Classification and Novel Class Detection in Concept-Drifting Data Streams under Time Constraints

Mohammad M. Masud, Jing Gao, Latifur Khan, Jiawei Han, and Bhavani Thuraisingham

Abstract

Most existing data stream classification techniques ignore one important aspect of stream data: arrival of a novel class. We address this issue, and propose a data stream classification technique that integrates a novel class detection mechanism into traditional classifiers, which enables it to detect a novel class automatically before the true labels of the novel class instances arrive. Novel class detection problem becomes more challenging in the presence of concept-drift, when the underlying data distributions evolve in streams. In order to determine whether an instance belongs to a novel class, the classification model sometimes need to wait for more test instances to discover similarities among those instances. Therefore, unlike traditional classifiers that can classify a test instance immediately, a time delay is involved in the novel class detection mechanism. The maximum allowable time delay T_c, is imposed as a time constraint to classify a test instance. Furthermore, most existing stream classification approaches make impractical assumption about data labeling. They assume that the true label of a data point can be accessed immediately after the data point is classified using the classification model. In reality, true label of a data point is likely to be available after quite some time, since manual labeling is time consuming. Therefore, a delay is involved in labeling. We assume that label of a data point becomes available T_l time units after its arrival. We show how to take fast and correct classification decisions under these constraints, and apply them to real benchmark data. Comparison with state-of-the-art stream classification techniques prove the superiority of our approach.

Index Terms

Data streams, novel class, Ensemble classification

I. Introduction

Data stream classification poses many challenges, some of which have not been addressed yet. Most existing data stream classification algorithms [4], [8], [12], [17], [22], [24] address two major problems related to data streams: their "infinite length", and "concept-drift". Since data streams have infinite length, traditional multi-pass learning algorithms are not applicable as they would require infinite storage and training time. Concept-drift occurs in the stream when the underlying concept of the data changes over time. Thus, the classification model must be updated continuously so that it reflects the most recent concept. However, another major problem is ignored by most state-of-the-art data stream classification techniques, which is "concept-evolution", meaning, emergence of a novel class. Most of the existing solutions assume that the total number of classes in the data stream is fixed. But in real world data stream classification problems, such as intrusion detection, text classification and fault detection, novel classes may appear at any time in the stream (e.g. a new intrusion). Traditional data stream classification techniques would be unable to detect the novel class until the classification models are trained with labeled instances of the novel class. Thus, all novel class instances will go undetected (i.e., misclassified) until the novel class is manually detected by experts, and a training data with the instances of that class is made available to the learning algorithm. We address this concept-evolution problem and provide a solution that handles all three problems, namely, infinite length, concept-drift, and concept-evolution. Novel class detection should be an integral part of any realistic data stream classification technique because of the evolving nature of streams. It can be useful in various domains, such as network intrusion detection [9], fault detection [6], and credit card fraud detection [22]. For example, in case of intrusion detection, a new kind of intrusion might go undetected by traditional classifier, but our approach should not only be able to detect the intrusion, but also deduce that it is a new kind of intrusion. This discovery would lead to an intense analysis of the intrusion more by human experts in order to understand its cause, find a remedy, and make the system more secure.

We address the infinite length problem by dividing the stream into equal-sized chunks, so that each chunk can be accommodated in memory and processed online. Each chunk is used to train one classification model as soon as all the instances in the chunk is labeled. We handle concept-drift by maintaining an ensemble of M such classifiers. An unlabeled instance is classified by taking majority vote among the classifiers in the ensemble. The ensemble is continuously updated so that it represents the most recent concept in the stream. The update is performed as follows: as soon as a new model is trained, one of the existing models in the ensemble is replaced by it, if necessary. The victim is chosen by evaluating the error of each of the existing models in the ensemble on the latest labeled chunk, and discarding the one with the highest error. Our approach provides a solution to concept-evolution problem by enriching each classifier in the ensemble with a novel class detector. If majority of the classifiers discovers a novel class, the instances of that class are separated and treated accordingly. Thus, novel class can be automatically identified without manual intervention.

Our novel classes detection technique is different from traditional “one-classs” novelty detection techniques [11], [16], [23] that can only distinguish between the normal and anomalous data. That is, the traditional novelty detection techniques assume that there is only one “normal” class and any instance that does not belong to the normal class is an anomaly/novel class instance. Therefore, they are unable to distinguish among different types of anomaly. But our approach offers a “multi-class” framework for the novelty detection problem, that can distinguish between different classes of data and discover the emergence of a completely novel class. Besides, traditional novelty detection techniques simply identify data points as outliers/anomalies that deviate from the “normal” class. But our approach not only detects whether a single data point deviates from the existing classes, but also discovers whether a group of outliers possess the potential of forming a new class by showing strong cohesion among themselves. Therefore, our approach is a synergy of a “multi-class” classification model and a novel class detection model.

Traditional stream classification techniques also make impractical assumptions about the availability of labeled data. Most techniques [4], [8], [24] assume that the label of a data point can be accessed as soon as it has been classified by the classification model. Thus, according to their assumption, the existing model can be updated immediately using the labeled instance. In reality, we would not be so lucky in obtaining the label of a data instance immediately, since manual labeling of data is time consuming and costly. For example, in a credit card fraud detection problem, the actual labels (i.e., authentic/fraud) of credit card transactions are revealed only after the customer reports fraud transactions to the credit card company. Thus, a more realistic assumption would be to have a data point labeled after T_l time units of its arrival. For simplicity, we assume that the i-th instance in the stream arrives at i-th time unit. Thus, T_l can be considered as a time constraint imposed on data labeling process. Note that traditional stream classification techniques assume T_l = 0. Finally, we impose another time constraint, T_c, on classification decision. That is, an instance must be classified by the classification model within T_c time units of its arrival. If it assumed that there is no concept-evolution, it is customary to have T_c=0, i.e., an instance should be classified as soon as it arrives. However, when new concepts evolve, classification decision may have to be postponed until enough instances are seen by the model to gain confidence in deciding whether an instance belongs to a novel class or not. Note that $T_c < T_l$ must be maintained in any practical classification model. Otherwise, we would not need the classifier at all, we could just wait for the labels to arrive. We will discuss this issue in details in later sections.

Figure 1 illustrates the significance of T_l and T_c with an example. Here x_k is the last instance that has arrived in the stream. Let x_j be the instance that arrived T_c time unit earlier, and x_i

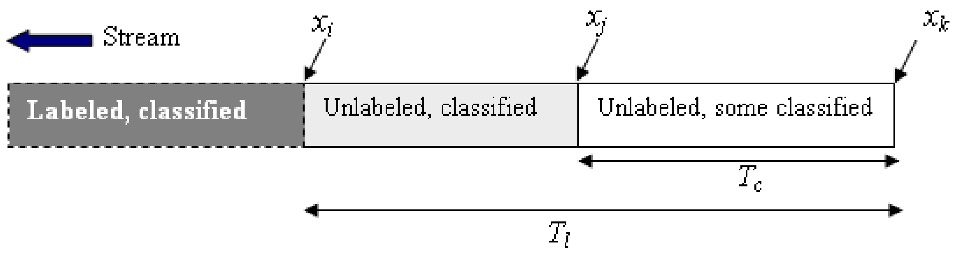


Fig. 1. Illustration of T_l and T_c

be the instance that arrived T_l time unit earlier. Then x_i and all instances that arrived before x_i (shown with dark-shaded area) are labeled, since all of them are at least T_l time units old. Similarly, x_j and all instances that arrived before x_j (both the light-shaded and dark-shaded areas) are classified by the classifier since they are at least T_c time units old. However, the instances inside the light-shaded area are unlabeled. Instances that arrived after x_j (age less than T_c) are unlabeled, and may or may not be classified (shown with the unshaded area).

Integrating classification with novel class detection is a nontrivial task, especially in the presence of concept-drift, and under time constraints. We assume an important property of each class: the data points belonging to the same class should be closer to each other (cohesion) and should be far apart from the data points belonging to other classes (separation). If a test instance is *well-separated* from the training data (called "Raw outlier"), it has potential to be a novel class instance. Raw outliers that possibly appear as a result of concept-drift or noise are filtered out. A filtered outlier (or Foutlier) has potential to be a novel class instance. However, we must *wait* to see whether more such Foutliers appear in the stream, which show strong cohesion among themselves. If a sufficient number of such strongly cohesive Foutliers are observed, a novel class is assumed to have appeared, and the Foutliers are classified as a novel class instance. However, we can wait at most T_c time unit to output the classification decision of a test instance after its arrival, which makes the problem more challenging. Furthermore, we must keep detecting novel class instances in this 'unsupervised' fashion for at least T_l time units from the arrival of the first novel class instance, since labeled training data of the novel class(es) would not be available before that.

We have several contributions. First, to the best of our knowledge, no other stream classification techniques address the concept-evolution problem. This is a major problem with data stream that must be dealt with. In this light, this paper offers a more realistic solution to data stream classification. Second, we propose a more practical framework for stream classification by introducing time constraints for delayed data labeling and making classification decision. Third, our proposed technique enriches traditional classification model with a novel class detection mechanism. Finally, we apply our technique on both synthetic and real-world data and obtain much better results than state-of the art stream classification algorithms.

The rest of the paper is organized as follows: section II discusses related work, section III discusses the overview of our approach, section IV discusses our approach in detail, section V discusses the datasets and experimental evaluation, and finally, section VI concludes with direction to future works.

II. Related work

We discuss the works related to both data stream classification and novelty detection.

Data stream classification has been an interesting research topic for years. These approaches fall into one of two categories: single model and ensemble classification. Single model classification techniques maintain and incrementally update a single classification model [4], [8], [24]. These techniques also effectively respond to concept-drift. Several ensemble techniques for stream data mining have been proposed [7], [17], [22]. Ensemble techniques require relatively simpler operations to update the current concept than their single model counterparts, and also handle concept-drift efficiently. Our approach follows the ensemble technique. However, our approach is different from all other stream classification techniques in two different aspects. First, none of the existing techniques can detect novel classes, but our technique can. Second, our approach is based on a more practical assumption about the time delay in data labeling, which is not considered in most of the existing algorithms.

Our technique is also related to novelty/anomaly detection. Markou and Singh study novelty detection in details in [11]. Most novelty detection techniques fall into one of two categories: parametric, and non-parametric. Parametric approaches assume a distribution of data, and estimate parameters of the distribution from the normal data. According to this assumption, any test instance is assumed to be novel if it does not follow the distribution [14], [16]. Our technique is a non-parametric approach, therefore, it is not restricted to any specific data distribution. There are several non-parametric approaches available, such as parzen window method [23] and K-nearest neighbor (K-NN) based approaches [25].

