mincuts shows good performance in learning from datasets with only small portion of labelled samples, graph mincuts, as a batch learner, can be only used for learning from static datasets.

Consider online learning of graph mincuts, a straightforward solution is to derive from an existing batch graph mincuts to its corresponding online version. Kumar [17] previously developed an online minicuts by updating preflow push operation. In contract, this work offers another option which is to perform online mincuts through incremental and decremental path augmentation. The graph learned from labelled and unlabelled data is updated dynamically for accommodating data adding and retiring. For learning such a non-stationary graph, proposed online max-flow addresses all possible graph changes in two categories: capacity decrease and increase on edges. Our algorithm de-augments paths to enable capacity decrease and augment paths after capacity increase to compute the up-to-date max-flow. In general, proposed online max-flow has a theoretical guarantee that updated max-flow is equal to that of batch retraining.

The rest of this paper is organized as follows: Section 2 reviews previous work on max-flow and semi-supervised learning. Section 3 introduces the batch graph mincuts system which is the fundamental of our work. Section 4 presents the proposed online max-flow algorithm, including proofs and derivations. Experiments and discussion are given in Section 5. Finally, we conclude this paper in Section 6.

For the convenience of algorithm derivation and clarity of presentation, we summarize most notations used in the paper in Table 1.

2 Related Work

2.1 Max-flow

In graph theory, Max-flow seeks a maximum feasible flow through a single-source, single-sink graph [6]. As various real world problems can be abstracted into max-flow problems or equivalent problems, there is a vast number of applications for max-flow. The bottleneck identification for city traffic network [7] is an known max-flow application in which traffic network in the city is abstracted into a road graph. Applying max-flow computation on the road graph, a road with its all capacity taken to carry flow is considered as a bottleneck. Similarly, bottleneck identification for power system has been used for computing power system security index [8] in which power supply links are represented as

TABLE 1
Notations

Notation	Descriptions
$\boldsymbol{X}$	instance matrix
$\boldsymbol{x}_i$	the i-th instance
L_+, L_-, U	index set of positive, negative, unlabelled samples
$\boldsymbol{D}$	distance matrix, $d_{i,j} = dist(\boldsymbol{x}_i, \boldsymbol{x}_j)$
G	weighted graph, $G = (V, E, C)$
V	node set
E	edge set
C	capacity set
u, v	node u, node v
s, t	source node, sink node
e	edge e
(u, v)	edge from node u to node v
$C(e)$, $C(u, v)$	capacity on edge e and (u, v)
R	residual graph, $R = (V, E, R)$
$R(e)$, $R(u, v)$	residual capacity on edge e and (u, v)
$f(u, v)$	flow value on edge (u, v)
F	net flow value, $F = \sum_{u \in V} f(u, t)$
P	path

edges, and factories or towns are denoted as nodes. Max-flow is also applied in wireless mobile environment to optimize the association between wireless clients to access points in order to maximize the traffic flow to clients [9] [10]. In addition, Max-flow/min-cut has been widely used in computer vision for image segmentation [11] [12] [13], stereo [14] [15] and shape reconstruction [16]. All above applications are associated with computing max-flow from a static graph.

In big data era, data is becoming available quickly in a sequential manner, which requires system to process data in real time. Towards the online learning of max-flow, Kumar et al. [17] proposed an incremental max-flow based on generic preflow push. Recall that batch max-flow has the solution of either preflow push or augmenting path [6]. Kumar in his work successfully updated the preflow push operation in response to unit edge capacity adding and reducing in a graph. To our best knowledge, this is the only previous work on online max-flow. The augmenting path based online max-flow is still left as an open question.

2.2 Semi-supervised Learning

Semi-supervised learning uses unlabelled data to promote learning accuracy. This impact works also for online learning. Here, we review briefly recent works with a focus on how unlabelled data is being used for modeling. In [18], unlabelled samples that have the same distributions with the target domain are utilized with labelled data to train a transfer semi-supervised SVM. Zhang et al [19] cluster unlabelled data, then ensemble the obtained clusters with classifiers built on labelled data to deal with the concept drift of data streams. In [20] unlabelled data are utilized to train a Gaussian mixture model which determines the voting weight for each weak classifier trained from labelled data received at different learning stage. Besides, there are also other ways for unlabelled data to be used in online semi-supervised learning [21], [22], [23], [24], [25], [26]. All above works set the ratio of unlabelled data within the range of 90% to 99%, but haven't yet addressed even higher ratio.

3 Batch Semi-supervised Learning via Graph Mincuts

Graph mincuts has been used for classification learning from both labelled and unlabelled data [5]. The idea is straightforward: close samples in feature space are more likely to be from the same class. Let $\boldsymbol{X}$ be a partially labelled dataset. We assume each sample of the dataset has a unique index, and L_+, L_- and U be the index set of positive, negative and unlabelled samples, respectively. Graph mincuts constructs a weighted graph G according to sample closeness (similarity) and then split G by removing an edge set with minimum total weight. For semi-supervised learning, min-cut ensures that a minimized number of similar sample pairs are classified into distinct classes.

3.1 Graph Construction

We consider learning a weighted graph G from data $\boldsymbol{X}$. A weight graph $G = (V, E, C)$ consists of a finite node set V, an edge set $E \in V \times V$, and a weight function $C : E \to R^+$ (called capacity hereafter) which associates a positive weight value $C(e)$ with each edge $e \in E$.

For creating node set V, each sample $\boldsymbol{x}_i$ either labelled or unlabelled is represented by a vertex v_i. To make the min-cut (max-flow) feasible, two virtual nodes v_+ and v_- are created for positive and negative class respectively. Thus $V = \{v_i, v_+, v_- | \forall i \in L_+ \cup L_- \cup U\}$.

For determining edges E and capacity C, two steps are taken to connect nodes in V:

Firstly, each labelled node is connected to the virtual node with the same class label, where the edge has an infinite weight as

$$\begin{aligned} C(v_i, v_+) &= \infty, \quad \forall i \in L_+ \\ C(v_i, v_-) &= \infty, \quad \forall i \in L_-. \end{aligned} \tag{1}$$

This setup prevents labelled samples from being classified into the opposite class, as any cut associated with an infinite edge is not a min-cut [5].

Secondly, the Euclidean similarity between any of two samples is measured and have the pairwise distance matrix $\boldsymbol{D}$ in which $d_{i,j} = dist(\boldsymbol{x}_i, \boldsymbol{x}_j)$. Then, the sample nodes are connected by edges of weight 1 according to the one of the following connecting rules:

1) Mincut-N, each unlabelled sample is connected to its N nearest neighbors in terms of pairwise similarity shown in matrix $\boldsymbol{D}$. To avoid having isolated area in graph, one of the N-nearest neighbors is forced to be labelled. In other words, each unlabelled sample is connected to its nearest labelled neighbor and $N-1$ nearest unlabelled neighbors;
2) Mincut-δ, $\boldsymbol{x}_i$ and $\boldsymbol{x}_j$ are connected if their distance $d_{i,j}$ is less than a given threshold δ. Here, δ is determined through multiple attempts to meet one of the following conditions:
   (a) Mincut-δ_0, choose the maximum δ on which graph G has a 0 valued min-cut;
   (b) Mincut-$\delta_{1/2}$, find δ which forms the largest connected component by $1/2$ number of data points.

3.2 Solve Min-cut

Given G learned from a static dataset $\boldsymbol{X}$, we consider solving min-cut (i.e., to find a min-cut splitting G). According to [6], min-cut is equivalent to max-flow. Thus we address max-flow from now on.

Definition 3.1. *A flow on G is a real valued function $f()$ if the following conditions are satisfied:*

$$f(u,v) = -f(v,u), \quad \forall (u,v) \in V \times V; \tag{2a}$$

$$f(u,v) \leq C(u,v), \quad \forall (u,v) \in V \times V; \tag{2b}$$

$$\sum_u f(u,v) = 0, \quad \forall v \in V \setminus \{s,t\}. \tag{2c}$$

Let net flow $F = \sum_{u \in V} f(u,t)$ be the summation of flows into sink t. Then, the max-flow problem is to determine a flow from s to t with the maximum net flow F. In the rest of paper, we denote the direction from s to t as $s-t$.

In solving max-flow problem, existing algorithms fall in two categories, namely preflow push method and augmenting path method [6]. Online preflow push has been studied in [17]. In this paper, we address online max-flow learning through augmenting path.

Augmenting path algorithm stores information about current $s-t$ flow on G using residual graph R. The topology of R is identical to G (i.e., G and R have the same V and E). At the beginning of augmenting path, the residual graph R is initialized as G, i.e., let $R(e) = C(e) \forall e \in E$ where $R(e)$ is the residual capacity of edge e. Next, the algorithm involves an iterative procedure of the following two steps:

1) find $s-t$ path using Breadth-First Search (BFS). The resulting path P is a set of edges with positive residual capacity laid end to end connecting s to t, such as $P = \{(s,u),(u,v),(v,t) \mid R(s,u) > 0, R(u,v) > 0, R(v,t) > 0\}$.

2) augment the $s-t$ path found above. We firstly find the max amount of flow can go through path P, which is termed augmentation value and denoted as Δ_P in the rest of paper. Δ_P can be calculated as the minimum residual capacity of the whole path, i.e., $\Delta_P = min(R(u,v) \mid \forall (u,v) \in P)$. Next, we send Δ_P flow through path P in R as

$$\begin{aligned} R(u,v) &= R(u,v) - \Delta_P, \forall (u,v) \in P, \\ R(v,u) &= R(v,u) + \Delta_P, \forall (u,v) \in P. \end{aligned} \tag{3}$$

The above two steps are iteratively executed until no more $s-t$ path can be found. Algorithm 1 gives the pseudo code of batch augmenting path.

