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

Mohammad M. Masud, Jing Gao, Latifur Khan, *Member, IEEE,* Jiawei Han, *Fellow, IEEE,* and Bhavani Thuraisingham, *Fellow, IEEE*

Mohammad M. Masud, Latifur Khan, and Bhavani Thuraisingham are with the University of Texas at Dallas, Richardson, TX 75080, USA. email: {mehedy,lkhan,bhavani.thuraisingham}@utdallas.edu

Jing Gao and Jiawei Han are with the University of Illinois at Urbana Champaign, Urbana, IL 61801, USA. email: jinggao3@illinois.edu, hanj@cs.uiuc.edu

Digital Object Indentifier 10.1109/TKDE.2010.61 1041-4347/10/$26.00 © 2010 IEEE

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, enabling automatic detection of novel classes 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 needs to wait for more test instances to discover similarities among those instances. A maximum allowable wait time T_c is imposed as a time constraint to classify a test instance. Furthermore, most existing stream classification approaches assume that the true label of a data point can be accessed immediately after the data point is classified. In reality, a time delay T_l is involved in obtaining the true label of a data point, since manual labeling is time consuming. We show how to make 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, concept-drift, novel class, ensemble classification, K-means clustering, k-nearest neighbor classification, silhouette coefficient

I. INTRODUCTION

Data stream classification poses many challenges, some of which have not been addressed yet. Most existing data stream classification algorithms [2], [6], [11], [13], [17], [22], [27], [29] 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 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 [12], fault detection [8], and credit card fraud detection [27]. 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 by human experts in order to understand its cause, find a remedy, and make the system more secure.

Note that in our technique we consider mining from only one stream. 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 classification models. 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 all of the classifiers discover a novel class, then arrival of a novel class is declared, potential novel class instances are separated and classified as "novel class". Thus, novel class can be automatically identified without manual intervention.

Our novel classes detection technique is different from traditional "one-classs" novelty detection techniques [16], [21], [28] that can only distinguish between the normal and anomalous data. That is, 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 novel class. Furthermore, 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 such 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 [6], [11], [29] assume that the true 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 usually become available in the next billing cycle after a customer reviews all his transactions in the last statement and 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. T_c is the maximum allowable time up to which the classification decision can be postponed. 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.

Example illustrating time constraints: 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 units earlier, and x_i be the instance that arrived T_l time units earlier. Then

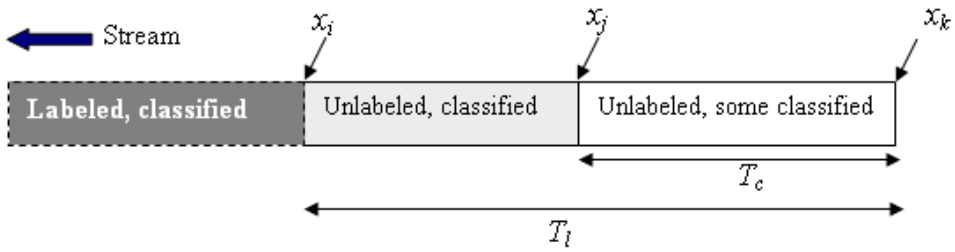


Fig. 1. Illustration of T_l and T_c

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). In summary, T_l is enforced/utilized by labeling an instance x after T_l time units of its arrival, and T_c is enforced by classifying x within T_c time units of its arrival, for every instance x in the stream.

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, it is identified as an $Foutlier$. $Foutlier$s have potential to be a novel class instance. However, we must *wait* to see whether more such $Foutlier$s appear in the stream that observe strong cohesion among themselves. If a sufficient number of such strongly cohesive $Foutlier$s are observed, a novel class is assumed to have appeared, and the $Foutlier$s are classified as a novel class instance. However, we can defer the classification decision of a test instance at most T_c time units 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 data stream classification techniques address the concept-evolution problem. This is a major problem with data streams 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 provides an overview of our approach and section IV discusses our approach in detail. Section V then describes the datasets and experimental evaluation of our technique. Finally, Section VI concludes with directions to future works.