Our approach is different from the above novelty/anomaly detection techniques in three aspects. First, existing novelty detection techniques only consider whether a test point is significantly different from the normal data. However, we not only consider whether a test instance is sufficiently different from the training data, but also consider whether there are strong similarities among the test instances. Therefore, existing techniques discover novelty in a single test point, whereas but our technique discovers novelty among a collection of test points to detect the presence of a novel class. Second, our model can be considered as a "multi-class" novelty detection technique, since it can distinguish among different classes of data, and also discover emergence of a novel class. But other novelty detection can only distinguish between normal and novel, and, therefore, can be considered as "one-class" classifiers. Finally, most of the existing novelty detection techniques assume that the "normall" model is static, i.e., there is no concept-drift in the data. But our approach can detect novel classes in the presence of concept-drift.

Novelty detection is also closely related to outlier detection techniques. There are many outlier detection techniques available, such as [1]–[3], [10]. Some of them are also applicable to data streams [19], [20]. However, the main difference with these outlier detection techniques from ours is that our primary objective is novel class detection, not outlier detection. Outliers are the by-product of intermediate computation steps in our algorithm. Thus, the precision of our outlier detection technique is not too critical to the overall performance of our algorithm.

Spinosa et al. [18] propose a cluster based novel concept detection technique that is applicable to data streams. However, this is also a "single-class" novelty detection technique, where authors assume that there is only one 'normal' class and all other classes are novel. Thus, it not directly applicable to a multi-class environment, where more than one classes are considered as 'normal' or 'non-novel'. But our approach can handle any number of existing classes, and also detect a novel class that do not belong to any of the existing classes. Therefore, our approach offers a more practical solution to the novel class detection problem, which has been proved empirically.

This paper is an extension to our previous work [13] in which we proposed a novel class detection technique. However, in our previous work, we did not consider the time constraints T_l and T_c. Therefore the current version is more practical than the previous one. These time constraints impose several restrictions on the classification algorithm, making classification more challenging. We encounter these challenges and provide efficient solutions.

III. Overview

At first, we mathematically formulate the data stream classification problem.

- The data stream is a continuous sequence of data points: $\{x_1$,...,$x_{now}\}$, where each x_i is a d-dimensional feature vector. x_1 is the very first data point in the stream, and x_{now} is the latest data point that has just arrived.
- Each data point x_i is associated with two attributes: y_i, and t_i, being its class label, and time of arrival, respectively.
- For simplicity, we assume that t_{i+1}=t_i+1, and t_1=1.
- The latest T_l instances in the stream: $\{x_{now-T_l+1}$,...,$x_{now}\}$ are unlabeled, meaning, their corresponding class labels are unknown. But the class labels of all other data points are known.
- We are to predict the class label of x_{now} before the time t_{now} + T_c, i.e., before the data point x_{now+T_c} arrives, and $T_c < T_l$.

A. Top level algorithm

Algorithm 1 outlines the top level overview of our approach. The algorithm starts with building the initial ensemble of models $L = \{L_1, ..., L_M\}$ with the first M labeled data chunks. The algorithm maintains three buffers: a temporary buffer buf, that keeps potential novel class instances, an unlabeled data buffer U, that keeps unlabeled data points until they are labeled, and labeled data buffer $\mathcal{L}$, that keeps labeled instances until they are used to train a new classifier. After initialization, the while loop begins from line 5, which continues indefinitely. At each iteration of the loop, the latest data point in the stream, x_j is classified (line 7) using **Classify**() (algorithm 2). The novel class detection mechanism is situated inside algorithm 2. If the class of x_j cannot be predicted immediately, it is stored in buf for future processing. Details of this step will be discussed in section IV. x_j is then pushed into the unlabeled data buffer U (line 8). If the buffer size exceeds T_l, the oldest element x_k is dequeued and labeled (line 9), since T_l unit of time has elapsed since x_k arrived in the stream (so it is time to label x_k). The pair $< x_k, y_k >$ is pushed into the labeled data buffer $\mathcal{L}$ (line 9). When we have S instances in $\mathcal{L}$, where S is the chunk size, a new classifier L' is trained using the chunk (line 13). Then the existing ensemble is updated (line 14) by choosing the best M classifiers from the M + 1 classifiers $L \cup \{L'\}$

based on their accuracies on $\mathcal{L}$, and the buffer $\mathcal{L}$ is emptied to receive the next chunk of training data (line 15). Our algorithm will be mentioned henceforth as "ECSMiner" (pronounced like

Algorithm 1 ECSMiner

```
1: $L \leftarrow$ Build-initial-ensemble()
2: buf $\leftarrow$ empty //temporary buffer
3: $U \leftarrow$ empty //unlabeled data buffer
4: $\mathcal{L} \leftarrow$ empty //labeled data buffer (training data)
5: while true do
6:     $x_j \leftarrow$ the latest data point in the stream
7:     Classify($L$,$x_j$,buf) //section IV
8:     $U \Leftarrow x_j$ //enqueue
9:     if $|U| > T_l$ then //time to label the oldest instance
10:        $x_k \Leftarrow U$ //dequeue the instance
11:        $\mathcal{L} \Leftarrow < x_k, y_k >$ //label it and save in training buffer
12:        if $|\mathcal{L}|$ = $S$ then //training buffer is full
13:            $L' \leftarrow$ Train-and-save-decision-boundary($\mathcal{L}$) (section III-E)
14:            $L \leftarrow$ Update($L$,$L'$,$\mathcal{L}$)
15:            $\mathcal{L} \leftarrow$ empty
16:        end if
17:    end if
18: end while
```

ExMiner), which stands for Enhanced Classifier for Data Streams with novel class Miner. We believe that any base learner can be enhanced with the proposed novel class detector, and used in ECSMiner. The only operation that needs to be treated specially for a particular base learner is *Train-and-save-decision-boundary*. We illustrate this operation for two base learners in this section.

B. Nearest neighborhood rule

We assume that the instances belonging to a class c is generated by a an underlying generative model θ_c, and the instances in each class are independently identically distributed. With this assumption, one can reasonably argue that the instances which are close together under some distance metric are supposed to be generated by the same model, i.e., belong to the same class.

This is the basic assumption for nearest-neighbor classifications [5]. Besides, this assumption is used in numerous semi-supervised learning techniques, such as [15], and in many other semi-supervised learning works [26]. We generalize this assumption by introducing the concept of a “nearest neighborhood”.

Definition 1 ($\lambda_{c,q}$-neighborhood): $\lambda_{c,q}$-neighborhood, or $\lambda_{c,q}$(x) of any instance x is the set of q nearest neighbors of x within class c.

For example, let there be three classes c_+, and c_-, and c_0, denoted by the symbols “+”, “-”, and black dots, respectively (figure 2). Also, let q=5. then $\lambda_{c_+,q}(x)$ of any arbitrary instance x is the set of 5 nearest neighbors of x in class c_+, and so on.

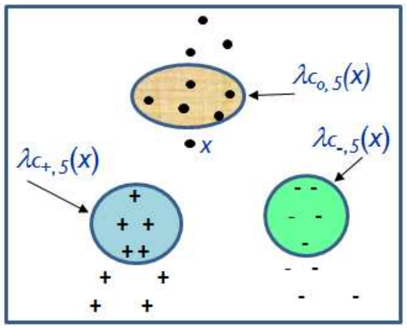


Fig. 2. Illustrating $\lambda_{c,q}(x)$ for q=5

Let $\bar{D}_{c,q}(x)$ be the average distance from x to $\lambda_{c,q}(x)$, i.e.,

$$\bar{D}_{c,q}(x) = \frac{1}{q} \sum_{x_i \in \lambda_{c,q}(x)} D(x, x_i) \tag{1}$$

where $D(x_i, x_j)$ is the distance between the data points x_i and x_j in some appropriate metric.

Let $\lambda_{min,q}$-neighborhood, or $\lambda_{min,q}(x)$ of any instance x be the $\lambda_{c,q}$-neighborhood of x, whose corresponding $\bar{D}_{c,q}(x)$ is the minimum. For example, in figure 2, $\lambda_{min,q}(x)$ = $\lambda_{c_0,q}(x)$.

Definition 2 (q-nearest neighborhood rule (q-NH rule)): Let c_{min} be the class label of the instances in $\lambda_{min,q}(x)$. According to the q-NH rule, the predicted class label of an unlabeled test instance x is c_{min}.

In the example of figure 2, c_{min} = c_0, so x is labeled as class c_0. Our novel class detection technique is based on the assumption that any class of data follow the q-NH rule. In section IV, we discuss the similarity of this rule with k-NN rule, and highlight its significance.

C. Novel class and its properties

Definition 3 (Existing class and Novel class): Let L be the current ensemble of classification models. A class c is an existing class if at least one of the models $L_i \in L$ has been trained with the instances of class c. Otherwise, c is a novel class.

Therefore, if a novel class c appears in the stream, none of the classification models in the ensemble will be able to correctly classify the instances of c. An important property of the novel class follows from the q-NH rule.

Property 1: Let x be an instance belonging to a novel class c, and let c' be an existing class. Then according to q-NH rule, $\bar{D}_{c,q}(x)$, i.e., the average distance from x to $\lambda_{c,q}(x)$ is smaller than $\bar{D}_{c',q}(x)$, the average distance from x to $\lambda_{c',q}(x)$, for any existing class c'. In other words, x is closer to the neighborhood of its own class (cohesion), and farther from the neighborhood of any existing classes (separation).

Figure 3 shows an hypothetical example of a decision tree and the appearance of a novel class. A decision tree and its corresponding feature vector partitioning by its leaf nodes are shown in the figure. The shaded portions of the feature space represents the training data. After the decision tree is built, a novel class appears in the stream (shown with “x” symbol). The decision tree model misclassifies all the instances in the novel class as existing class instance since the model is unaware of the novel class. Our goal is to detect the novel class without having to train the model with that class. Note that instances in the novel class follows property 1, since the novel-class neighborhood of any novel-class instance is much closer to the instance than the neighborhoods of any other classes. If we observe this property in a collection of unlabeled test instances, we can detect the novel class. This is not a trivial task, since we must decide when to classify an instance immediately, and when to postpone the classification decision, and wait for more test instances so that property 1 can be revealed among those instances. Because in order to discover property 1 (cohesion) we need to deal with a collection of test instances simultaneously. Besides, we cannot defer the decision more than T_c time units after the arrival of a test instance.