As the result of Algorithm 1, we obtain a residual graph R. From max-flow to min-cut, we simply perform BFS or DFS (Depth First Search) on R to find the set of nodes S reachable from s, and define $T = V \setminus S$. Then (S,T) is the $s-t$ min-cut.

Before further derivation, let's review some basic property of the residual graph R. Recall the augmentation procedure in Algorithm 1, if Δ_P flow is sent through any edge (u,v), we reduce $R(u,v)$ and increase $R(v,u)$ by Δ_P which follows that Δ_P capacity is taken from $R(u,v)$ (what left is the capacity available for later use) and expanded for $R(v,u)$ (available capacity can be used for sending flow through (u,v)). Thus, in any state of Algorithm 1, we have

$$R(u,v) + R(v,u) = C(u,v) + C(v,u), \quad \forall (u,v) \in E. \tag{4}$$

Algorithm 1 Augmenting Path Max-flow Batch Learning

Input: $G = (V,E,C)$, s and t.
Output: R and F.

```
Initialize R(e) = C(e), ∀e ∈ E, F = 0;
Find a s − t path P from the initial residual graph R;
while There is a s − t path P do
    Compute the amount of flow for augmentation: Δ_P = min(R(u,v) | ∀(u,v) ∈ P);
    Augment the path P, via updating the residual graph as R(u,v) = R(u,v) − Δ_P, ∀(u,v) ∈ P and R(v,u) = R(v,u) + Δ_P, ∀(u,v) ∈ P;
    Update the flow value as F = F + Δ_P;
    Find a s − t path P from the updated residual graph R.
end while
```

The flow that goes through any edge (u,v) can be traced by

$$f(u,v) = C(u,v) - R(u,v) = R(v,u) - C(v,u), \quad \forall (u,v) \in E. \tag{5}$$

Note that flow condition (2) holds for (5).

Now we can give the condition for R to be the residual graph of G.

Theorem 1. *Given a graph $G = (V,E,C)$ and a pseudo residual graph R, compute f through (5), if (4) and (2) are satisfied, R is the residual graph for G.*

Then we give the termination criteria for Algorithm 1 as

Theorem 2. *A flow F stored on R is a max-flow for G if and only if the residual graph R contains no $s-t$ path.*

The proof of above theorems can be found in [27] and [6].

4 Proposed Incremental Decremental Graph Max-flow for Online Semi-supervised Learning

Let $\mathcal{C}$, $\mathcal{A}$ and $\mathcal{R}$ be the set of sample index for current data, data to be added and removed respectively. Given newly acquired dataset $\boldsymbol{X}^{\mathcal{A}}$ and dataset $\boldsymbol{X}^{\mathcal{R}}$ to be retired from current data $\boldsymbol{X}^{\mathcal{C}}$. The goal of our work is to develop incremental decremental function $f()$ capable of updating min-cut on $\boldsymbol{X}^{\mathcal{C}}$ in response to data updates $\boldsymbol{X}^{\mathcal{A}}$ and $\boldsymbol{X}^{\mathcal{R}}$ as

$$M' = f(M, \boldsymbol{X}^{\mathcal{A}}, \boldsymbol{X}^{\mathcal{R}}) \tag{6}$$

where M is current min-cut on $\boldsymbol{X}^{\mathcal{C}}$ and M' is updated min-cut computed by incremental decremental learning on data updates $\boldsymbol{X}^{\mathcal{A}}$ and $\boldsymbol{X}^{\mathcal{R}}$. In principle, M' should be exactly the same as the batch min-cut on the updated dataset,

$$f(M, \boldsymbol{X}^{\mathcal{A}}, \boldsymbol{X}^{\mathcal{R}}) = g(\boldsymbol{X}^{\mathcal{C}} \setminus \boldsymbol{X}^{\mathcal{R}} \cup \boldsymbol{X}^{\mathcal{A}}) \tag{7}$$

where $g()$ is the batch learning system corresponding Algorithm 1.

As described above, a batch graph mincuts system consists of two main steps: 1. construct an undirected graph from the labelled and unlabelled samples based on their close neighbor relationship; and 2. conduct min-cut separation on above such graph through max-flow optimization. The objective of proposed online mincut, namely oMincut,

is to update the batch min-cut in response to any newly acquired samples and/or samples to be retired. Accordingly, proposed online system thus can be interpolated as updating a batch mincuts in two steps: online graph updating and incremental decremental max-flow.

4.1 Graph Updating

The objective of graph updating is to update current graph in response to data updates. Corresponding to the steps of graph construction for batch min-cut learning, we firstly update the pairwise distance matrix.

For removing a subset $\boldsymbol{X}^{\mathcal{R}}$ from current data $\boldsymbol{X}^{\mathcal{C}}$, we simply calculate the residual index set $\mathcal{C} \setminus \mathcal{R}$ and apply to $\boldsymbol{D}^{\mathcal{C}}$, then we have the updated pairwise distance matrix as

$$\boldsymbol{D}' = \boldsymbol{D}^{\mathcal{C}\setminus\mathcal{R}} \tag{8}$$

For adding data $\boldsymbol{X}^{\mathcal{A}}$ into current data $\boldsymbol{X}^{\mathcal{C}}$, we first calculate $\boldsymbol{X}^{\mathcal{A}}$ pairwise distance matrix as $\boldsymbol{D}^{\mathcal{A}}$ in which $d^{\mathcal{A}}_{i,j} = dist(\boldsymbol{x}^{\mathcal{A}}_i, \boldsymbol{x}^{\mathcal{A}}_j)$. Next we calculate the distance matrix in between $\boldsymbol{X}^{\mathcal{C}}$ and $\boldsymbol{X}^{\mathcal{A}}$ as $\boldsymbol{D}^{\mathcal{C}\mathcal{A}}$ in which $d^{\mathcal{C}\mathcal{A}}_{i,j} = dist(\boldsymbol{x}^{\mathcal{C}}_i, \boldsymbol{x}^{\mathcal{A}}_j)$. Then, we have the updated pairwise distance matrix computed as

$$\boldsymbol{D}' = \begin{bmatrix} \boldsymbol{D}^{\mathcal{C}} & \boldsymbol{D}^{\mathcal{C}\mathcal{A}} \\ \boldsymbol{D}^{\mathcal{C}\mathcal{A}T} & \boldsymbol{D}^{\mathcal{A}} \end{bmatrix}. \tag{9}$$

In practice, we address data adding and retiring in one batch. In other words, given data $\boldsymbol{X}^{\mathcal{C}}$, $\boldsymbol{D}^{\mathcal{C}}$, $\boldsymbol{X}^{\mathcal{R}}$, and $\boldsymbol{X}^{\mathcal{A}}$, the final updated pairwise distance matrix is calculated by a) apply (8) to remove $\boldsymbol{X}^{\mathcal{R}}$; b) let $\boldsymbol{D}^{\mathcal{C}} = \boldsymbol{D}'$ and $\boldsymbol{X}^{\mathcal{C}} = \boldsymbol{X}^{\mathcal{C}} \setminus \boldsymbol{X}^{\mathcal{R}}$; and c) calculate $\boldsymbol{D}'$ by (9) to add $\boldsymbol{X}^{\mathcal{A}}$.

Having the updated $\boldsymbol{D}'$, we construct the updated graph G' according to the connecting rules described in 3.1.

4.2 Online Max-Flow Setup

According to the batch min-cut stated in Section 3, a max-flow model is represented as a residual graph R. Thus the goal of incremental decremental max-flow is to update R in response to graph update due to the changes of data.

Given graph $G = (V, E, C)$ and its updated graph $G' = (V', E', C')$. We observe four types of graph change: edges deleted, nodes deleted, nodes added, and edges added. For edges deleted, an edge with positive capacity indicates a sequential operation on G: reduce first the capacity of the edge to zero, followed by the deletion of this edge. Similarly for edges added, an edge with positive capacity implies a two-step operation: 1) add edge with zero capacity and 2) increase edge capacity to $C'(e)$. In general, we summarize the following six categories of graph operation, by which G can be transformed to G'.

(a) a set of edges E_r have capacity C_r to be reduced (i.e., $C'(e) = C(e) - C_r(e), \quad \forall e \in E_r$);
(b) a set of edges E_d to be deleted (i.e., $e \in E \quad e \notin E', \quad \forall e \in E_d$);
(c) a set of nodes V_d to be deleted (i.e.,$v \in V \quad v \notin V', \quad \forall v \in V_d$);
(d) a set of nodes V_a to be added (i.e., $v \notin V \quad v \in V', \quad \forall v \in V_a$);
(e) a set of edges E_a to be added (i.e., $e \notin E \quad e \in E', \quad \forall e \in E_a$);
(f) a set of edges E_g with each edge e capacity to be increased by $C_g(e)$ (i.e., $C'(e) = C(e) + C_g(e), \quad \forall e \in E_g$).

As updating residual graph R in response to all changes of G' against G, we combine all above six categories operations into one task list. For each step of graph update, we pick up a set of tasks from the list, apply the tasks to G, and conduct the corresponding graph update on R meanwhile always keeping R to be the residual graph of G. Afterwards, we remove the processed tasks from current task list. This iteration continues until current task list is empty (i.e., G becomes G'). Consequently, we obtain the updated residual graph R'. We address decremental learning first instead of the other way around, because decremental learning deducts the scale of graph which reduces the complexity of incremental max-flow learning.

4.3 Decremental Max-Flow

In the case a), an edge $(u, v) \in E_r$ has capacity $C_r(u, v)$ to be reduced. Consider edge capacity has non-negativity constraint. $C_r(u, v)$ thus is required to be no greater than initial capacity $C(u, v)$.