II. Related work

Our technique is related to both data stream classification and novelty detection. Data stream classification has been an interesting research topic for years, and many approaches are available. 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 and effectively respond to concept-drift [6], [11], [29]. Several ensemble techniques for stream data mining have been proposed [9], [13], [17], [22], [27]. 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 [16]. Most novelty detection techniques fall into one of two categories: parametric, and non-parametric. Parametric approaches assume a particular 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 [19], [21]. Our technique is a non-parametric approach, and therefore, it is not restricted to any specific data distribution. There are several non-parametric approaches available, such as parzen window method [28], k-nearest neighbor (k-NN) based approach [30], kernel based method [3], and

rule based approach [15].

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 such test instances. Therefore, existing techniques discover novelty individually in each test point, whereas our technique discovers novelty collectively among several coherent 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 existing novelty detection techniques 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 "normal" model is static, i.e., there is no concept-drift in the data. But our approach can detect novel classes even if concept-drift occurs in the existing classes.

Novelty detection is also closely related to outlier detection techniques. There are many outlier detection techniques available, such as [1], [4], [5], [14]. Some of them are also applicable to data streams [24], [25]. 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. [23] 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 significantly extends our previous work on novel class detection [18] in several ways. First, 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. Second, it adds considerable amount of mathematical analysis over the previous version. Third, evaluation is done in a more realistic way (continuous evaluation rather than chunk by chunk evaluation) and a newer version of baseline technique [23] is used (version 2008 instead of 2007). Finally, more figures, discussions, and experiments are added for improved readability, clarity, and analytical enrichment.

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 $T_c < T_l$.

TABLE I

Commonly used symbols and terms

C	Total number of classes in a data stream	c_i	A class label
D_i	A data chunk	H	A cluster
h	A pseudopoint created from cluster H	K	Number of clusters created per chunk
k	Parameter k of a k-NN classifier	M	Ensemble size
L	The ensemble	L_i	The i-th model in the ensemble
q	Parameter q of the q-nearest neighborhood (q-NH)	S	Chunk size
T_l	Labeling time constraint	T_c	Classification time constraint
	Novel class A class that appears due to concept-evolution		

Table I summarizes the most commonly used symbols and terms used throughout the paper.

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: buffer buf keeps potential novel class instances, buffer U keeps unlabeled data points until they are labeled, buffer $\mathcal{L}$ 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 implemented inside algorithm 2. If the class label 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 units 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 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 [7]. Besides, this assumption is used in numerous semi-supervised learning techniques, such as [20], and in many other semi-supervised learning works [31]. We generalize this assumption by introducing the concept of

Algorithm 1 ECSMiner

```
L ← Build-initial-ensemble()
buf ← empty //temporary buffer
U ← empty //unlabeled data buffer
𝓛 ← empty //labeled data buffer (training data)
while true do
    x_j ← the latest data point in the stream
    Classify(L,x_j,buf) //(algorithm 2, section IV)
    U ⇐ x_j //enqueue
    if |U| > T_l then //time to label the oldest instance
        x_k ⇐ U //dequeue the instance
        𝓛 ⇐ < x_k, y_k > //label it and save in training buffer
        if |𝓛| = S then //training buffer is full
            L' ← Train-and-save-decision-boundary(𝓛) (section III-E)
            L ← Update(L,L',𝓛)
            𝓛 ← empty
        end if
    end if
end while
```

"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.

Let $\bar{D}_{c,q}(x)$ be the mean 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 c_{min} be the class label such that $\bar{D}_{c_{min},q}(x)$ is the minimum among all $\bar{D}_{c,q}(x)$, i.e., $\lambda_{c_{min},q}(x)$ is the nearest $\lambda_{c,q}(x)$ neighborhood (or q-nearest neighborhood or q-NH) of x. For example, in figure 2, c_{min} = c_0, i.e., $\lambda_{c_0,q}(x)$ is the q-NH of x.

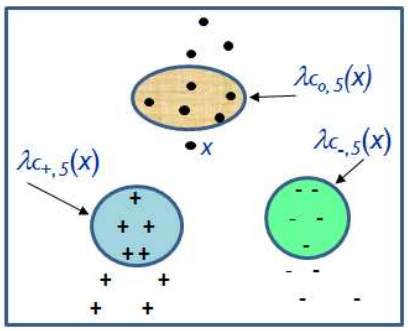


Fig. 2. Illustrating $\lambda_{c,q}(x)$ for q=5

Definition 2 (q-NH rule)): Let c_{min} be the class label of the instances in q-NH of x. According to the q-NH rule, the predicted class label of x is c_{min}.