Therefore, the *main challenges* in novel class detection are as follows:

- Saving the training data efficiently without using much memory.
- Knowing when to classify a test instance immediately, and when to postpone the classifi-

cation decision.

- Classifying the deferred instances within T_c time unit.
- Predicting the presence of a novel class quickly and correctly.

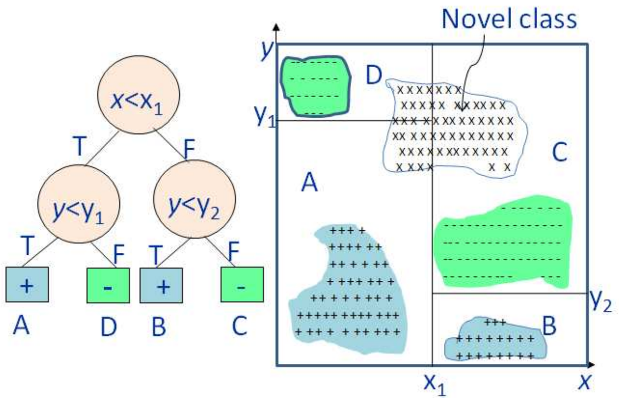


Fig. 3. A hypothetical example.

D. Base learners

We apply our technique on two different classifiers: decision tree, and k-nearest neighbor (k-NN). When decision tree is used as a classifier, each training data chunk is used to build a decision tree. When k-NN is used, each chunk is used to build a k-NN classification model. The simplest way to build such a model is to just store all the data points of the training chunk in memory. But this strategy would lead to a inefficient classification model, both in terms of memory and running time. In order to make the model more efficient, we build K clusters with the training data [12]. We apply a semi-supervised clustering technique using Expectation Maximization (E-M) that tries to minimize both intra-cluster dispersion (same objective as unsupervised K-means) and cluster impurity.

$$\mathcal{O}bj = \sum_{i=1}^{K}\Big(\sum_{\boldsymbol{x}\in\mathcal{X}_i} ||\boldsymbol{x}-\boldsymbol{u_i}||^2 + \sum_{\boldsymbol{x}\in\mathcal{X}_i} ||\boldsymbol{x}-\boldsymbol{u_i}||^2 * Imp_i\Big) \tag{2}$$

The first term in equation 2 is the same as unsupervised K-means, which penalizes intra-cluster dispersion. The second term penalizes cluster impurity. A cluster is considered pure if all data points in the cluster comes from the same class. We use *entropy* and *Gini index* for impurity measure. After building the clusters, we save the cluster summary (mentioned as "pseudopoint") of each cluster (centroid, and frequencies of data points belonging to each class), and discard the raw data points. Since we store and use only K pseudopoints, both the time and memory requirements become functions of K (a constant number). A test instance x_j is classified as follows: we find the pseudopoint whose centroid is nearest from x_j, and assign it a class label that has the highest frequency in that pseudopoint.

E. *Creating decision boundary during training*

The training data are clustered using K-means and the cluster centroids and other statistics of each cluster are saved as pseudopoints. Then the raw training data are discarded. These pseudopoints form a decision boundary for the training data.

Clustering: K clusters are built per chunk from the training data. This clustering step is specific to each base learner. For example, for decision tree, clustering is done at each leaf node of the tree, since we need to create decision boundaries in each leaf node separately. For k-NN, existing clusters are used that were created using the approach discussed in section III-D. For decision tree, clustering is done locally at each leaf node as follows. Suppose S is the chunk-size. During decision tree training, when a leaf node l_i is reached, $k_i = (t_i/S) * K$ clusters are built in that leaf, where t_i denotes the number of training instances belonging to leaf node l_i. Therefore, the number of clusters built in each leaf node is proportional to the number of training instances that belong to the leaf node. If a leaf node is not empty (has one or more instances), then at least one cluster is built in that node.

Storing the cluster summary information: For each cluster, we store the following summary information: i) *Weight*, w: Total number of points in the cluster. ii) *Centroid*, ζ. iii) *Radius*, $\mathcal{R}$: Distance between the centroid and the farthest data point in the cluster. iv) *Mean distance*, μ_d: The mean distance from each point to the cluster centroid. The cluster summary of a cluster will be referred to henceforth as a "pseudopoint" h. So, $w(h)$ denotes the "weight" value of pseudopoint h, and so on. After computing the cluster summaries, the raw data are discarded and only the pseudopoints are stored in memory. Thus, the memory requirement for storing the training data becomes constant, i.e., $O(K)$. Let $\mathcal{H}_i$ denote the set of pseudopoints corresponding to the classifier L_i. Any pseudopoint having too few (less than 3) instances is considered as noise and is not stored in memory.

Each pseudopoint h corresponds to a hypersphere in the feature space having center $\zeta(h)$ and radius $\mathcal{R}(h)$. Let us denote the portion of feature space covered by a pseudopoint h as the "region" of h or $RE(h)$. Let $RE(\mathcal{H}_i)$ denote the union of the regions of all pseudopoints in $\mathcal{H}_i$, i.e.,

$$RE(\mathcal{H}_i) = \cup_{h \in \mathcal{H}_i} RE(h)$$

Therefore, $RE(\mathcal{H}_i)$ forms a decision boundary for the training data of classifier L_i.

IV. Classification with novel class detection

Algorithm 2 (Classify) sketches the classification and novel class detection technique. The algorithm consists of two main parts: classification (lines 1-5) and novel class detection (lines 6-14). Details of the steps of this algorithm will be explained in the following subsections.

Algorithm 2 Classify(L,x_j,buf)

```
Input: $L$: Current ensemble of best $M$ classifiers
    $x_j$: test instance
    $buf$: buffer holding temporarily deferred instances
Output: Immediate or deferred class prediction of $x_j$
 1: fout ← true
 2: if Foutlier($L$,$x_j$) = false then
 3:     $y_i' \leftarrow$ majority-voting($L$,$x_j$) //classify immediately
 4:     fout ← false
 5: end if
 6: Filter($buf$)
 7: if fout = true then
 8:     $buf \Leftarrow x_j$ //enqueue
 9:     if $buf$.length $> q$ and last_trial + q $\leq t_i$ then
10:         last_trial $\leftarrow t_i$
11:         novel ← Detect-NovelClass($L$,$buf$)
12:         if novel = true then remove_novel ($buf$)
13:     end if
14: end if
```

A. *Classification*

In line 2 of algorithm 2 we first check whether the test instance x_j is an $Foutlier$, which is to be defined shortly. If any test instance x_j falls outside the decision boundary $RE(\mathcal{H}_i)$ of a classifier L_i, then x_j is an outlier. If x_j is a novel class instance, it must be an outlier, which would be justified shortly. However, x_j may also appear an outlier because of other reasons: noise, concept-drift, or insufficient training data for L_i. Therefore, we apply filtering so that most of the outliers, which appear for any reason other than being novel class instance, are filtered out. The filtered outliers are called $Foutlier$s.

Definition 4 (Foutlier): A test instance is an Foutlier (i.e., filtered outlier) if it is outside the decision boundary of all classifiers $L_i \in L$.

Intuitively, all novel class instances should be $Foutlier$s. Because, if any test instance x_j is not an $Foutlier$, then it must be inside the decision boundary of some classifier L_i. Therefore, it must be inside $RE(h')$ of some pseudopoint h'. This implies that x_j is closer to the centroid of h' than at least one training instance in h' (the one at the farthest distance from the centroid of h'), which leads to the conclusion that x_j is most likely an existing class instance having the same class label as the instances in h'. So, if x_j is not an $Foutlier$, we classify it immediately using the ensemble voting (line 3).

B. *Novel class detection*

The buffer *buf* temporarily holds potential novel class instances. These instances are analyzed periodically in order to detect novel class, which is explained in the next paragraph. buf needs to be cleared periodically (line 6, algorithm 2) to remove instances that no longer contribute to novel class detection. Besides, instances in the buffer that has reached classification deadline T_c are classified immediately. An instance is removed from buf if it fulfills either of the three conditions:

1) Age $>$ S: the front of the buffer contains the oldest element in the buffer. It is removed if its age is greater than S, the chunk size.
2) Ensemble update: the ensemble may be updated while an instance x_k is waiting inside the buffer. As a result, x_k may no longer be an $Foutlier$ for the new ensemble of models, and it must be removed if so. If x_k is no longer an $Foutlier$, and it is not removed, it

could be falsely identified as a novel class instance, and also it could interfere with other valid novel class instances, misleading the detection process.

3) Existing class: any instance is removed from buf if it has been labeled, and it belongs to one of the existing classes.

When an instance is removed from buf, it is classified immediately using the current ensemble (if not classified already).

Lines (7-14) are executed only if x_j is an $Foutlier$. At first, x_j is enqueued into the buffer (line 9). Then we check whether the buffer length is at least q, and the last check on buf for detecting novel class had been executed (*last_trial*) at least q time unit earlier (line 10). Since novel class detection is more expensive than simple classification, this operation is performed at most once in every q time unit. In line 11, algorithm 3 (DetectNovelClass) is called, which returns true of a novel class is found. Finally, if a novel class is found, all instances that are identified as novel class are removed from buf (line 12).

Next, we examine algorithm 3 to understand how buf is analyzed to detect presence of novel class. First, we define q-neighborhood silhouette coefficient, or q-NSC, as follows:

Definition 5 (q-NSC): Let $\bar{D}_{q,c_{out}}(x)$ be the mean distance from an $Foutlier$ x to $\lambda_{q,c_{out}}(x)$ defined by equation 1, where $\lambda_{q,c_{out}}(x)$ is the set of q-nearest neighbors of x within the $Foutlier$ instances. Also, let $\bar{D}_{q,c_{min}}(x)$ be the minimum among all $\bar{D}_{q,c}(x)$, where c is an existing class. Then q-NSC of x is given by:

$$q\text{-}NSC(x) = \frac{\bar{D}_{q,c_{min}}(x) - \bar{D}_{q,c_{out}}(x)}{max(\bar{D}_{q,c_{min}}(x), \bar{D}_{q,c_{out}}(x))} \tag{3}$$

q-NSC, which is a unified measure of cohesion and separation, yields a value between -1 and +1. A negative value indicates that x is closer to the existing classes (less separation) and farther away from other $Foutlier$s, and vice versa. We declare a *new class* if there are at least q' $(> q)$ $Foutlier$s having positive q-NSC. The justification behind this decision is discussed in the next subsection.