If $R(u, v) \geq C_r(u, v))$ which means there is enough "unused" capacity for this reduce, then we simply reduce $R(u, v)$ by $C_r(u, v)$ and we have R' as

$$\begin{aligned} R'(u, v) &= R(u, v) - C_r(u, v) \\ R'(e) &= R(e) \mid \forall e \in E \setminus (u, v). \end{aligned} \tag{10}$$

Lemma 3. *If $R(u, v) \geq C_r(u, v)$, then R' in (10) is the residual graph of $G' = (V, E, C')$, where $C'(u, v) = C(u, v) - C_r(u, v)$ and $C'(e) = C(e) \mid \forall e \in E \setminus (u, v)$. In simple capacity reduce, the max-flow value remains $F' = F$.*

Proof. Consider $R'(u, v) = R(u, v) - C_r(u, v)$, $C'(u, v) = C(u, v) - C_r(u, v)$, $R'(e) = R(e) \mid \forall e \in E \setminus (u, v)$ and $C'(e) = C(e) \mid \forall e \in E \setminus (u, v)$. (4) holds for R' and $G' = (E, V, C')$. Also we have $f_{G'}(u, v) = C'(u, v) - R'(u, v) = C(u, v) - R(u, v) = f_G(u, v)$, so $f_{G'}$ satisfies (2). According to Theorem 1, R' is the residual graph of $G' = (V, E, C')$. □

If $R(u, v) < C_r(u, v)$ which follows that there is not enough residual capacity left to be reduced due to some capacity on (u, v) is occupied by current flows, then we need to release beforehand the occupied capacity.

Current flows can be either cycle flows or existing $s - t$ flows. If capacity on (u, v) is occupied by cycle flows, then we release the capacity by canceling the cycles, for which we firstly find cycles by searching $u - v$ path with positive residual capacity from the residual graph, then cancel the located cycles by sending the same amount of flows in a reverse direction along the cycles. Note that canceling a cycle flow does not change current $s - t$ flow, as a cycle flow has no overlap with current $s - t$ flow.

In the case that capacity on (u, v) is occupied by $s - t$ flow. Let Σ be the amount of capacity to be released, clearly $\Sigma = C_r(u, v) - R(u, v)$. To release Σ capacity on (u, v), we send Σ flows from sink t to source s through a number (i.e. could be more than one) of paths that go through (u, v). Note that a single $t - s$ path passing (u, v) may not be able to deliver required Σ flows.

As discussed before, augmentation is about sending flow from s to t. Contrarily, sending flow reversely from t to s means a reverse direction augmentation. We name the operation as de-augmentation. Similar to augmentation, de-augmentation is an iterative process, and each iteration consists of three steps: a) finding path P as a $t-v-u-s$ path (i.e., a $t-v$ plus a $u-s$ path); b) determining the amount of flow to be de-augmented as $\Omega = min(\{\Sigma, R(e) \mid \forall e \in P\})$; and c) sending Ω capacity through path P by updating residual graph.

The steps of one iteration cycle cancellation and de-augmentation are described in Algorithm 2.

Algorithm 2 Cycle Cancellation and De-augmentation

Input: $G = (V, E, C)$, R, F, s, t, (u, v) and Σ.
Output: R', F' and Ω.

1: Initialize $R' = R$;
2: **if** A $u - v$ path P_{u-v} can be found from R' **then**
3: Form a complete cycle path as $P = \{P_{u-v}, (v, u)\}$;
4: Compute the flow value in the cycle P as $\Omega = min(R(e) \mid \forall e \in P)$;
5: Cancel the cycle P, via updating the residual graph as $R'(u, v) = R'(u, v) - \Omega, \forall (u, v) \in P$ and $R'(v, u) = R'(v, u) + \Omega, \forall (u, v) \in P$;
6: $F' = F$;
7: **else if** A $t - v$ path P_{t-v} and a $u - s$ path P_{u-s} can be found from residual graph R' **then**
8: Form a complete path to de-augment as $P = \{P_{t-v}, (v, u), P_{u-s}\}$;
9: Compute the amount of flow to de-augment as $\Omega = \mathbf{min}(\mathbf{\Sigma}, \{\mathbf{R(e)} \mid \forall \mathbf{e} \in \mathbf{P}\})$;
10: De-augment the path P by Ω, via updating the residual graph as $R'(u, v) = R'(u, v) - \Omega, \forall (u, v) \in P$ and $R'(v, u) = R'(v, u) + \Omega, \forall (u, v) \in P$;
11: Compute $F' = F - \Omega$.
12: **end if**

Lemma 4. *Algorithm 2 output R' is the residual graph of $G = (V, E, C)$, and F' is the flow value on R'.*

Proof. Consider R' is initialized by residual R and only edges on path P are updated. Condition (4) and (2) hold for all remaining nodes and edges in R' except those on path P. All edges on path P lose Ω capacity and their revised edges receive Ω capacity. This means, the cycle flow is canceled and the $s - t$ flow remains unchanged for cycle flow. In de-augmentation case, a Ω flow is removed from the original $s-t$ flow, so that $F' = F - \Omega$. Also, condition (4) and (2) are satisfied for these nodes and edges on path P. By Theorem 1, R' is a residual graph of $G = (V, E, C)$. □

By Algorithm 2, we can only release Ω capacity on (u, v). To release $\Sigma (\Sigma \geq \Omega)$ capacity, we call Algorithm 2 iteratively until Σ capacity is released in total. Next we apply simple reduce to $R(u, v)$. The complete procedure is given in Algorithm 3. Note that Algorithm 2 is called here just once when the graph is an unit graph. This is because one de-augmentation releases all flows on (u, v).

By Lemma 3 and 4, we easily know that

Algorithm 3 Capacity Reduce through Cycle Cancellation and De-augmentation

Input: $G = (V, E, C)$, R, F, s, t, (u, v) and $C_r(u, v)$
Output: $G' = (V, E, C')$, R' and F'.

1: Initialize $R' = R$, $G' = G$, $F' = F$;
2: Compute the total amount to de-augment $\Sigma = C_r(u, v) - R(u, v)$;
3: **while** $\Sigma > 0$ **do**
4: Conduct de-augmentation through Algorithm 2 to update R', F' and calculate Ω;
5: Update Σ by $\Sigma = \Sigma - \Omega$;
6: **end while**
7: Apply simple capacity reduce on (u, v) as $R'(u, v) = R'(u, v) - C_r(u, v)$ and $C'(u, v) = C'(u, v) - C_r(u, v)$

Lemma 5. *Algorithm 3 output R' is the residual graph of $G' = (V, E, C')$, and F' is the flow value of R'. Here, $C'(u, v) = C(u, v) - C_r(u, v)$ and $C'(e) = C(e) \mid \forall e \in E \setminus (u, v)$.*

Remind that we have a set of edges E_r whose capacity needs to be reduced for max-flow update. Firstly, we perform simple reduce on those edges that are applicable and have $R(e) \geq C_r(e)$. For remaining edges, we conduct iteratively the following two operations until all edges in E_r are addressed: a) apply Algorithm 3 on an edge to release its capacity, and b) conduct simple reduce on those edges that newly become applicable due to operation a). Algorithm 4 presents the steps of capacity reduce on E_r.

Algorithm 4 Capacity Reduce on E_r

Input: $G = (V, E, C)$, R, F, s, t, E_r and C_r.
Output: $G' = (V, E, C')$, R' and F'.

1: Initialize $R' = R$, $C' = C$, $G' = (V, E, C')$, $F' = F$;
2: Find subset E_{sr} in E_r such that $R'(e) \geq C_r(e) \mid \forall e \in E_{sr}$;
3: Conduct simple capacity reduce on each edge in E_{sr} as $R'(e) = R(e) - C_r(e)$, $C'(e) = C(e) - C_r(e) \mid \forall e \in E_{sr}$;
4: Delete edges in E_{sr} from E_r as $E_r = E_r \setminus E_{sr}$;
5: **while** E_r is not empty **do**
6: Pick the first edge (u, v) from E_r;
7: Apply Algorithm 3 on (u, v) to obtain updated G', R' and F';
8: Delete edge (u, v) from E_r as $E_r = E_r \setminus (u, v)$;
9: Find subset E_{sr} in E_r such that $R'(e) \geq C_r(e) \mid \forall e \in E_{sr}$;
10: Conduct simple capacity reduce on each edge in E_{sr} as, $R'(e) = R(e) - C_r(e)$, $C'(e) = C(e) - C_r(e) \mid \forall e \in E_{sr}$;
11: Delete edges in E_{sr} from E_r as $E_r = E_r \setminus E_{sr}$
12: **end while**.

In Algorithm 4, simple reduce (i.e., line 9 and 10) is conducted when Algorithm 3 is called. This is because that edges originally not applicable for simple capacity reduce may become applicable (i.e., satisfy the condition of line 9) after Algorithm 3 is executed. For instance, we have a $s - u - t$ path that carries unit flow. Here, neither (s, u) nor (u, t) is applicable for simple reduce because their capacities are occupied by the flow. When we apply Algorithm 3 to release the residual capacity on (s, u), the algorithm releases

actually the residual capacity of the whole $s-u-t$ path including (s,u) and (u,t). Therefore the capacity release applied on (s,u) has changed the status of (u,t) to being applicable for simple reduce.