In the example of figure 2, $c_{min} = c_0$, therefore, the predicted class label of x is 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 represent 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 follow 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: i) Saving the training data efficiently without using much memory. ii) Knowing when to classify a test instance immediately, and when to postpone the classification decision. iii) Classifying the deferred instances within T_c time unit, and iv) Predicting the presence of a novel class quickly and correctly.

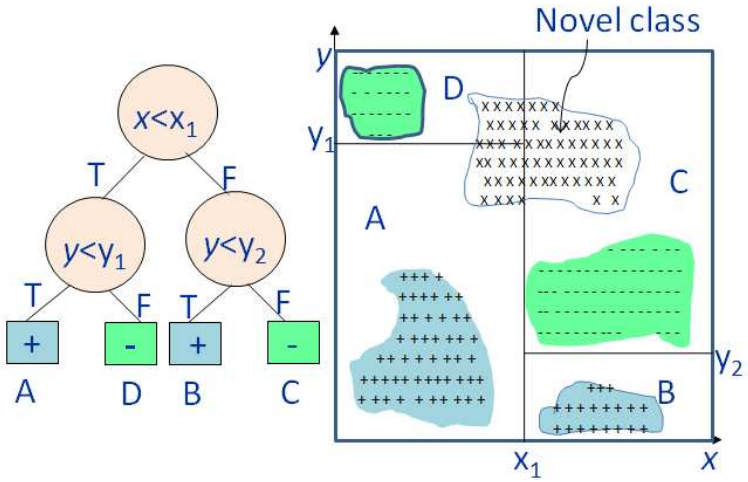


Fig. 3. A decision tree and corresponding feature space partitioning. Shaded areas represent the training data. A novel class arrives in the stream that follows property 1.

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 [17]. 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. After building the clusters, we save the cluster summary of each cluster (centroid, and frequencies of data points belonging to each class) in a data structure called "micro-cluster", and discard the raw data points. Since we store and use only K micro-clusters, 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 micro-cluster whose centroid is nearest from x_j, and assign it a class label that has the highest frequency in that micro-cluster.

E. Creating decision boundary during training

The training data are clustered using K-means and the summary of each cluster are saved as "pseudopoint". 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 k-NN, existing clusters are used that were created using the approach discussed in section III-D. 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. This is done 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 in a data structure called pseudopoint: i) *Weight*, w: Total number of points in the cluster. ii) *Centroid*, μ. iii) *Radius*, $\mathcal{R}$: Distance between the centroid and the farthest data point in the cluster. iv) *Mean distance*, μd: The mean distance from each point to the cluster centroid. So, $w(h)$ denotes the "weight" value of a pseudopoint h, and so on. After computing the cluster

summaries, the raw data are discarded and only the pseudopoints are stored in memory. Any pseudopoint having too few (less than 3) instances is considered as noise and is also discarded. Thus, the memory requirement for storing the training data becomes constant, i.e., $O(K)$.

Each pseudopoint h corresponds to a hypersphere in the feature space having center $\mu(h)$ and radius $\mathcal{R}(h)$. Let us denote the portion of feature space covered by a pseudopoint h as the "region" of h or $RE(h)$. Therefore, $RE(L_i)$ denotes the union of the regions of all pseudopoints h in the classifier L_i, i.e., $RE(L_i) = \cup_{h \in L_i} RE(h)$. $RE(L_i)$ forms a decision boundary for the training data of classifier L_i. The decision boundary for the ensemble of classifiers L is the union of the decision boundaries of all classifiers in the ensemble, i.e., $RE(L) = \cup_{L_i \in L} RE(L_i)$. The decision boundary plays an important role in novel class detection. It defines the physical boundary of existing class instances. Lemma 1 emphasizes the significance of the decision boundary in distinguishing the existing class instances from novel class instances.