Speeding up the computation of q-NSC: Computing q-NSC for every $Foutlier$ instance x takes quadratic time in the number of $Foutlier$s. In order to make the computation faster, we also create K_o (= (buf.length/S) * K) pseudopoints from $Foutlier$s using K-means clustering and perform the computations on the pseudopoints (referred to as $Fpseudopoint$s), where S is the chunk size. Thus, the time complexity to compute the q-NSC of all of the $Fpseudopoint$s is $O(K_o * (K_o + K))$, which is constant, since both K_o and K are independent of the input size. Note that q-NSC of a $Fpseudopoint$ is actually an approximate average of the q-NSC of each $Foutlier$ in that $Fpseudopoint$. By using this approximation, although we gain speed, we also lose some precision. However, this drop in precision is negligible, as shown in the analysis to be presented shortly.

In line 1 of algorithm 3, we create $Fpseudopoint$s using the $Foutlier$s as explained earlier. For each classifier $L_i \in L$, we compute q-NSC(h) of every $Fpseudopoint$ h (line 4). If the total weight of the $Fpseudopoint$s having positive q-NSC() is greater than q, then L_i votes for novel class. If all classifiers vote for novel class, then we ultimately declare a novel class. Once novel class is declared, we need to find the instances of the novel class. This is done as follows: suppose h is an $Fpseudopoint$ having positive q-NSC(h) with respect to all classifiers $L_i \in L$ (note that q-NSC(h) is computed with respect to each classifier separately). Therefore, all $Foutlier$ instances belonging to h are identified as novel class instances.

Algorithm 3 DetectNovelClass(L,buf)

```
Input: $L$: Current ensemble of best $M$ classifiers
  $buf$: buffer holding temporarily deferred instances
Output: true, if novel class is found; false, otherwise
1: Make $K_o$=($K$*$buf$.length/$S$) clusters with the instances in $buf$ using $K$-means clustering, and create $K_o$
   $Fpseudopoint$s
2: Let $\mathcal{H}_o$ be the set of $Fpseudopoint$s
3: for each classifier $L_i \in L$ do
4:     for each $h \in \mathcal{H}_o$ do Compute $q$-NSC($h$)
5:     $\mathcal{H}_p \leftarrow \{h | h \in \mathcal{H}_o$ and $q$-NSC($h$) $> 0$ $\}$ //Fpseudopoints with positive $q$-NSC
6:     $w(\mathcal{H}_p) \leftarrow \sum_{h \in \mathcal{H}_p} w(h)$. //sum of their weights
7:     if $w(\mathcal{H}_p) > q$ then NewClassVote++
8: end for
9: found $\leftarrow$ NewClassVote $==$ $M$
```

This algorithm can detect one or more novel classes concurrently as long as each novel class follows property 1 and contains at least q instances. This is true even if the class distributions are skewed. However, if more than one such novel classes appear concurrently, our algorithm will identify the instances belonging those classes as novel, without imposing any distinction between dissimilar novel class instances (i.e., it will treat them simply as “novel”). But the distinction will be learned by our model as soon as the true labels of those novel class instances arrive, and a classifier is trained with those instances.

It should be noted that the larger the value of q, the greater the confidence with which we can decide whether a novel class has arrived. However, if q is too large, then we may also fail to detect a new class if the total number of instances belonging to the novel class in the corresponding data chunk is $\leq q$. An optimal value of q is obtained empirically (section V).

Impact of evolving class labels on ensemble classification: As reader might have realized already, arrival of novel classes in the stream causes the classifiers in the ensemble to have different sets of class labels. There are two scenarios to consider. Scenario (a): suppose an older (earlier) classifier L_i in the ensemble has been trained with classes c_0 and c_1, and an younger (later) classifier L_j has been trained with classes c_1, and c_2, where c_2 is a new class that appeared after L_i had been trained. This puts a negative effect on voting decision, since the older classifier mis-classifies instances of c_2. So, rather than counting the votes from each classifier, we selectively count their votes as follows: if an younger classifier L_j classifies a test instance x as class c, but an older classifier L_i does not have the class label c in its model, then the vote for L_i will be ignored if x is found to be an outlier for L_i. Scenario (b): the opposite situation may also arise where the oldest classifier is trained with some class c', but none of the newer classifiers are trained with that class. This means class c' has been outdated, and in that case, we remove L_i from the ensemble. Figure 4 (a) illustrates scenario (a). The classifier in the ensemble are sorted according to their age, with L_1 being the oldest, and L_4 being the youngest. Each classifier L_i is marked with the classes with which it has been trained. For example, L_1 is trained with classes c_1, c_2, and c_3, and so on. Note that class c_4 appears only in the two youngest classifiers. x appears as an outlier to L_1. Therefore, L_1's vote is not counted since x is classified as c_4 by an younger classifier L_3, and L_1 does not contain class c_4. Figure 4 (b) illustrates scenario (b). Here L_1 contains class c_1, which is not contained by any younger classifiers in the ensemble. Therefore, c_1 has become outdated, and L_1 is removed from the ensemble. In this way we ensure that older classifiers have less impact in the voting process. If class c_1 later re-appears in the stream, it will be automatically detected again as a novel class (see definition 3).

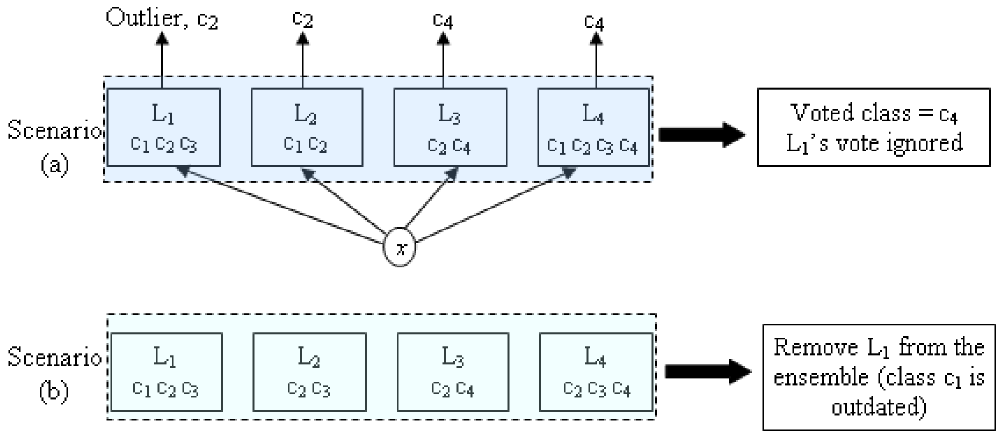


Fig. 4. Impact of evolving class label on ensemble

C. Analysis and discussion

In this subsection at first we justify the novel class detection algorithm, then analyze the extent of precision loss in computing q-NSC, and finally analyze the time complexity of ECSMiner.

Justification of the novel class detection algorithm: In algorithm 3, we declare a novel class if there are at least $q' > q$ $Foutliers$ that have positive q-NSC for all the classifiers in the ensemble. First, we illustrate the significance of this condition, i.e., "more than q $Foutlier$s have poisitive q-NSC". Equation (3) establishes a relationship between i) the mean distance from an $Foutlier$ instance x to *all* instances in its nearest $Foutlier$ neighborhood, and ii) the mean distance from x to *all* instances in its nearest existing class neighborhood. Now we go one step further to establish a relationship between i) the distance from x to a *single* instance in its nearest $Foutlier$ neighborhood, and ii) the distance from x to a *single* instance in its nearest existing class neighborhood.

Let $\mathcal{F}$ be the set of $Foutlier$s having positive q-NSC. Therefore, for any $x \in \mathcal{F}$:

$$\bar{D}_{q,c_{min}}(x) - \bar{D}_{q,c_{out}}(x) > 0 \qquad \text{(from equation 3)}$$

$$\Rightarrow \bar{D}_{q,c_{min}}(x) > \bar{D}_{q,c_{out}}(x)$$

Summing up for all $Foutlier$s $x \in \mathcal{F}$:

$$\begin{aligned}
&\sum_{x\in\mathcal{F}} \bar{D}_{q,c_{min}}(x) > \sum_{x\in\mathcal{F}} \bar{D}_{q,c_{out}}(x) \\
&\Rightarrow \sum_{x\in\mathcal{F}} \frac{1}{q} \sum_{x_i\in\lambda_{c_{min},q}(x)} D(x,x_i) > \sum_{x\in\mathcal{F}} \frac{1}{q} \sum_{x_j\in\lambda_{c_{out},q}(x)} D(x,x_j) \quad \text{(from equation 1)} \\
&\Rightarrow \frac{1}{m}\frac{1}{q} \sum_{x\in\mathcal{F}} \sum_{x_i\in\lambda_{c_{min},q}(x)} D(x,x_i) > \frac{1}{m}\frac{1}{q} \sum_{x\in\mathcal{F}} \sum_{x_j\in\lambda_{c_{out},q}(x)} D(x,x_j) \text{ (letting m = } |\mathcal{F}|)
\end{aligned} \tag{4}$$

Therefore, the mean pairwise distance between any pair of $Foutlier$s (x,x_j), (such that x is an $Foutlier$ with positive q-NSC and x_j is an $Foutlier$ in the q-nearest neighborhood of x), is less than the mean pairwise distance between an $Foutlier$ x and any existing class instance x_i. In other words, an $Foutlier$ with positive q-NSC is more likely to have majority of its k-nearest neighbors (k-NNs) within the $Foutlier$ instances. So, each of the $Foutlier$s $x \in \mathcal{F}$ should have the same class as the $Foutlier$ instances, and should have a different class than any of the existing classes. The higher the value of q, the larger the support we have in favor of the arrival of a new class. Furthermore, when all the classifiers unanimously agree on the arrival of a novel class, we have very little choice other than announcing the appearance of a novel class. The q-NH rule can be thought of another way of expressing the k-NN rule. Therefore, this rule is applicable to any dataset irrespective of its data distribution, and shape of classes (e.g. convex and non-convex).