Further based on Lemma 3 and 5, we have

Lemma 6. *Algorithm 4 output R' is the residual graph of $G' = (V,E,C')$, F' is the flow value in R'. Here $C'(e) = C(e) - C_r(e) \mid \forall e \in E_r$ and $C'(e) = C(e) \mid \forall e \in E \setminus E_r$.*

Note that R' here may not carry the max-flow of G', because new $s-t$ paths might be formed by newly released edges and those edges that originally carry no flow. To compute max-flow on G', according to Theorem 2 we simply augment $s-t$ paths in R' until no more $s-t$ paths are found. For simplicity, we postpone this augmentation until graph G is expanded with new edges and nodes (i.e, the completion of graph update case (d), (e) and (f)), and carry out all augmentation works in one batch.

Now we address graph update case (b). We can safely delete all E_d edges in G' and R', as the capacity of these edges have been reduced to zero. For graph update case (c), we simply delete all V_d nodes in G' and R' simultaneously, as any edge associated with these nodes has already been deleted in case (b).

4.4 Incremental Max-flow

For graph update case (d), we simply add all V_a nodes into G' and R', respectively. Similarly for case (e), we just expand G' and R' with all edges in E_a. Note that R' is the residual graph of G' holds for case (b), (c), (d) and (e) since no capacity is changed in these updates and the topology of R' for all cases is kept the same as G'.

For graph update case (f), the capacity of each edge $(u,v) \in E_g$ is required to increase by $C_g(u,v)$. Thus we have,

$$\begin{aligned} R'(u,v) &= R(u,v) + C_g(u,v) \\ R'(e) &= R(e) \mid \forall e \in E \setminus (u,v). \end{aligned} \tag{11}$$

Lemma 7. *For (11) R' is the residual graph of $G' = (V,E,C')$, and the flow value keeps $F' = F$. Here $C'(u,v) = C(u,v) + C_g(u,v)$ and $C'(e) = C(e) \mid \forall e \in E \setminus (u,v)$.*

The proof of Lemma 7 is similar to that of Lemma 3, thus is omitted here.

Applying (11) to all edges in E_g, we have

$$\begin{aligned} R'(u,v) &= R(u,v) + C_g(u,v) \mid \forall (u,v) \in E_g \\ R'(e) &= R(e) \mid \forall e \in E \setminus E_g, \end{aligned} \tag{12}$$

where we have the following guarantee by Lemma 7:

Lemma 8. *R' in (12) graph is the residual graph of $G' = (V,E,C')$, and the flow value keeps $F' = F$. Here $C'(e) = C(e) + C_g(e) \mid \forall e \in E_g$ and $C'(e) = C(e) \mid \forall e \in E \setminus E_g$.*

As the result of graph update case (a) to (f), we have the updated G' and R'. By Lemma 6 and 8, R' is the residual graph of G'. In finding max-flow on G', we augment iteratively any $s-t$ path found on R' until no more $s-t$ path can be found. According to Theorem 2, the resulting R' from the augmentation carries the max-flow on G'. The complete procedure of online max-flow is given in Algorithm 5.

Algorithm 5 Incremental and Decremental Augmenting Path algorithm

Input: $G=(V,E,C)$, R, F, s, t, E_r, C_r, E_d, V_d, V_a, E_a, E_g, C_g.
Output: $G'=(V',E',C')$, R', F'.

```
Apply Algorithm 4 to conduct capacity reduce on E_r, and obtain G' = (V, E, C'), R' and F' ;
Delete edges in E_d from E as E' = E \ E_d
Delete nodes in V_d from V as V' = V \ V_d;
Add nodes in V_a into V' as V' = V' ∪ V_a;
Add edges in E_a into E' as E' = E' ∪ E_a;
Conduct simple capacity increase in E_g as (12);
Find a s − t path P in the residual graph R';
while There is a s − t path P do
    Compute the amount of flow to augment as Δ_P = min(R'(u,v) | ∀(u,v) ∈ P);
    Augment the path P, via updating the residual graph as R'(u,v) = R'(u,v) − Δ_P, ∀(u,v) ∈ P and R'(v,u) = R'(v,u) + Δ_P, ∀(u,v) ∈ P;
    Update the flow value as F' = F' + Δ_P;
    Find a s − t path P from the updated residual graph R'
end while.
```

4.5 Complexity Analysis

According to [6], batch augmenting path algorithm takes $O(|V||E|^2)$ time to find a max-flow from a graph. Our graph as defined in Section 3.1 is basically an unit graph with most edges capacity as 1. In this case, batch augmenting path takes $O(F|E|)$ time for computing a max-flow. Here, F is the number of augmentation.

Consider decremental learning of max-flow. Proposed algorithm involves an iterative de-augmentation plus a followed iterative augmentation. For each iteration of de-augmentation, BFS takes $O(|E_o|)$ time to find a $t-s$ path, where E_o is the set of edges occupied by current max-flow. The total number of de-augmentation is ΔF, which equals to the amount of flow lost in the de-augmentation step. In the worest case, the graph after de-augmentation requires additional ΔF augmentation to find the max-flow. Each augmentation takes $O(|E|)$ time. Thus the overall complexity for decremental learning is $O(\Delta F|E_o| + \Delta F|E|) = O(\Delta F|E|)$.

For incremental learning of max-flow, proposed algorithm performs an iterative graph augmentation in response to newly added edges. The graph after edge adding has $|E'|$ edges in which $|E_o|$ edges are occupied by existing max-flow thus are not involved in the $s-t$ path search. In this sense, each augmentation costs $O(|E'| - |E_o|)$. Due to new edges added, $\Delta F = F' - F$ augmentations are required to calculate emerged flows, and the total cost on updating max-flow is $O(\Delta F(|E'| - |E_o|))$. As compared to batch retraining whose complexity is $O(F'|E'|)$, proposed algorithm saves computational costs in reducing the number of augmentations and the scale of $s-t$ path search.

5 Experiments and Discussions

In this section, the effectiveness of our algorithms for semi-supervised learning is testified. We evaluate proposed oM-

incut algorithms (corresponding to batch Mincut defined in Section 3.1) on both artificial and real world datasets. Details regarding these datasets, and the performance has been obtained on them are presented in the following.

5.1 Graphical Demonstration

In this section, a graphical demonstration is given to show the online graph/model updating of proposed algorithm.

Given that a dataset consists of both labelled and unlabelled samples, and an initial min-cut model is trained upon it. When new samples being added and old samples being retired, our algorithm conducts incremental and decremental learning, which updates the initial min-cut model to a new state.

Figure 1a gives an initial labelled and unlabelled dateset, where cycled points in red, blue and green represent positive, negative and unlabelled samples, and S and T denote the virtual source and sink nodes, respectively. By a max-flow batch learning, we have Figure 1b as the obtained k-NN graph and the corresponding max-flow. In this figure, edges in red and blue refer to the infinite weighted edges connecting S to all positive labelled nodes and T to all negative labelled nodes respectively. The remaining edges are all 1 weighted nearest neighbor connections, in which edges in yellow carry no flow and black edges have the max-flow going through. As we can see from the figure, the max-flow value is 3, and the flow goes through: a) $S, 12, 14, 17, 22, T$, b) $S, 12, 15, 19, 23, 26, T$, and c) $S, 12, 16, 19, 26, T$. We then have the min-cut $\{(12, 14), (15, 19), (16, 19)\}$ which splits the whole graph into two isolated parts. Thus, all unlabelled nodes are naturally classified into positive and negative classes. To show the classification of unlabelled nodes, the numbers are colored in the figure to differentiate nodes, red as positive class and blue for negative, respectively.

Figure 1c-1j describes how the graph is being updated, when new samples are added and/or old samples are retired. Consider a set of nodes are required to be removed and added which are drawn in light blue and purple respectively in Figure 1c. We construct a k-NN graph on the updated dataset, and compare the graph with the one before update (i.e., Figure 1b). Consequently, a set of edge updates are shown in Figure 1d as the lines in light blue and purple, which correspond to those edges to be removed and added, respectively.

For updating the min-cut, we do removal first. For those edges carry no flow, we simply remove them from the graph, since the removal of these edges causes no change on current max-flow model. The resulting graph is shown in Figure 1f. For those edges with flow, such as $15 - 19$, $16 - 19$, $19 - 23$ and $19 - 26$, we deaugement path $S, 12, 15, 19, 23, 26, T$ and $S, 12, 16, 19, 26, T$ by Algorithm 2 and set edges free (i.e., with no flow) as in Figure 1g, then remove all together light blue edges and nodes. Figure 1h gives the obtained graph from removal. Next, we handle adding. We simply include those purple nodes and edges in Figure 1i, and expand them into the graph. To obtain the max-flow in this expanded graph, we iteratively augment any $s - t$ path found, and have the final max-flow model shown in Figure 1j.

5.2 Static Classification

In this experiment, we compare classic supervised SVM, semi-supervised SVM self-training and K Nearest Neighbor with proposed algorithms on a series of benchmark UCI datasets. Since proposed algorithms are applicable for two-class problems, several two-class datasets are selected, and the multiclass datasets in UCI are converted into two-class datasets by combining several classes into one. The name of the dataset used, the dimensionality of the dataset, and the number of instances from positive/negative class are summarized in the first column in Table 2.

For each dataset, we form our training datasets with both labelled and unlabelled data in which labelled instances are randomly selected from the original dataset, and unlabelled instances are adopted from those unused instances by hiding their label information. Here, we intend to explore the effect of label ratio on semi-supervised learning from 0.1 to 0.01 and further down to the level of 0.001. Thus, we set label ratio as 0.001, 0.002, 0.005, 0.01, 0.02, 0.05, 0.1 and 0.2, which correspond to the column 3 to 10 in Table 2, respectively. In our experiments, both labelled and unlabelled data are employed to train semi-supervised learners, and only labelled data are used to train supervised learners. Each learner is tested by its performance on predicting the labels of all unlabelled instances (i.e., testing dataset). For each label ratio, 25 rounds of independent tests are taken.