Lemma 1: Let x be a test instance inside the decision boundary $RE(L)$. That is, there is a pseudopoint h such that the distance from x to the center of h is less than or equal to the radius of h, i.e., $D(x, \mu(h)) \leq \mathcal{R}(h)$. Then x must be an existing class instance.

Proof: Without loss of generality, let $D(a, b)$ be the square of Euclidean distance between a and b, i.e., $D(a, b) = (a - b)^2$. Note that $\mathcal{R}(h)$ is the distance between $\mu(h)$ and the farthest data point in the corresponding cluster H. Let the data point be x'. Therefore, $D(\mu, x') = \mathcal{R}(h)$. Also, x' is an existing class data point, since it is a training instance that was used to form the cluster. Let $x_i \in H$ be an arbitrary data point in cluster H, and the total number of data points in H is n, i.e., $w(h) = n$. In order to simplify notation, we use μ instead of $\mu(h)$ in the proof.

From the lemma statement, we can deduce that:

$$D(\mu, x) \leq D(\mu, x') \Rightarrow (x - \mu)^2 \leq (x' - \mu)^2 \Rightarrow x^2 - 2x\mu + \mu^2 \leq x'^2 - 2x'\mu + \mu^2$$

$$\Rightarrow x^2 - 2x\mu \leq x'^2 - 2x'\mu \Rightarrow x^2 - 2x\frac{1}{n}\sum_{x_i \in H} x_i \leq x'^2 - 2x'\frac{1}{n}\sum_{x_i \in H} x_i \quad \text{(by definition of } \mu)$$

$$\Rightarrow x^2 - 2x\frac{1}{n}\sum_{x_i \in H} x_i + \frac{1}{n}\sum_{x_i \in H} x_i^2 \leq x'^2 - 2x'\frac{1}{n}\sum_{x_i \in H} x_i + \frac{1}{n}\sum_{x_i \in H} x_i^2 \quad \text{(adding } \frac{1}{n}\sum_{x_i \in H} x_i^2 \text{ on both sides)}$$

$$\Rightarrow \frac{1}{n}\sum_{x_i \in H}(x^2 - 2xx_i + x_i^2) \leq \frac{1}{n}\sum_{x_i \in H}(x'^2 - 2x'x_i + x_i^2)$$

$$\Rightarrow \frac{1}{n}\sum_{x_i \in H}(x - x_i)^2 \leq \frac{1}{n}\sum_{x_i \in H}(x' - x_i)^2 \Rightarrow \bar{D}(x, H) \leq \bar{D}(x', H)$$

where $\bar{D}(x, H)$ denotes the mean distance from x to the instances in H. Therefore, the mean distance from x to the instances in H is less than the mean distance from x' to the instances in H. Since x', as well as all $x_i \in H$ are existing class instances, according to property 1, x must also be an existing class instance. ■

We deduce from the lemma that a novel class instance must be outside the decision boundary. We call any test instance outside the decision boundary as an $Foutlier$.

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

Again, a novel class instance must be an $Foutlier$.

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.

A. Classification

In line 2 of algorithm 2 we first check whether the test instance x_j is an $Foutlier$. So, if x_j is not an $Foutlier$, we classify it immediately using the ensemble voting (line 3). Recall that a novel class instance must be an $Foutlier$. However, an $Foutlier$ is not necessarily an existing class instance. Therefore, we perform further analysis on the $Foutlier$s to determine whether they really belong to novel class.

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 buf that has reached classification deadline T_c are classified immediately. An instance is removed from buf if it fulfills either of the three conditions: i) Age $>$ S: the front of buf contains the oldest element in buf. It is removed if its age is greater than S, the chunk size. Therefore, at any moment in time, there can be at most S instances in buf. ii) Ensemble update: the ensemble may be updated while an instance x_k