Computing precision loss in approximate q-NSC computation: As discussed earlier, we compute q-NSC for each $Fpseudopoint$, rather than each $Foutler$ individually in order to reduce time complexity, which causes a reduction of precision in computation. However, following analysis shows that this loss is negligible. Without loss of generality, let ϕ_i be an $Fpseudopoint$ having weight q_1, and ϕ_j be an existing class Pseudopoint having weight q_2, which is nearest from ϕ_i (figure 5). We compute the approximate q-NSC of ϕ_i (q-$NSC'(\phi_i)$) using the following formula:

$$q\text{-}NSC'(\phi_i) = \frac{D(\mu_i,\mu_j) - \bar{D}_i}{max(D(\mu_i,\mu_j),\bar{D}_i)} \tag{5}$$

Where μ_i is the centroid of ϕ_i, μ_j is the centroid of ϕ_j, and $\bar{D}_i$ is the mean distance from centroid μ_i to the instances in ϕ_i. Equation (5) is an approximate of the real q-NSC. The exact

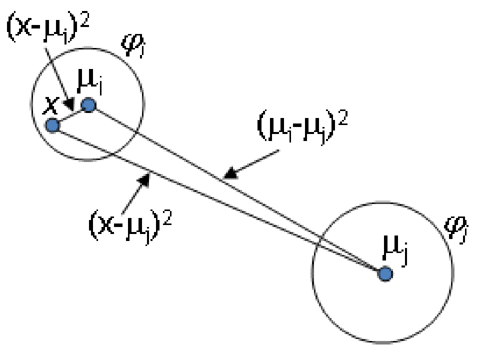


Fig. 5. Illustrating the computation of precision loss. ϕ_i is an $Fpseudopoint$, i,e., a cluster of $Foutlier$s, and ϕ_j is an existing class Pseudopoint, i.e., a cluster of existing class instances. In this particular example, all instances in ϕ_i belong to a novel class.

value of q-NSC follows from equation (3):

$$q\text{-}NSC(\phi_i) = \frac{1}{q_1}\sum_{x\in\phi_i}\frac{\frac{1}{q}\sum_{x_j\in\lambda_{q,j}(x)}D(x,x_j)-\frac{1}{q}\sum_{x_i\in\lambda_{q,i}(x)}D(x,x_i)}{max(\frac{1}{q}\sum_{x_j\in\lambda_{q,j}(x)}D(x,x_j),\frac{1}{q}\sum_{x_i\in\lambda_{q,i}(x)}D(x,x_i))} \tag{6}$$

Where $\lambda_{q,i}(x)$ is the q-nearest neighborhood of x within $FPseudopoint$ ϕ_i, and $\lambda_{q,j}(x)$ is the q-nearest neighborhood of x within $FPseudopoint$ ϕ_j, for some $x \in \phi_i$. Therefore, the precision loss is their difference: q-NSC(ϕ_i) - q-NSC'(ϕ_i)

$$\mathcal{L}_{prec} = \frac{1}{q_1}\sum_{x\in\phi_i}\frac{\frac{1}{q}\sum_{x_j\in\lambda_{q,j}(x)}D(x,x_j)-\frac{1}{q}\sum_{x_i\in\lambda_{q,i}(x)}D(x,x_i)}{max(\frac{1}{q}\sum_{x_j\in\lambda_{q,j}(x)}D(x,x_j),\frac{1}{q}\sum_{x_i\in\lambda_{q,i}(x)}D(x,x_i))} - \frac{D(\mu_i,\mu_j)-\bar{D}_i}{max(D(\mu_i,\mu_j),\bar{D}_i)} \tag{7}$$

In order to simplify the computations, we assume that $q_1 = q_2 = q$, and q-NSC is positive for any $x \in \phi_i$. Therefore, $\lambda_{q,i}(x) = \phi_i$, $\lambda_{q,j}(x) = \phi_j$. Also, we consider square of Eucledian distance as the distance metric, i.e., $D(x, y) = (x - y)^2$. Since q-NSC is positive for any $x \in \phi_i$, we have $max(D(\mu_i, \mu_j), \bar{D}_i) = D(\mu_i, \mu_j)$, and $max(\frac{1}{q}\sum_{x_j\in\lambda_{q,j}(x)}D(x,x_j),\frac{1}{q}\sum_{x_i\in\lambda_{q,i}(x)}D(x,x_i))$ = $\frac{1}{q}\sum_{x_j\in\lambda_{q,j}(x)}D(x,x_j)$. Also, $\bar{D}_i = \frac{1}{q}\sum_{x\in\phi_i}(x-\mu_i)^2 = \sigma_i^2$, the mean distance of the instances in ϕ_i from the centroid. Continuing from equation (7):

$$\begin{aligned}\mathcal{L}_{prec} =& \frac{1}{q}\sum_{x\in\phi_i}\frac{\frac{1}{q}\sum_{x_j\in\phi_j}(x-x_j)^2-\frac{1}{q}\sum_{x_i\in\phi_i}(x-x_i)^2}{\frac{1}{q}\sum_{x_j\in\phi_j}(x-x_j)^2} - \frac{(\mu_i-\mu_j)^2-\sigma_i^2}{(\mu_i-\mu_j)^2}\\ =& \frac{1}{q}\sum_{x\in\phi_i}(\frac{\frac{1}{q}\sum_{x_j\in\phi_j}(x-x_j)^2-\frac{1}{q}\sum_{x_i\in\phi_i}(x-x_i)^2}{\frac{1}{q}\sum_{x_j\in\phi_j}(x-x_j)^2} - \frac{(\mu_i-\mu_j)^2-(x-\mu_i)^2}{(\mu_i-\mu_j)^2})\end{aligned}$$

It is easy to show that $\frac{1}{q}\sum_{x\in\phi_i}(x-x_i)^2-(x-u_i)^2 = \sigma_i^2$ and $\frac{1}{q}\sum_{x\in\phi_j}(x-x_j)^2-(x-u_j)^2 = \sigma_j^2$. Substituting these values, we obtain:

$$\begin{aligned}
\mathcal{L}_{prec} =& \frac{1}{q}\sum_{x\in\phi_i}(\frac{\sigma_j^2+(x-\mu_j)^2-\sigma_i^2-(x-\mu_i)^2}{\sigma_j^2+(x-\mu_j)^2} - \frac{(\mu_i-\mu_j)^2-(x-\mu_i)^2}{(\mu_i-\mu_j)^2}) \\
=& \frac{1}{q}\sum_{x\in\phi_i}(1-\frac{\sigma_i^2+(x-\mu_i)^2}{\sigma_j^2+(x-\mu_j)^2} - 1 + \frac{(x-\mu_i)^2}{(\mu_i-\mu_j)^2}) \\
=& \frac{1}{q}\sum_{x\in\phi_i}(\frac{(x-\mu_i)^2}{(\mu_i-\mu_j)^2} - \frac{\sigma_i^2+(x-\mu_i)^2}{\sigma_j^2+(x-\mu_j)^2}) \\
=& \frac{\sigma_i^2}{(\mu_i-\mu_j)^2} - \frac{1}{q}\sum_{x\in\phi_i}\frac{\sigma_i^2}{\sigma_j^2+(x-\mu_j)^2} - \frac{1}{q}\sum_{x\in\phi_i}\frac{(x-\mu_i)^2}{\sigma_j^2+(x-\mu_j)^2}) \\
\leq& \frac{\sigma_i^2}{(\mu_i-\mu_j)^2} - \frac{\sigma_i^2}{\sigma_i^2+\sigma_j^2+(\mu_i-\mu_j)^2} - \frac{1}{q}\sum_{x\in\phi_i}\frac{(x-\mu_i)^2}{\sigma_j^2+(x-\mu_j)^2}
\end{aligned}$$

The last line follows since using the relationship between harmonic mean and arithmetic mean it can be shown that:

$$\frac{1}{q}\sum_{x\in\phi_i}\frac{\sigma_i^2}{\sigma_j^2+(x-\mu_j)^2} \geq \frac{\sigma_i^2}{\frac{1}{q}\sum_{x\in\phi_i}\sigma_j^2+(x-\mu_j)^2} = \frac{\sigma_i^2}{\sigma_j^2+\sigma_i^2+(\mu_i-\mu_j)^2}$$

Following the fact that all $x\in\phi_i$ has positive q-NSC, we can deduce the following relationships: i) $(x-\mu_i)^2 \leq (\mu_i-\mu_j)^2$, ii) $(x-\mu_j)^2 \leq (x-\mu_i)^2+(\mu_i-\mu_j)^2$: by triangle inequality. iii) $\sigma_i^2 \leq (\mu_i-\mu_j)^2$: since $\sigma_i^2 = \frac{1}{q}\sum_{x\in\phi_i}(x-\mu_i)^2 \leq \frac{1}{q}\sum_{x\in\phi_i}(\mu_i-\mu_j)^2 = (\mu_i-\mu_j)^2$. iv) $\sigma_j^2 \leq (\mu_i-\mu_j)^2$: because ϕ_j represents an existing class, and similar reasoning as iii) can be applied here too.

Applying these relationships, and after several algebraic manipulations, we obtain:

$$\mathcal{L}_{prec} \leq \frac{\sigma_i^2}{(\mu_i-\mu_j)^2} - \frac{\sigma_i^2}{3(\mu_i-\mu_j)^2} - \frac{\sigma_i^2}{3(\mu_i-\mu_j)^2} = \frac{\sigma_i^2}{3(\mu_i-\mu_j)^2} \tag{8}$$

Usually, if ϕ_i belongs to a novel class, it is empirically observed in almost all datasets that q-$NSC'(\phi_i) \geq 0.9$. In this case, $\sigma_i^2 \leq (1-0.9)(\mu_i-\mu_j)^2$. Therefore, $\mathcal{L}_{prec} \leq 1/30$, which is a negligible loss.

Time and space complexity: Line 1 of algorithm 3 (clustering) takes $O(KS)$ time, and the for loop (lines 3-8) takes $O(K^2L)$ time. The overall time complexity of algorithm 3 is $O(K^2L+KS)$ = $O(KS)$, since $S >> KL$. Lines 1-5 of algorithm 2 takes $O(KLS + Lf_c(S))$ per chunk, where $f_c(S)$ is the time to classify an instance using a classifier. Line 6 takes $O(S)$ time.

Line 11 (algorithm 3) is executed at most once in every q time unit. Therefore, the worst case complexity of lines 7-14 is $O(Sq^{-1}KS$. So, the overall complexity of algorithm 2 is $O(KLS + Lf_c(S) + Sq^{-1}KS)$ per chunk. For most classifiers, $f_c(S) = O(S)$. Also, let $S/q = m$. So, the overall complexity of algorithm 2 becomes $O(KLS + LS + mS)$ = $O(mS)$, since $m >> KL$. Finally, the overall complexity of algorithm 1 (ECSMiner) is $O(mS + f_t(S))$ per chunk, where $f_t(S)$ is the time to train a classifier with S training instances.

ECSMiner keeps three buffers: buf, the training buffer, and the unlabeled data buffer. Both buf and hold at most S instances, whereas the unlabeled data buffer holds at most T_l instances. Therefore, the space required to store all three buffers is: $O(max(S, T_l))$. The space required to store a classifier (along with the pseudopoints) is much less than S. So, the overall space complexity remains $O(max(S, T_l))$.