For performance evaluation, the mean accuracy and the standard deviation are caculated and shown in form of $mean \pm std$. Hypothesis test **(t-test)** is also performed, where p-value is computed for each algorithm with respect to the algorithm that gives the highest mean accuracy. In Table 2, we categorize all p-values into $0.05 > p > 0.01$, $0.01 > p > 0.001$ and $0.001 > p$, and present as *, ** and *** respectively.

As seen from the table, oMincut learners outperform others on 4 out of 6 datasets at the level of 0.1 label ratio, which indicates that proposed learners are competitive to benchmark learners if no less than 10% data are labelled. When the label ratio is 10 times smaller, proposed learners achieve the best performance for 5 out of 6 datasets. It is worth noting that at the lowest level of 0.001, proposed learners are giving consistently the highest prediction accuracy for all applicable datasets. This follows that given 0.1% of data labelled, proposed leaner is still giving the best performance. Further, the oMincut-3's superiority on class-imbalanced classification is shown in Table 3 in which recall is evaluated on **G-mean (** $\sqrt{PosRecall \times NegRecall}$**).**

5.3 Tracing Concept Drift

In this experiment, we show how proposed algorithm handles concept drift while incremental and decremental learning in Figure 2.

Figure 2a shows the initial learning stage. In a two-dimensional feature space, there are two opposite concepts (i.e., data distribution) identified in blue and red colors respectively. Each concept consists of two sub-concepts with their data distribution bounded by circles. In this figure, dots and stars represent labelled and unlabelled samples respectively; and the color of symbols shows the class label of labelled samples or predicted label of unlabelled samples.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TKDE.2016.2550042, IEEE Transactions on Knowledge and Data Engineering

(a) Initial Dataset

(b) Initial Graph and Max-flow Model

(c) Data Update

(d) Edges to be Updated

(e) Edges to be Removed

(f) Edges Carry Flow to be Removed

(g) Result of De-augmentation

(h) Edge and Node Removal Done

(i) Edge and Node Adding

(j) Augmenting Path to Find Max-flow

Fig. 1. Graphical Demonstration

Let two concepts start drifting by rotating all four sub-concepts against $(0, 0)$ for 5 stages. For each step rotation, we keep constant the scale of each sub-concept (i.e., the radius) and its distance to to $(0, 0)$, then turn all circles around $(0, 0)$ for 18 degrees. Consequently when all sub-concepts rotate 90 degrees, the two concepts exchange their positions. Figure 2b-2e show the procedure of concept drifting in which solid circles represent current and dashed ones represent the past sub-concepts. As the result of concept drifting, we form a dynamic data stream whose samples that are no longer in the scope of current concept are being removed; and new samples fall into current concept are

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TKDE.2016.2550042, IEEE Transactions on Knowledge and Data Engineering

TABLE 2
Accurcay Comparison on UCI Datasets

Dataset	Algorithm	Ratio of Label 0.001	0.002	0.005	0.01	0.02	0.05	0.1	0.2
BankNote	oMincut-3	–	74.49 ± 8.85 ***	80.72 ± 7.99 ***	88.85 ± 4.98 ***	**94.64** ± 2.25 –	**97.66** ± 1.27 –	**99.06** ± 0.58 –	**99.61** ± 0.38 –
	oMincut-δ_0	–	93.24 ± 0.34 **	91.74 ± 7.14 *	**92.05** ± 3.92 –	92.08 ± 4.75 *	94.90 ± 2.42 ***	94.67 ± 2.21 ***	96.69 ± 1.41 ***
4 Features	oMincut-$\delta_{1/2}$	–	**93.65** ± 0.77 –	**92.04** ± 7.26 –	92.05 ± 4.91 ***	92.08 ± 4.75 *	94.90 ± 2.42 ***	94.66 ± 2.18 ***	96.69 ± 1.41 ***
610 PosInst	SVM	–	73.28 ± 8.24 ***	76.84 ± 9.23 ***	81.76 ± 6.99 ***	89.11 ± 3.11 ***	95.89 ± 1.52 ***	97.51 ± 0.47 ***	97.81 ± 0.52 ***
762 NegInst	SVM-self	–	73.32 ± 9.18 ***	76.67 ± 10.00 ***	81.41 ± 7.89 ***	89.89 ± 3.95 ***	96.67 ± 1.61 *	97.88 ± 0.54 ***	97.99 ± 0.61 ***
	k-NN	–	67.46 ± 7.90 ***	73.40 ± 9.47 ***	84.77 ± 6.47 ***	91.58 ± 2.56 ***	96.62 ± 1.28 ***	98.58 ± 0.60 ***	99.33 ± 0.52 **
CNAE9	oMincut-3	–	–	–	**93.37** ± 6.03 –	**95.13** ± 2.38 –	**96.54** ± 1.48 –	**97.35** ± 1.44 –	**98.22** ± 0.62 –
	oMincut-δ_0	–	–	–	87.91 ± 7.07 *	90.05 ± 0.99 ***	90.68 ± 0.68 ***	90.90 ± 0.67 ***	91.55 ± 0.31 ***
856 Features	oMincut-$\delta_{1/2}$	–	–	–	88.03 ± 6.75 *	90.05 ± 0.99 ***	90.68 ± 0.68 ***	90.90 ± 0.67 ***	91.55 ± 0.31 ***
120 PosInst	SVM	–	–	–	89.01 ± 0.10 **	89.01 ± 0.11 ***	89.09 ± 0.18 ***	89.54 ± 0.41 ***	90.55 ± 0.32 ***
960 NegInst	SVM-self	–	–	–	89.00 ± 0.09 **	89.01 ± 0.11 ***	89.08 ± 0.18 ***	89.54 ± 0.43 ***	90.55 ± 0.33 ***
	k-NN	–	–	–	92.10 ± 2.61	93.49 ± 2.70 **	94.45 ± 1.71 ***	95.63 ± 1.41 ***	97.39 ± 0.74 ***
InternetAdvS	oMincut-3	–	**87.64** ± 1.66 –	**87.78** ± 1.72 –	**87.92** ± 1.74 –	**87.92** ± 2.92 –	**90.75** ± 1.91 –	**91.60** ± 2.38 –	**93.58** ± 1.27 –
	oMincut-δ_0	–	84.65 ± 6.27 *	85.54 ± 2.87 ***	86.44 ± 0.54 ***	86.12 ± 0.52 **	86.57 ± 0.49 ***	86.78 ± 0.40 ***	87.09 ± 0.27 ***
1558 Features	oMincut-$\delta_{1/2}$	–	84.39 ± 4.59 **	85.17 ± 2.86 ***	86.30 ± 0.46 ***	86.12 ± 0.52 **	86.57 ± 0.49 ***	86.78 ± 0.40 ***	87.09 ± 0.27 ***
459 PosInst	SVM	–	86.02 ± 0.03 ***	86.08 ± 0.07 ***	86.09 ± 0.07 ***	86.17 ± 0.08 **	86.37 ± 0.15 ***	86.72 ± 0.18 ***	87.22 ± 0.17 ***
2820 NegInst	SVM-self	–	86.02 ± 0.03 ***	86.08 ± 0.07 ***	86.09 ± 0.07 ***	86.17 ± 0.08 **	86.37 ± 0.15 ***	86.71 ± 0.18 ***	87.22 ± 0.17 ***
	k-NN	–	86.00 ± 0.00 ***	86.59 ± 0.73 **	86.79 ± 0.90 **	87.42 ± 0.99	89.40 ± 0.58 **	91.16 ± 0.66	93.35 ± 0.58
ionosphere	oMincut-3	–	–	–	66.06 ± 7.01 ***	71.95 ± 7.04 **	75.10 ± 9.79 **	79.25 ± 7.73 ***	82.91 ± 6.23 ***
	oMincut-δ_0	–	–	–	79.94 ± 12.92 **	80.10 ± 10.67 **	81.22 ± 10.48	74.15 ± 13.80 ***	69.06 ± 9.65 ***
34 Features	oMincut-$\delta_{1/2}$	–	–	–	**83.62** ± 13.38 –	**82.82** ± 11.61 –	81.04 ± 10.28	74.22 ± 13.86 ***	69.00 ± 9.51 ***
225 PosInst	SVM	–	–	–	66.34 ± 3.69 ***	69.90 ± 7.52 ***	83.49 ± 7.80	**89.64** ± 3.79 –	**92.30** ± 1.49 –
126 NegInst	SVM-self	–	–	–	66.38 ± 3.65 ***	69.90 ± 7.50 ***	**83.60** ± 7.77 –	89.55 ± 3.87	92.30 ± 1.52
	k-NN	–	–	–	65.31 ± 5.57 ***	68.29 ± 5.67 ***	72.05 ± 5.86 ***	76.97 ± 6.69 ***	81.91 ± 3.76 ***
Musk	oMincut-3	–	–	53.23 ± 5.53 ***	54.10 ± 5.74 ***	57.00 ± 4.00 ***	63.80 ± 4.48 *	**68.48** ± 4.07 –	**75.47** ± 2.91 –
	oMincut-δ_0	–	–	**60.68** ± 5.11 –	64.56 ± 4.64	63.66 ± 4.36	**66.38** ± 3.65 –	65.33 ± 5.30 *	65.36 ± 4.57 ***
166 Features	oMincut-$\delta_{1/2}$	–	–	59.83 ± 6.91	**64.74** ± 4.35 –	**63.93** ± 4.39 –	66.38 ± 3.65 ***	65.33 ± 5.30 *	65.36 ± 4.57 ***
207 PosInst	SVM	–	–	54.90 ± 6.09 **	58.00 ± 8.38 **	59.34 ± 1.59 ***	61.06 ± 1.16 ***	65.53 ± 1.69 **	71.95 ± 2.30 ***
269 NegInst	SVM-self	–	–	54.88 ± 6.05 **	57.91 ± 8.31 **	59.14 ± 1.56 ***	60.80 ± 1.12 ***	65.35 ± 1.75 **	71.71 ± 2.32 ***
	k-NN	–	–	49.90 ± 4.02 ***	49.91 ± 4.20 ***	55.44 ± 3.75 ***	59.44 ± 4.25 ***	65.50 ± 2.79 **	72.46 ± 2.00 ***
waveform	oMincut-3	**72.26** ± 4.01 –	**77.02** ± 4.58 –	**80.60** ± 2.09 –	81.22 ± 1.92 ***	82.65 ± 1.18 ***	83.63 ± 0.69 ***	84.33 ± 0.51 ***	84.24 ± 0.69 ***
	oMincut-δ_0	64.48 ± 4.79 ***	66.76 ± 1.10 ***	66.29 ± 1.32 ***	66.62 ± 0.71 ***	66.97 ± 0.34 ***	66.99 ± 0.17 ***	66.94 ± 0.09 ***	66.97 ± 0.10 ***
21 Features	oMincut-$\delta_{1/2}$	64.17 ± 4.73 ***	66.76 ± 1.10 ***	66.29 ± 1.32 ***	66.62 ± 0.71 ***	66.97 ± 0.34 ***	66.99 ± 0.17 ***	66.94 ± 0.09 ***	66.97 ± 0.10 ***
1657 PosInst	SVM	67.27 ± 1.37 ***	71.94 ± 5.09 ***	79.68 ± 3.82	84.37 ± 2.59	87.20 ± 1.17 *	88.31 ± 0.58	88.99 ± 0.30 *	89.45 ± 0.36 ***
3343 NegInst	SVM-self	67.19 ± 1.24 ***	71.70 ± 5.21 ***	79.62 ± 4.03	**84.41** ± 2.66 –	**87.28** ± 1.13 –	**88.36** ± 0.65 –	**89.05** ± 0.33 –	**89.58** ± 0.39 –
	k-NN	70.41 ± 5.13 *	74.66 ± 4.77 *	80.02 ± 2.24	81.44 ± 1.34 ***	82.10 ± 1.50 ***	83.51 ± 0.79 ***	84.42 ± 0.71 ***	84.84 ± 0.58 ***