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 ← majority-voting(L,x_j) //classify immediately
4:     fout ← false
5: end if
6: Filter(buf)
7: if fout = true then
8:     buf ⇐ x_j //enqueue
9:     if buf.length > q and last_trial + q ≤ t_i then
10:        last_trial ← t_i
11:        novel ← DetectNovelClass(L,buf) // (algorithm 3, section IV-B)
12:        if novel = true then remove_novel (buf)
13:    end if
14: end if
```

is waiting inside buf. 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. iii) Existing class: any instance is removed from buf if it has been labeled, and it belongs to one of the existing classes. If it is not removed, it will also mislead novel class detection. 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 buf (line 8). Then we check whether buf.length, i.e., the size of buf is at least q, and the last check on buf for detecting novel class had been executed (i.e., *last_trial*) at least q time units earlier (line 9). Since novel class detection is more expensive than simple classification, this operation is performed at most once in every q time units. In line 11, algorithm 3 (DetectNovelClass) is called, which returns true if 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 $\bar{D}_{c_{out},q}(x)$ be the mean distance from an $Foutlier$ x to $\lambda_{c_{out},q}(x)$ defined by equation (1), where $\lambda_{c_{out},q}(x)$ is the set of q-nearest neighbors of x within the $Foutlier$ instances. Also, let $\bar{D}_{c_{min},q}(x)$ be the minimum among all $\bar{D}_{c,q}(x)$, where c is an existing class. Then q-NSC of x is given by:

$$q\text{-}NSC(x) = \frac{\bar{D}_{c_{min},q}(x) - \bar{D}_{c_{out},q}(x)}{max(\bar{D}_{c_{min},q}(x), \bar{D}_{c_{out},q}(x))} \tag{2}$$

q-NSC, which is a unified measure of cohesion and separation, yields a value between -1 and +1. A positive value indicates that x is closer to the $Foutlier$ instances (more cohesion) and farther away from existing class instances (more separation), and vice versa. Note that q-NSC(x) of an $Foutlier$ x must be computed separately for each classifier $L_i \in L$. We declare a *new class* if there are at least q' ($> q$) $Foutlier$s having positive q-NSC for all classifiers $L_i \in L$. The justification behind this decision is discussed in the next subsection.

Speeding up the computation of q-NSC: For each classifier $L_i \in L$, computing q-NSC for all $Foutlier$ instance takes quadratic time in the number of $Foutlier$s. Let B = buf.length. In order to compute q-NSC for one element x in buf, we need $O(B)$ time to compute the distances from x to all other elements in buf, and $O(K)$ time to compute the distances from x to all existing class pseudopoints $h \in L_i$. Therefore, the total time to compute q-NSC of all elements in buf is $O(B(B+K))$ = $O(B^2)$, since $B >> K$. In order to make the computation faster, we create K_o (= (B/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. The time required to apply K-means clustering on B instances is $O(K_oB)$. The time complexity to compute 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. Therefore, the overall complexity for computing q-NSC including the overhead for clustering becomes $O(K_o * (K_o + K) + K_oB)$ = $O(K_o(B + K_o + K)$ = $O(K_oB)$, since $B >> K \geq K_o$. So, the running time to compute q-NSC after speedup is linear in B compared to quadratic in B before speedup. q-NSC of a $Fpseudopoint$ computated in this way 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. This approximate q-NSC of an $Fpseudopoint\ h$ is denoted as $q\text{-}NSC'(h)$.