V. Experiments

In this section we describe the datasets, experimental environment, and discuss and analyze the results.

A. Data sets

Synthetic data with only concept-drift (SynC): SynC simulates only concept-drift, with no novel classes. This is done to show that concept-drift does not erroneously trigger a new-class detection in our approach. SynC data are generated with a moving hyperplane. The equation of a hyperplane is as follows: $\sum_{i=1}^{d} a_i x_i = a_0$. If $\sum_{i=1}^{d} a_i x_i \leq a_0$, then an example is negative, otherwise it is positive. Each example is a randomly generated d-dimensional vector $\{x_1, ..., x_d\}$, where $x_i \in [0, 1]$. Weights $\{a_1, ..., a_d\}$ are also randomly initialized with a real number in the range [0, 1]. The value of a_0 is adjusted so that roughly the same number of positive and negative examples are generated. This can be done by choosing $a_0 = \frac{1}{2}\sum_{i=1}^{d} a_i$. We also introduce noise randomly by switching the labels of p% of the examples, where p=5 is set in our experiments.

There are several parameters that simulate concept drift. Parameter m specifies the percent of total dimensions whose weights are involved in changing, and it is set to 20%. Parameter t specifies the magnitude of the change in every N examples. In our expeiments, t is varied from 0.1 to 1.0, and N is set to 1000. $s_i, i \in \{1, ..., d\}$ specifies the direction of change for each weight. Weights change continuously, i.e., a_i is adjusted by $s_i.t/N$ after each example is generated. There is a possibility of 10% that the change would reverse direction after every N examples are generated. We generate a total of 250,000 records and generate equal-sized chunks.

Synthetic data with concept-drift and novel-class (SynCN): This synthetic data simulates both concept-drift and novel-class. Data points belonging to each class are generated by following a Normal distribution having different mean (-5.0 to +5.0) and variance (0.5 to 6) for different classes. Besides, in order to simulate the evolving nature of data streams, the probability distributions of different classes are varied with time. This caused some classes to appear and some other classes to disappear at different time slots. In order to introduce concept-drift, the mean values of a certain percentage of attributes have been shifted at a constant rate. As done in the SynC dataset, this rate of change is also controlled by the parameters m, t, s, and N in a similar way. The dataset is normalized so that all attribute values fall within the range [0,1]. We generate the SynCN dataset with 20 classes, and 40 real valued attributes, having a total of 400K data points.

Real data - KDDCup 99 network intrusion detection: We have used the 10% version of the dataset, which is more concentrated, hence more challenging than the full version. It contains around 490,000 instances. Here different classes appear and disappear frequently, making the new class detection challenging. This dataset contains TCP connection records extracted from LAN network traffic at MIT Lincoln Labs over a period of two weeks. Each record refers to either to a normal connection or an attack. There are 22 types of attacks, such as buffer-overflow, portsweep, guess-passwd, neptune, rootkit, smurf, spy, etc. So, there are 23 different classes of data. Most of the data points belong to the normal class. Each record consists of 42 attributes, such as connection duration, the number bytes transmitted, number of root accesses, etc. we use only the 34 continuous attributes, and remove the categorical attributes. This dataset is also normalized to keep the attribute values within [0,1].

Real data - Forest cover (UCI repository): The dataset contains geospatial descriptions of different types of forests. It contains 7 classes and 54 attributes and around 581,000 instances. We normalize the dataset, and arrange the data so that in any chunk at most 3 and at least 2 classes co-occur, and new classes appear randomly.

B. Experimental setup:

We implement our algorithm in Java. The code for decision tree has been adapted from the Weka machine learning open source repository
(http://www.cs.waikato.ac.nz/ml/weka/). The experiments were run on an Intel P-IV machine with 2GB memory and 3GHz dual processor CPU. Our parameter settings are as follows, unless mentioned otherwise: i) K (number of pseudopoints per chunk) = 50, ii) q (minimum number of instances required to declare novel class) = 50, iii) M (ensemble size) = 6, iv) S (chunk size) = 2,000. These values of parameters are tuned to achieve an overall satisfactory performance.

C. Baseline method:

To the best of our knowledge, there is no approach that can classify data streams and detect novel class. So, we compare MineClass with a combination of two baseline techniques: $OLINDDA$ [18], and Weighted Classifier Ensemble (WCE) [22], where the former works as novel class detector, and the latter performs classification. This is done as follows: For each chunk, we first detect the novel class instances using $OLINDDA$. All other instances in the chunk are assumed to be in the existing classes, and they are classified using WCE. We use $OLINDDA$ as the novelty detector, since it is a recently proposed algorithm that is shown to have outperformed other novelty detection techniques in data streams [18].

However, $OLINDDA$ assumes that there is only one "normal" class, and all other classes are "novel". So, it is not directly applicable to the multi-class novelty detection problem, where any combination of classes can be considered as the "existing" classes. We propose two alternative solutions. First, we build parallel $OLINDDA$ models, one for each class, which evolve simultaneously. Whenever the instances of a novel class appear, we create a new $OLINDDA$ model for that class. A test instance is declared as novel, if *all the existing class models* identify this instance as novel. We will refer to this baseline method as WCE-OLINDDA_PARALLEL. Second, we initially build an $OLINDDA$ model with all the available classes. Whenever a novel class is found, the class is absorbed into the existing $OLINDDA$ model. Thus, only one "normal" model is maintained throughout the stream. This will be referred to as WCE-OLINDDA_SINGLE. In all experiments, the ensemble size and chunk-size are kept the same for both these techniques. Besides, the same base learner is used for WCE and XM. The parameter settings for $OLINDDA$ are: i) number of clusters built in the initial model, K =

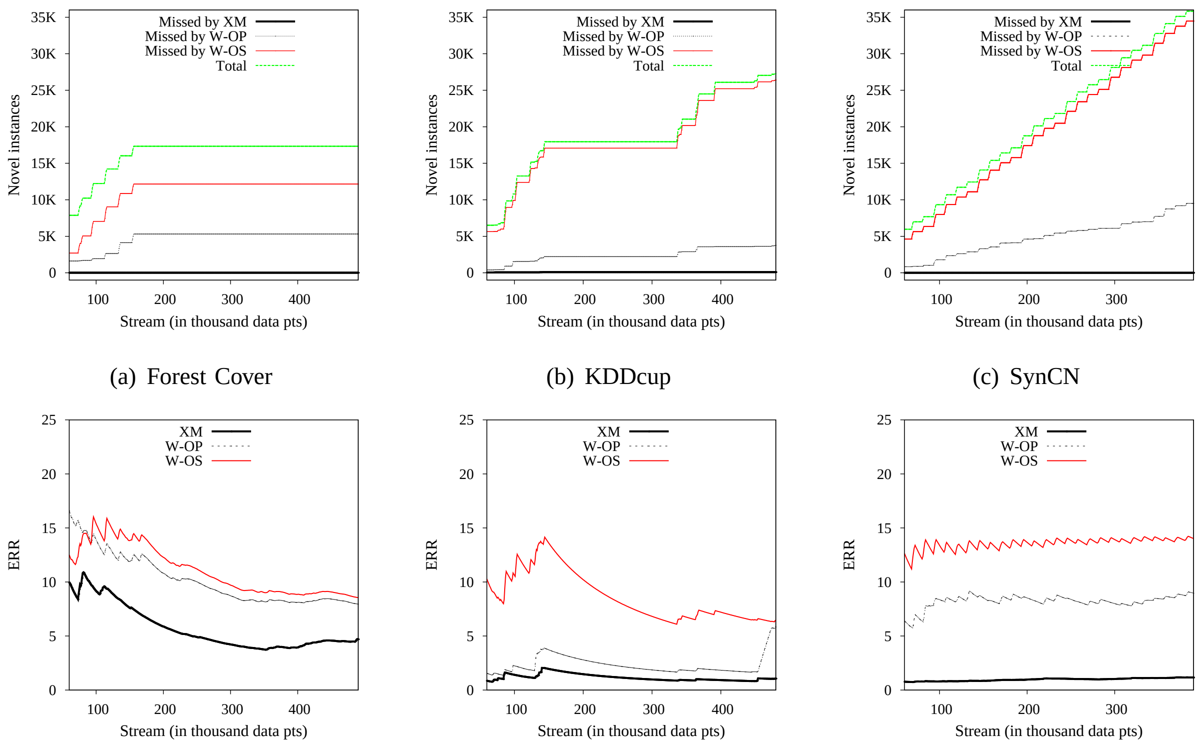
30, ii) least number of normal instances needed to update the existing model = 100, iii) least number of instances needed to build the initial model = 100, iv) maximum size of the "unknown memory" = 200. These parameters are chosen either according to the default values used in [18] or by trial and error to get an overall satisfactory performance. *We will henceforth use the acronyms **XM for ECSMiner**, **W-OP for WCE-OLINDDA_PARALLEL** and **W-OS for WCE-OLINDDA_SINGLE**.*

D. Performance study

Evaluation approach: We use the following performance metrics to evaluate our technique: $\boldsymbol{M_{new}}$ = % of novel class instances Misclassified as existing class = $\frac{F_n*100}{N_c}$, $\boldsymbol{F_{new}}$ = % of existing class instances Falsely identified as novel class = $\frac{F_p*100}{N-N_c}$, **ERR** = Total misclassification error (%)(including M_{new} and F_{new}) = $\frac{(F_p+F_n+F_e)*100}{N}$, where F_n = total novel class instances misclassified as existing class, F_p = total existing class instances misclassified as novel class, F_e = total existing class instances misclassified (other than F_p), N_c = total novel class instances in the stream, N = total instances the stream. From the definition of the error metrics, it is clear that ERR is not necessarily equal to the sum of M_{new} and F_{new}.

Evaluation is done as follows: we build the initial models in each method with the first 3 chunks. From the 4th chunk onward, we evaluate the performances of each method on each data point using the time constraints. We update the models with a new chunk whenever all data points in that chunk is labeled.