TABLE 3
G-mean Comparison on Imbalanced Datasets

Dataset	Algorithm	Ratio of Label 0.002	0.005	0.01	0.02	0.05	0.1	0.2
CNAE9	Mincut-3	–	–	**67.62** ± 22.59 –	**72.97** ± 17.98 –	**82.79** ± 8.44 –	**87.47** ± 7.98 –	**91.90** ± 3.25 –
	Mincut-δ_0	–	–	24.98 ± 17.83 ***	27.31 ± 16.54 ***	39.04 ± 9.66 ***	41.67 ± 8.59 ***	48.87 ± 2.85 ***
856 Features	Mincut-δ_r	–	–	24.92 ± 17.76 ***	27.31 ± 16.54 ***	39.04 ± 9.66 ***	41.67 ± 8.59 ***	48.87 ± 2.85 ***
120 PosInst	SVM	–	–	4.66 ± 6.12 ***	5.32 ± 6.46 ***	11.09 ± 7.64 ***	22.35 ± 9.64 ***	38.50 ± 3.73 ***
960 NegInst	SVM-self	–	–	3.77 ± 5.88 ***	5.32 ± 6.46 ***	10.49 ± 7.79 ***	21.88 ± 10.54 ***	38.49 ± 3.86 ***
	k-NN	–	–	46.45 ± 26.87 ***	60.32 ± 23.02 **	69.45 ± 14.52 ***	78.00 ± 9.22 ***	88.01 ± 4.16 ***
InternetAdvS	Mincut-3	**36.36** ± 12.37 –	**37.63** ± 12.66 –	**39.13** ± 12.76 –	**49.77** ± 8.20 –	**65.41** ± 5.40 –	**72.41** ± 3.20 –	**78.48** ± 2.25 –
	Mincut-δ_0	13.25 ± 12.40 ***	14.46 ± 8.69 ***	16.04 ± 9.63 ***	14.57 ± 4.51 ***	20.82 ± 6.78 ***	24.43 ± 5.40 ***	28.96 ± 2.96 ***
1558 Features	Mincut-δ_r	24.74 ± 26.33	17.03 ± 15.93 ***	16.81 ± 12.53 ***	14.57 ± 4.51 ***	20.82 ± 6.78 ***	24.43 ± 5.40 ***	28.96 ± 2.96 ***
459 PosInst	SVM	1.85 ± 2.88 ***	4.93 ± 4.81 ***	6.79 ± 3.79 ***	10.07 ± 2.91 ***	16.08 ± 3.67 ***	22.86 ± 2.89 ***	29.94 ± 2.11 ***
2820 NegInst	SVM-self	1.90 ± 3.00 ***	5.01 ± 4.91 ***	6.84 ± 3.82 ***	9.91 ± 3.15 ***	16.09 ± 3.75 ***	22.79 ± 2.92 ***	29.99 ± 2.14 ***
	k-NN	0.00 ± 0.00 ***	16.27 ± 13.55 ***	18.17 ± 16.28 ***	29.70 ± 12.56 ***	49.98 ± 4.13 ***	61.94 ± 4.12 ***	73.42 ± 3.02 ***

being added in.

Through five stages concept drift, we trace the error rate of oMincut at each stage, and compare the final stage rate with that of batch learning. As seen from the parentheses below the plots, the transductive error rates of oMincut are smaller than 7% for all five stages, and the final stage rate is exactly the same as the rate from the batch learning.

5.4 Stream Learning

In this section, we compare proposed algorithms with SVM, SVM self-training and k-NN on a real-world data stream learning. The data used here is the KDD99 intrusion detection stream, which consists of 122 features and over 125k samples. Each sample corresponds to a TCP connection which is either a normal connection or an attack. In this experiment, two online learning scenarios are adopted :

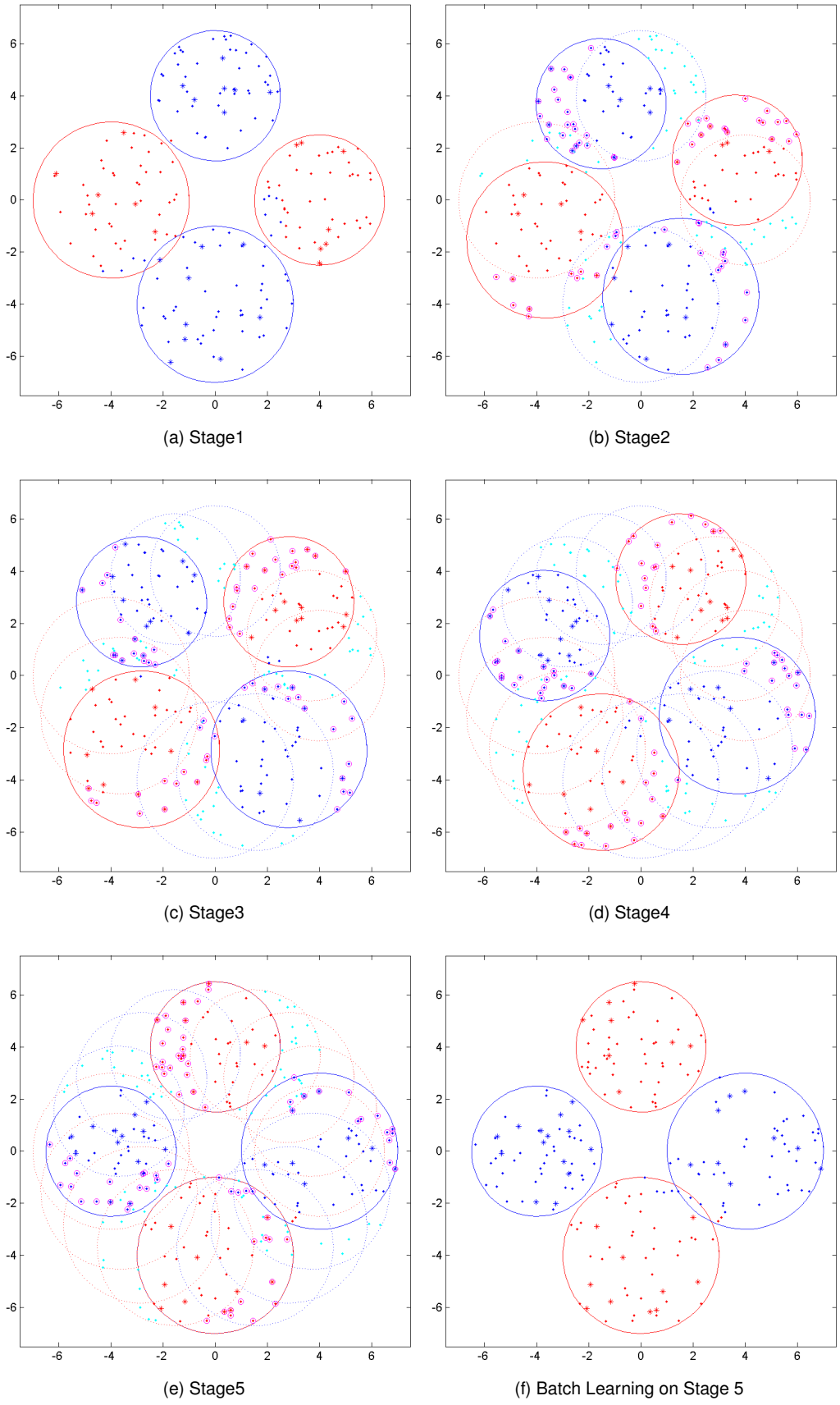
a) Sliding Window Snapshot: Consider the class concepts involve concept drifts. We let algorithms learn at each stage from a segment of data stream bounded by a sliding window. In this way, all learners retain at all time an up-to-date view of the drifting concepts. Figure 3a illustrates the setup of this stream learning, where the size of sliding window is 4000 and the length of sliding step is 2000.

b) Data Accumulation: Assume the class concepts are constant for the entire data stream. We let algorithms conduct incremental learning at each stage to reinforce learning effectiveness by accommodating new chunk of data. Here, the training data is fed to learner as in Figure 3b where the chunk size is set as 4000.