Results: Figures 6(a)-(c) show the total number of novel class instances missed (i.e., misclassified as existing class) and Figures 6(d)-(f) show the overall error rates (ERR) of each of the techniques for decision tree classifier up to a certain point in the stream in different datasets. We omit SynC from the figures since it does not have any novel class. K-NN classifier also has similar results. For example, in figure 6(a) at X axis = 100, the Y values show the total number of novel class instances missed by each approach in the first 100K data points in the stream (Forest Cover). At this point, XM misses only 15 novel class instances, whereas W-OP, and W-OS misses 1937, and 7053 instances, respectively. Total number of novel class instances appeared in the stream by this point of time is shown by the corresponding Y value of the curve "Total", which is 12,226. Likewise, in figure 6(d), the ERR rates are shown throughout the stream history. In this figure, at the same position (X=100), Y values show the ERR of each of the three techniques upto the first 100K data points in the stream. The ERR rates of XM, W-OP, and W-OS at this point are: 9.2%, 14.0%, and 15.5%, respectively.



(a) Forest Cover (b) KDDcup (c) SynCN

(d) Forest Cover (e) KDDcup (f) SynCN

Fig. 6. Top row: novel class instances missed by each method, bottom row: overall error of each method (T_l=1000, T_c=400)

Table I summarizes the error metrics for each of the techniques in each dataset for decision tree and KNN. The columns headed by ERR, M_{new} and F_{new} report the average of the corresponding metric on an entire dataset. For example, while using decision tree in KDD dataset, XM, W-OP, and W-OS have 1.0%, 5.8%, and 6.7% ERR, respectively. Also, their corresponding M_{new} are 1.0%, 13.2% and 96.9%, respectively. Note that there is no novel class in SynC, and so, there is no M_{new} for any approach. Both W-OP and W-OS have some F_{new} in SynC dataset, which appears since W-OP and W-OS are less sensitive to concept-drift than XM. Therefore, some existing class instances are misclassified as novel class because of concept drift. All approaches have lower error rates in SynCN than SynC because SynCN is generated using Gaussian distribution, which is naturally easier for the classifiers to learn. Also, SynC has lower error rates with K-NN than with decision tree since the hyperplane dataset is easire to learn for K-NN. In general, XM

TABLE I

PERFORMANCE COMPARISON

Classifier	Dataset	ERR			M_{new}			F_{new}		
		XM	W-OP	W-OS	XM	W-OP	W-OS	XM	W-OP	W-OS
Decision tree	SynC	**6.9**	14.1	12.8	-	-	-	**0.0**	2.4	1.1
	SynCN	**1.2**	8.9	13.9	**0.0**	26.5	96.2	**0.02**	1.6	0.1
	KDD	**1.0**	5.8	6.7	**1.0**	13.2	96.9	0.9	4.3	**0.03**
	Forest Cover	**4.7**	7.9	8.5	**0.2**	30.7	70.1	3.0	1.1	**0.2**
K-NN	SynC	**0.0**	2.4	1.1	-	-	-	**0.0**	2.4	1.1
	SynCN	**0.01**	8.9	13.9	**0.0**	26.5	96.2	**0.0**	1.6	0.1
	KDD	**1.2**	4.9	5.2	**5.9**	12.9	96.5	0.9	4.4	**0.03**
	Forest Cover	**3.6**	4.1	4.6	**8.4**	32.0	70.1	1.3	1.1	**0.2**

outperforms the baseline techniques in overall classification accuracy and novel class detection. The main reason behind the poorer performance of W-OP in detecting novel classes is the way OLINDDA detects novel class. Simply said, OLINDDA makes two strong assumptions about a novel class and normal classes. First, it assumes a spherical boundary of the normal model. It updates the radius of the sphere periodically, and declares anything outside the sphere as a novel class if there is evidence of sufficient cohesion among the instances outside the boundary. The assumption that a data class would have spherical boundary is too strict to be maintained for real world problem. Second, it assumes that the data density of a novel class must be at least that of the normal class. If a novel class is more sparse than the normal class, the instances of that class would never be recognized as a novel class. But in a real world problem, two different classes may have different data densities. OLINDDA would fail in those cases where any of the assumptions are violated. On the other hand, XM does not assume any spherical boundary of an existing class, or similar data densities of different classes. Therefore, XM can detect novel classes much more efficiently. Besides, OLINDDA is less sensitive to concept-drift, which results in falsely declaring novel classes when drift occurs in the existing class data. W-OS performs worse than W-OP since W-OS “assimilates” the novel classes into the normal model, making the normal model too generalized. Therefore, it considers most of the future novel classes as normal (non-novel) data, yielding very high false negative rate.

Figures 7(a),(b) show how XM and W-OP respond to the constraints T_l and T_c in Forest Cover

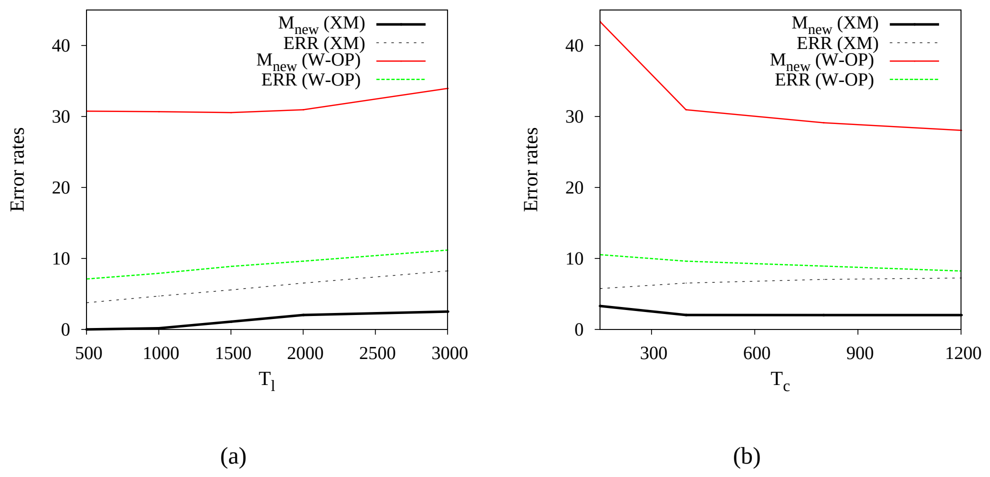


Fig. 7. M_{new} and overall error (ERR) rates on Forest Cover dataset for (a) T_c=400 and different values of T_l and (b) T_l = 2000 and different values of T_c

dataset. Similar characteristics are observed for other datasets and W-OS. From figure 7(a) it is evident that increasing T_l increases error rates. This is because of the higher delay involved in labeling, which makes the newly trained models more outdated. Naturally, M_{new} rate decreases with increasing T_c as shown in figure 7(b) because higher values of T_c means more time to detect novel classes. As a result, ERR rates also decreases.

Figures (8(a)-(d)) illustrate how the error rates of XM change for different parameter settings on Forest Cover dataset and decision tree classifier. These parameters have similar effects on other datasets, and K-NN classifier. Figure 8(a) shows the effect of chunk size on ERR, F_{new}, and M_{new} rates for default values of other parameters. We note that ERR and F_{new} rates decrease upto a certain point (2,000) then increases. The initial decrement occurs because larger chunk size means more training data for the classifiers, which leads to lower error rates. However, if chunk size is increased too much, then we have to wait much longer to build the next classifier. As a result, the ensemble is updated less frequently than desired, meaning, the ensemble remains outdated for longer period of time. This causes increased error rates.

Figure 8(b) shows the effect of ensemble size (M) on error rates. We observe that the ERR and F_{new} rates keeps decreasing with increasing M. This is because when M is increased, classification error naturally decreases because of the reduction of error variance [21]. But the rate of decrement is diminished gradually. However, M_{new} rate keeps increasing after some point

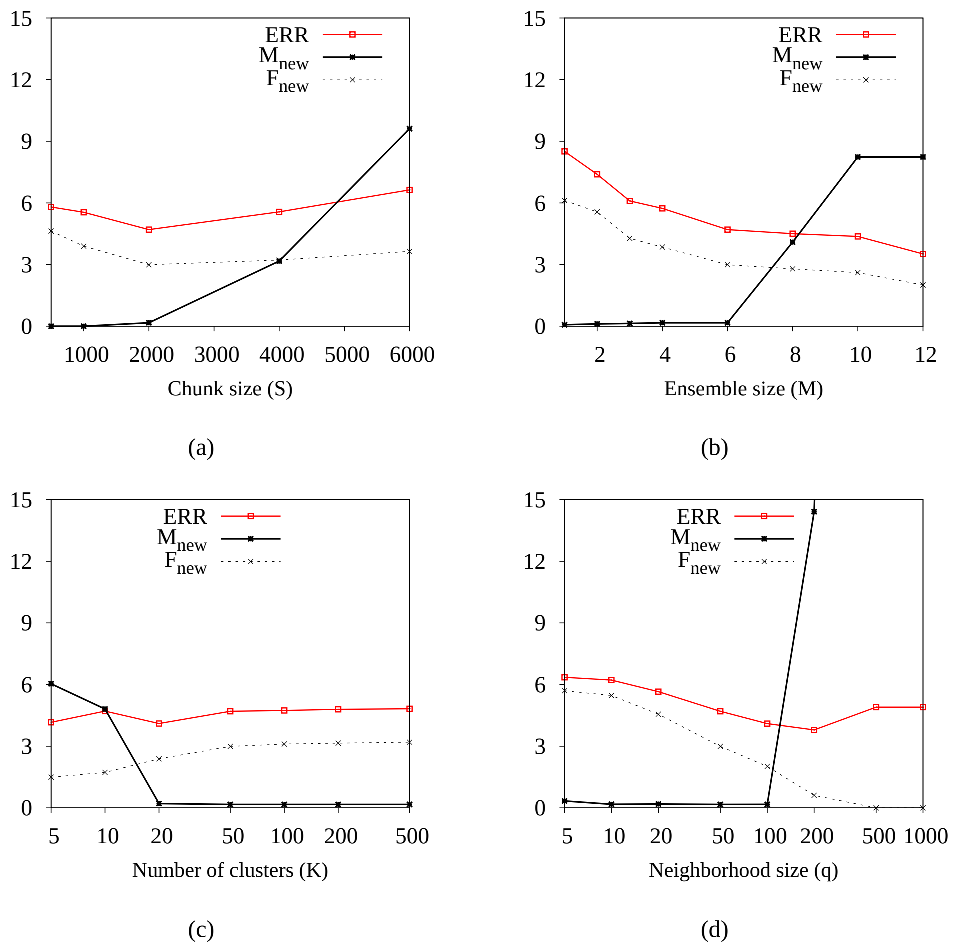


Fig. 8. Parameter sensitivity

(M=6), because a larger ensemble means more restriction on declaration of the arrival of novel classes. Therefore, we choose a value where the overall error (ERR) is considerably low and also M_{new} is low. Figure 8(c) shows the effect of number of clusters (K) on error. The x-axis in this chart is drawn in a logarithmic scale. Although the overall error is not much sensitive on K, M_{new} rate is. Increasing K reduces M_{new} rate, because outliers are more correctly detected. Figure 8(d) shows the effect of q (Minimum neighborhood size to declare a novel class) on error rates. The x-axis in this chart is also drawn in a logarithmic scale. Naturally, increasing q up to a certain point (e.g. 200) helps reducing F_{new} and ERR, since a higher value of q gives us a greater confidence (i.e., reduces possibility of false detection) in declaring a new class (see section IV). But a too large value of q increases M_{new} and ERR rates (which is observed in the chart), since a new class is missed by the algorithm if it has less than q instances in a data

TABLE II

RUNNING TIME COMPARISON IN ALL DATASETS

Dataset	Time(sec)/1K			Points/sec			Speed gain	
	XM	W-OP	W-OS	XM	W-OP	W-OS	XM over W-OP	XM over W-OS
SynC	0.33	0.41	**0.2**	2,960	2,427	**5,062**	**1.2**	0.6
SynCN	**1.7**	14.2	2.3	**605**	71	426	**8.5**	**1.4**
KDD	1.1	30.6	**0.5**	888	33	**1,964**	**26.9**	0.45
Forest Cover	0.93	8.3	**0.36**	1,068	120	**2,792**	**8.9**	0.4

chunk. We have found that any value between 20 to 100 is the best choice for q.

Finally, we compare the running times of all three competing methods on each dataset for decision tree in table II. K-NN also shows similar performances. The columns headed by “Time (sec)/1K ” show the average running times (train and test) in seconds per 1000 points, the columns headed by “Points/sec” show how many points have been processed (train and test) per second on average, and the columns headed by “speed gain” shows the ratio of the speed of XM to that of W-OP, and W-OS, respectively. For example, XM is 26.9 times faster than W-OP on KDD dataset. Also, XM is 1.2, 8.5, and 8.9 times faster than W-OP in SynC, SynCN, and Forest cover datasets, respectively. In general, W-OP is roughly C times slower than XM in a dataset having C classes. This is because W-OP needs to maintain C parallel models, one for each class. Besides, $OLINDDA$ model creates cluster using the “unknown memory” every time a new instance is identified as unknown, and tries to validate the clusters. As a result, the processing speed becomes diminished when novel classes occur frequently, as observed in KDD dataset. However, W-OS seems to run a bit faster than XM in three datasets, although W-OS shows much poorer performance in detecting novel classes and in overall error rates (see table I). For example, W-OS fails to detect 70% or more novel class instances in all datasets, but XM correctly detects 91% or more novel class instances in any dataset. Therefore, W-OS is virtually incomparable to XM for the novel class detection task. XM also outperforms W-OP both in speed and accuracy.

We also test the scalability of XM on higher dimensional data having larger number of classes. Figure 9 shows the results. The tests are done on synthetically generated data, having different dimensions (20-60) and number of classes (10-40). Each dataset has 250,000 instances. It is evident from the results that the time complexity of XM increases linearly with total number of dimensions in the data, as well as total number of classes in the data. Therefore, XM is scalable to high dimensional data.

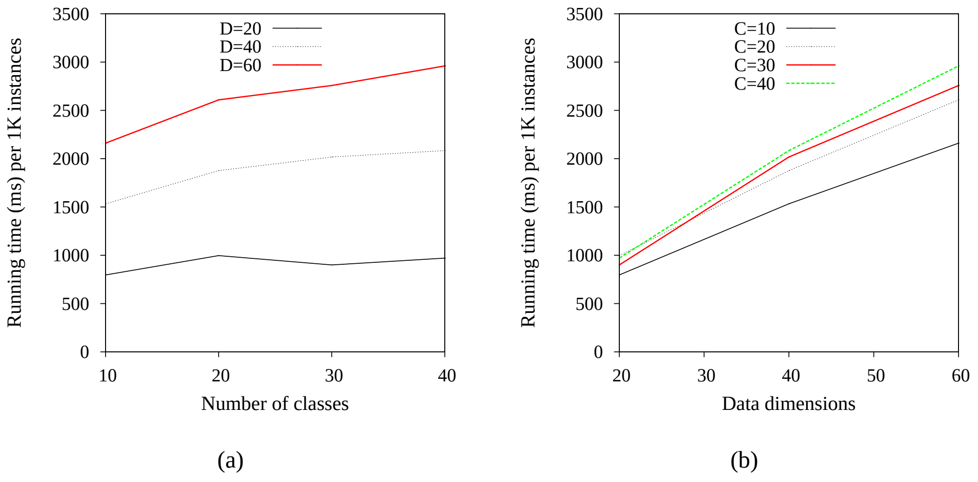


Fig. 9. Scalability test

VI. Conclusion

We have addressed several real world problems related to data stream classification. We have proposed a solution to the "novel class evolution" problem, which has been ignored by most of the existing data stream classification techniques. Existing data stream classification techniques assume that total number of classes in the stream is fixed. Therefore, when a novel class appears in the stream, instances belonging to those classes are misclassified by the existing techniques. We show how to detect novel classes automatically even when the classification model is not trained with the novel class instances. Novel class detection becomes more challenging in the presence of concept-drift. Existing novel class detection techniques have limited applicability, since those are similar to one-class classifiers. That is, they assume that there is only one "normal" class, and all other classes are novel. However, our technique is applicable to the more realistic scenario where there are more than one existing classes in the stream. Besides, our novel class detection technique is non-parametric, and it does require any specific data distribution, or does not require the classes to have spherical shape. We have also shown how to effectively classify stream data under different time constraints. Our approach outperforms the state-of-the art data stream based classification techniques in both classification accuracy and processing speed. We believe that our proposed technique will inspire more research toward solving real-world stream classification problems.

In future we would like to apply our technique to network traffic. Besides, we would like to address the data stream classification problem under dynamic feature sets.

REFERENCES

[1] D. Agarwal. An empirical bayes approach to detect anomalies in dynamic multidimensional arrays. In *Proc. ICDM*, 2005.

[2] S. D. Bay and M. Schwabacher. Mining distance-based outliers in near linear time with randomization and a simple pruning rule. In *Proc. KDD*, pages 29–38, 2003.

[3] M. M. Breunig, H.-P. Kriegel, R. T. Ng, and J. Sander. Lof: identifying density-based local outliers. In *Proc. ACM SIGMOD*, pages 93–104, 2000.

[4] S. Chen, H. Wang, S. Zhou, and P. Yu. Stop chasing trends: Discovering high order models in evolving data. In *Proc. ICDE*, pages 923–932, 2008.

[5] T. M. Cover and P. E. Hart. Nearest neighbor pattern classification. *IEEE Transactions on Information Theory*, 13(1):21–27, January 1967.

[6] V. Crupi, E. Guglielmino, and G. Milazzo. Neural-network-based system for novel fault detection in rotating machinery. *Journal of Vibration and Control*, 10(8):1137–1150, 2004.

[7] W. Fan. Systematic data selection to mine concept-drifting data streams. In *Proc. KDD*, pages 128–137, 2004.

[8] G. Hulten, L. Spencer, and P. Domingos. Mining time-changing data streams. In *Proc. SIGKDD*, 2001.

[9] L. Khan, M. Awad, and B. M. Thuraisingham. A new intrusion detection system using support vector machines and hierarchical clustering. *VLDB J.*, 16(4):507–521, 2007.

[10] A. Lazarevic and V. Kumar. Feature bagging for outlier detection. In *Proc. KDD*, pages 157–166, 2005.

[11] M. Markou and S. Singh. Novelty detection: A review-part 1: Statistical approaches, part 2: Neural network based approaches. *Signal Processing*, 83, 2003.

[12] M. M. Masud, J. Gao, L. Khan, J. Han, and B. Thuraisingham. A practical approach to classify evolving data streams: Training with limited amount of labeled data. In *Proc. ICDM*, pages 929–934, 2008.

[13] M. M. Masud, J. Gao, L. Khan, J. Han, and B. Thuraisingham. Integrating novel class detection with classification for concept-drifting data streams. In *Proc. ECML/PKDD*, 2009, to appear.

[14] A. Nairac, T. Corbett-Clark, R. Ripley, N. Townsend, and L. Tarassenko. Choosing an appropriate model for novelty detection. In *Proc. International Conference on Artificial Neural Networks*, pages 117–122, 1997.

[15] B. Pang and L. Lee. A sentimental education: Sentiment analysis using subjectivity summarization based on minimum cuts. In *Proc. of the Association for Computational Linguistics*, pages 271–278, 2004.

[16] S. J. Roberts. Extreme value statistics for novelty detection in biomedical signal processing. In *Proc. Int. Conf. on Advances in Medical Signal and Information Processing*, pages 166–172, 2000.

[17] M. Scholz and R. Klinkenberg. An ensemble classifier for drifting concepts. In *Proc. ICML/PKDD Workshop in Knowledge Discovery in Data Streams.*, 2005.

[18] E. J. Spinosa, A. P. de Leon F. de Carvalho, and J. Gama. Cluster-based novel concept detection in data streams applied to intrusion detection in computer networks. In *Proc. 2008 ACM symposium on Applied computing*, pages 976–980, 2008.

[19] S. Subramaniam, T. Palpanas, D. Papadopoulos, V. Kalogeraki, and D. Gunopulos. Online outlier detection in sensor data using non-parametric models. In *Proc. VLDB*, pages 187–198, 2006.

[20] G. Tandon and P. Chan. Weighting versus pruning in rule validation for detecting network and host anomalies. In *Proc. KDD*, pages 697–706, 2007.

[21] K. Tumer and J. Ghosh. Error correlation and error reduction in ensemble classifiers. *Connection Science*, 8(304):385–403, 1996.

[22] H. Wang, W. Fan, P. Yu, and J. Han. Mining concept-drifting data streams using ensemble classifiers. In *Proc. SIGKDD*, 2003.

[23] D. yan Yeung and C. Chow. Parzen-window network intrusion detectors. In *Proc. International Conference on Pattern Recognition*, pages 385–388, 2002.

[24] Y. Yang, X. Wu, and X. Zhu. Combining proactive and reactive predictions for data streams. In *Proc. KDD*, pages 710–715, 2005.

[25] Y. Yang, J. Zhang, J. Carbonell, and C. Jin. Topic-conditioned novelty detection. In *Proc. ACM SIGKDD*, pages 688–693, 2002.

[26] X. Zhu. Semi-supervised learning literature survey. *University of Wisconsin Madison Technical report # TR 1530*, July 2008.