For each scenario, we conduct experiments with the same label ratios as the ones used in Section 5.2; and we use the testing accuracy as performance measurement.

For the sliding window snapshot scenario, Figure 4 compares all learners performance under the label ratio of 0.001, 0.01 and 0.1, respectively. As seen from the figure, the superiority of proposed oMincut-3 is observed for all three label ratios. When the ratio is 0.1, oMincut-3 has a visible superiority for 16 out of total 40 stages. Such superiority becomes evident in the case of 0.01 ratio, and is further enlarged when the ratio is 10 times lower. Apparently, all learners perform sensitive to the label ratio. Figure 5 shows the performance (in terms of accuracy mean and standard deviation) variation of learners against the label ratio. As we can see, all learners lose classification capability performing with lower accuracy and higher variance (indicating reduced system stability) when the label ratio decreases, however the proposed oMincut-3 gives the slowest perfor-

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TKDE.2016.2550042, IEEE Transactions on Knowledge and Data Engineering

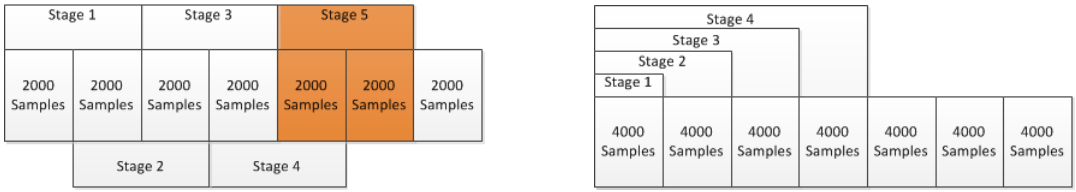


(a) Stage1 (b) Stage2

(c) Stage3 (d) Stage4

(e) Stage5 (f) Batch Learning on Stage 5

Fig. 2. oMincut incremental and decremental learning on five stages concept drift, with the final stage compared to batch learning. The transductive performance in terms of classification error rate is given at each stage in parentheses as (a) (6.43 percent), (b) (3.59 percent), (c) (4.46 percent), (d) (3.20 percent), (e) (4.40 percent) and (f) (4.40 percent).



(a) Sliding Window Snapshot (b) Data Accumulation

Fig. 3. Two stream learning scenarios

mance reduction on both accuracy and stability.

In the mode of data accumulation learning, learners are trained incrementally from an accumulating dataset. We measure learner performance by stage average accuracy and final stage accuracy, respectively. Figure 6 compares the performance of learners at different label ratios. Similar to the above snapshot learning, proposed oMincut-3 gives the lowest performance degeneration as the label ratio decreases.

6 Conclusions and Future Work

For learning from data streams with both labelled and unlabelled samples, an incremental decremental max-flow based online semi-supervised learning system is proposed. The basic assumption for this work is that samples with smaller distance are more likely to be in the same class. To conduct semi-supervised learning, a k-NN based sample network is built based on both labelled and unlabelled data to describe the "close to" relationship between samples. Then a min-cut is applied to split the whole set into two classes by removing the minimal number of "close to" relation. To find such min-cut, the max-flow problem is required to be solved on the sample network.

In this paper, we derive online max-flow for semi-supervised learning by proposing 1) an online sample network whose pair-wised sample distance matrix is being updated for every sample adding/retiring, and more importantly 2) a novel incremental/decremental max-flow algorithm to update the max-flow whenever the sample network changes. Proposed online semi-supervised learning system is proofed capable of accommodating new samples adding and old samples retiring, with a theoretical guarantee that online learning result equals that of batch retraining. Experiments on UCI benchmark datasets and KDD data stream show that the proposed system is less sensitive to the amount of labelled data (in terms of the ratio to the whole training data) as compared to k-NN, SVM and SVM self-training.

The characteristic of contemporary real-world application implies two valuable future directions of this work. 1) the scale of data stream increases fast, which requires high speed learner. Proposed algorithm works as a serial computation, but some time consuming modules can be parallelized to cope with high speed data streams learning, such as network construction/updating and network path searching for online max-flow. 2) proposed algorithm is able to retire unwanted samples, but in real-world application it is rare to have the priori knowledge of which set of data is no longer needed [19] [20]. Thus, incorporating concept drift detection mechanism is another interesting future topic.

References

[1] P. Kumar Mallapragada, R. Jin, A. Jain, and Y. Liu, "Semiboost: Boosting for semi-supervised learning," *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, vol. 31, no. 11, pp. 2000–2014, Nov 2009.

[2] O. Chapelle, B. Schlkopf, and A. Zien, *Semi-Supervised Learning*. The MIT Press, 2006.

[3] I. Cohen, "Semisupervised learning of classifiers with application to human-computer interaction," Ph.D. dissertation, Champaign, IL, USA, 2003, aAI3101819.

[4] J. Ortigosa-Hernández, I. Inza, and J. A. Lozano, "On the optimal usage of labelled examples in semi-supervised multi-class classification problems," UPV/EHU, Tech. Rep., April 2015. [Online]. Available: https://addi.ehu.es/handle/10810/15004

[5] A. Blum and S. Chawla, "Learning from labeled and unlabeled data using graph mincuts," in *Proceedings of the Eighteenth International Conference on Machine Learning*, ser. ICML '01. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 2001, pp. 19–26.

[6] J. Kleinberg and E. Tardos, *Algorithm Design*. Boston, MA, USA: Addison-Wesley Longman Publishing Co., Inc., 2005.

[7] D. Shengwu and Z. Yi, "Research on method of traffic network bottleneck identification based on max-flow min-cut theorem," in *Transportation, Mechanical, and Electrical Engineering (TMEE), 2011 International Conference on*, Dec 2011, pp. 1905–1908.

[8] O. Kosut, "Max-flow min-cut for power system security index computation," in *Sensor Array and Multichannel Signal Processing Workshop (SAM), 2014 IEEE 8th*, June 2014, pp. 61–64.

[9] S. Dandapat, B. Mitra, N. Ganguly, and R. Choudhury, "Fair bandwidth allocation in wireless mobile environment using max-flow," in *High Performance Computing (HiPC), 2010 International Conference on*, Dec 2010, pp. 1–10.

[10] S. Dandapat, B. Mitra, R. Choudhury, and N. Ganguly, "Smart association control in wireless mobile environment using max-flow," *Network and Service Management, IEEE Transactions on*, vol. 9, no. 1, pp. 73–86, March 2012.

[11] Y. Duan, W. Huang, and H. Chang, "Shape prior regularized continuous max-flow approach to image segmentation," in *Pattern Recognition (ICPR), 2012 21st International Conference on*, Nov 2012, pp. 2516–2519.

[12] T. Weglinski and A. Fabijanska, "Min-cut/max-flow segmentation of hydrocephalus in children from ct datasets," in *Signals and Electronic Systems (ICSES), 2012 International Conference on*, Sept 2012, pp. 1–6.

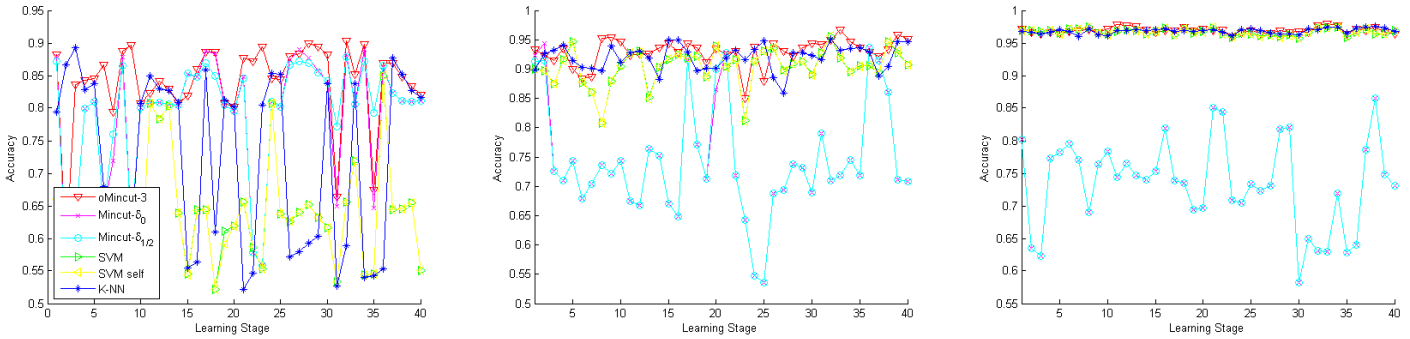
[13] Y. Boykov and M.-P. Jolly, "Interactive graph cuts for optimal boundary amp; region segmentation of objects in n-d images," in *Computer Vision, 2001. ICCV 2001. Proceedings. Eighth IEEE International Conference on*, vol. 1, 2001, pp. 105–112 vol.1.

[14] R. Shade and P. Newman, "Choosing where to go: Complete 3d exploration with stereo," in *Robotics and Automation (ICRA), 2011 IEEE International Conference on*, May 2011, pp. 2806–2811.

[15] H. Isack and Y. Boykov, "Energy based multi-model fitting amp;amp; matching for 3d reconstruction," in *Computer Vision and Pattern Recognition (CVPR), 2014 IEEE Conference on*, June 2014, pp. 1146–1153.

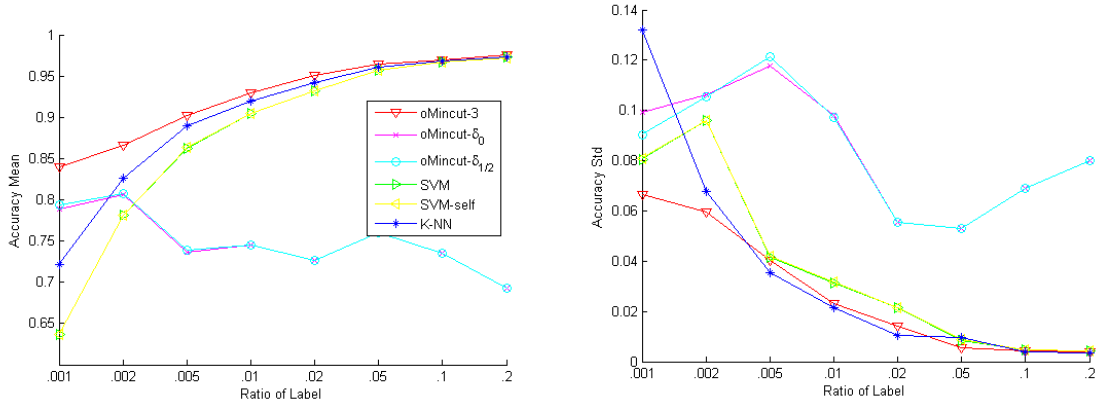
[16] D. Snow, P. Viola, and R. Zabih, "Exact voxel occupancy with graph cuts," in *Computer Vision and Pattern Recognition, 2000. Proceedings. IEEE Conference on*, vol. 1, 2000, pp. 345–352 vol.1.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TKDE.2016.2550042, IEEE Transactions on Knowledge and Data Engineering



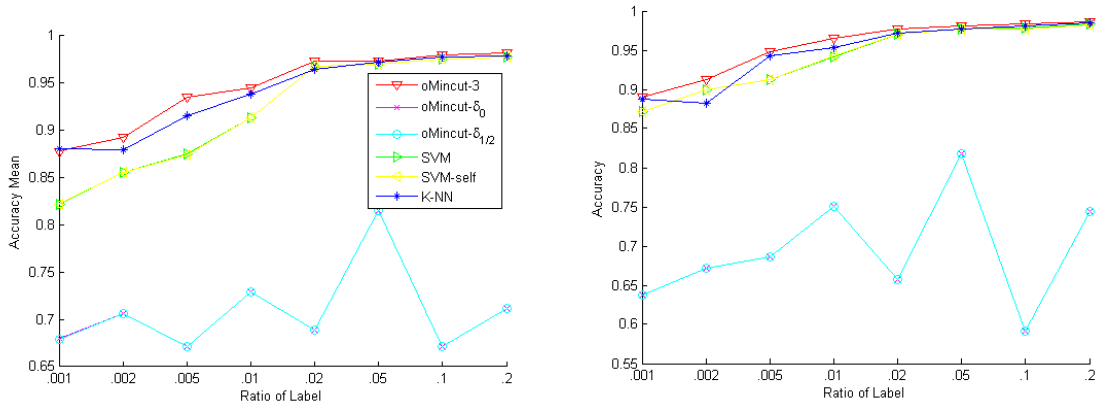
(a) R=.001 (b) R=.01 (c) R=.1

Fig. 4. Sliding Window Snapshot Learning on KDD Streams



(a) Accuracy Mean (b) Accuracy Standard Deviation

Fig. 5. The variation of learners performance against the label ratio.



(a) Stage Mean Accuracy (b) Final Stage Accuracy

Fig. 6. The variation of learners performance against the label ratio.

[17] S. Kumar and P. Gupta, "An incremental algorithm for the maximum flow problem," *Journal of Mathematical Modelling and Algorithms*, vol. 2, no. 1, pp. 1–16, 2003.

[18] P. Zhang, X. Zhu, and L. Guo, "Mining data streams with labeled and unlabeled training examples," in *Data Mining, 2009. ICDM '09. Ninth IEEE International Conference on*, 2009, pp. 627–636.

[19] P. Zhang, X. Zhu, J. Tan, and L. Guo, "Classifier and cluster ensembles for mining concept drifting data streams," in *Data Mining (ICDM), 2010 IEEE 10th International Conference on*, Dec 2010, pp. 1175–1180.

[20] G. Ditzler and R. Polikar, "Semi-supervised learning in non-stationary environments," in *Neural Networks (IJCNN), The 2011 International Joint Conference on*, July 2011, pp. 2741–2748.

[21] G. Carneiro and J. Nascimento, "Incremental on-line semi-supervised learning for segmenting the left ventricle of the heart from ultrasound data," in *Computer Vision (ICCV), 2011 IEEE International Conference on*, Nov 2011, pp. 1700–1707.

[22] B. Yver, "Online semi-supervised learning: Application to dynamic learning from radar data," in *Radar Conference - Surveillance for a Safer World, 2009. RADAR. International*, Oct 2009, pp. 1–6.

[23] S. Chu, S. Narayanan, and C.-C. Kuo, "A semi-supervised learning approach to online audio background detection," in *Acoustics, Speech and Signal Processing, 2009. ICASSP 2009. IEEE International Conference on*, April 2009, pp. 1629–1632.
[24] A. Bamdadian, C. Guan, K. K. Ang, and J. Xu, "Online semi-supervised learning with kl distance weighting for motor imagery-based bci," in *Engineering in Medicine and Biology Society (EMBC), 2012 Annual International Conference of the IEEE*, Aug 2012, pp. 2732–2735.
[25] K. K. Ang, Z. Y. Chin, H. Zhang, and C. Guan, "Filter bank common spatial pattern (fbcsp) algorithm using online adaptive and semi-supervised learning," in *Neural Networks (IJCNN), The 2011 International Joint Conference on*, July 2011, pp. 392–396.
[26] S. Imangaliyev, B. Keijser, W. Crielaard, and E. Tsivtsivadze, "Online semi-supervised learning: Algorithm and application in metagenomics," in *Bioinformatics and Biomedicine (BIBM), 2013 IEEE International Conference on*, Dec 2013, pp. 521–525.
[27] R. K. Ahuja, T. L. Magnanti, and J. B. Orlin, *Network Flows: Theory, Algorithms, and Applications*. Englewood Cliffs, NJ: Prentice Hall, 1993.

PLACE PHOTO HERE

Lei Zhu received the M.Sc. degree from Unitec Institute of Technology, New Zealand, in 2012.

He is now a Doctoral student with Unitec Institute of Technology. His research interests include machine learning, network security and mathematical modeling.

PLACE PHOTO HERE

Shaoning Pang received the B.Sc. degree in physics and the M.Sc. degree in electrical engineering from Xinjiang University, China, in 1994 and 1997, respectively, and the Ph.D. degree in computer science from Shanghai Jiao Tong University, Shanghai, China, in 2000.

From 2001 to 2003, he was a Research Associate with the Pohang University of Science and Technology (POSTECH), Pohang, South Korea. Currently, He is a Professor of Data Analytics, and the Director of the Decentralized Machine Learning Intelligence (DMLI) Laboratory (http://www.dmli.info), at the Department of Computing, Unitec Institute of Technology, New Zealand. His research interests include data analytics, cloud data and service resiliency, and computational intelligence for industry applications.

PLACE PHOTO HERE

Abdolhossein Sarrafzadeh is a professor and head of the Department of Computing at Unitec Institute of Technology, New Zealand. He is internationally known for his pioneering work on affective systems. He is the creator of Eve, world's first affect sensitive lifelike agent. He has made significant research contributions in computing resulting in more than 100 research articles. His research has gained international publicity with numerous TV and radio interviews and international media articles including the Le Monde. He has served on the editorial board of journals and program and organizing committees of numerous international conferences. He is a popular invited speaker. He is a member of the IEEE.

PLACE PHOTO HERE

Tao Ban received the B.S. degree from Xian Jiaotong University, Xian, China, in 1999, the M.S. degree in engineering from Tsinghua University, Beijing, China, in 2003, and the Ph.D. degree in information science from Kobe University, Kobe, Japan, in 2006.

He is currently an Senior Researcher with the Network Security Research Institute, National Institute of Information and Communications Technology, Tokyo, Japan. His research interests include: machine learning, support vector machine, and their applications to Cybersecurity.

PLACE PHOTO HERE

Daisuke Inoue received his B.E. and M.E. degrees in electrical and computer engineering and his Ph.D. degree in engineering from Yokohama National University in 1998, 2000 and 2003, respectively. He joined the Communications Research Laboratory (CRL), Japan, in 2003. CRL was relaunched as the National Institute of Information and Communications Technology (NICT) in 2004, where he is currently the director of the Cybersecurity Laboratory in the Network Security Research Institute. Since 2006, he has been conducting the NICTER project which consists of large-scale darknet (unused IP addresses) monitoring, fully automated malware analyses, and their correlations. The project is producing cutting-edge technologies such as a darknet-based alert system DAEDALUS and a real-time traffic visualizer NIRVANA, and so on. He has received several awards including the Best Paper award at the 2002 Symposium on Cryptography and Information Security (SCIS 2002), the Commendation for Science and Technology from the Minister of MEXT, Japan, in 2009, the Good Design Award 2013, and the Asia-Pacific Information Security Leadership Achievements award in 2014.