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 $q\text{-}NSC'(h)$ of every $Fpseudopoint\ h$ (line 4). If the total weight of the $Fpseudopoint$s having positive $q\text{-}NSC'()$ is greater than q, then L_i votes for novel class (line 7). If all classifiers vote for novel class, then we decide that a novel class has really appeared (line 9). 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 $q\text{-}NSC'(h)$ with respect to all classifiers $L_i \in L$ (note that $q\text{-}NSC'(h)$ 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 $q\text{-}NSC'(h)$
5: $\mathcal{H}_p \leftarrow \{h | h \in \mathcal{H}_o$ and $q\text{-}NSC'(h) > 0\ \}$ //$Fpseudopoint$s with positive $q\text{-}NSC'()$
6: $w(\mathcal{H}_p) \leftarrow \sum_{h \in \mathcal{H}_p} w(h)$. //$w(h)$ is the weight of h i.e., # of instances in the $Fpseudopoint\ h$.
7: **if** $w(\mathcal{H}_p) > q$ **then** NewClassVote++
8: **end for**
9: **if** NewClassVote $= M$ **then** return true **else** return false

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

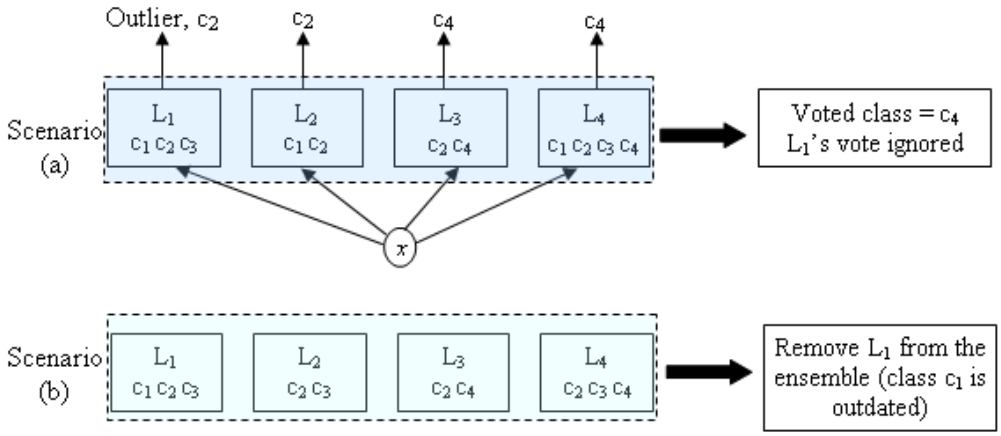


Fig. 4. Impact of evolving class label on ensemble

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 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 L_i obviously 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 had not been trained with training data of c, 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 has been 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).

Although there may be other techniques for updating the ensemble for handling evolving class labels, such as exponentially decaying weighting, we found our approach better than the others because of two reasons. First, uniform voting is preferred to weighted ensemble voting, which is supported by our initial observations, as also by other researchers (e.g. [10]). Second, by removing classifiers that contain outdated class labels, we make sure that if the outdated class re-appears, a new classification model will be included in the ensemble. This makes the ensemble more up-to-date with the current trend of that class, since the class characteristics might have been modified due to concept-drift. Note that a new model is trained in each batch anyway, (i.e., whether a novel class appears or not), therefore, there is no increase in run-time overhead due to our updating approach.

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 (2) deals with the mean distance between an $Foutlier$ and its *nearest neighborhood*. Now we go one step further to examine the mean distances between any pair of $Foutlier$s.

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

$$\bar{D}_{c_{min},q}(x) - \bar{D}_{c_{out},q}(x) > 0 \qquad \text{(from equation 2)}$$
$$\Rightarrow \bar{D}_{c_{min},q}(x) > \bar{D}_{c_{out},q}(x)$$

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

$$\begin{aligned}
&\sum_{x\in\mathcal{F}} \bar{D}_{c_{min},q}(x) > \sum_{x\in\mathcal{F}} \bar{D}_{c_{out},q}(x) \\
&\Rightarrow \sum_{x\in\mathcal{F}} \frac{1}{q} \sum_{x_i\in\lambda_{c_{min},q}(x)} D(x,x_i) > \sum_{x\in\mathcal{F}} \frac{1}{q} \sum_{x_j\in\lambda_{c_{out},q}(x)} D(x,x_j) \quad \text{(from equation 1)} \\
&\Rightarrow \frac{1}{m}\frac{1}{q} \sum_{x\in\mathcal{F}} \sum_{x_i\in\lambda_{c_{min},q}(x)} D(x,x_i) > \frac{1}{m}\frac{1}{q} \sum_{x\in\mathcal{F}} \sum_{x_j\in\lambda_{c_{out},q}(x)} D(x,x_j) \text{ (letting m = } |\mathcal{F}|)
\end{aligned} \tag{3}$$

Therefore, the mean pairwise distance between any pair (x,x_j) of $Foutlier$s, (such that x is an $Foutlier$ with positive q-NSC and x_j is an $Foutlier$ in $\lambda_{C_{out},q}(x)$), 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 its k-nearest neighbors (k-NN) within the $Foutlier$ instances (for k $\leq q$). So, each of the $Foutlier$s $x \in \mathcal{F}$ should have the same class label as the other $Foutlier$ instances, and should have a different class label than any of the existing classes. This implies that the $Foutlier$s should belong to a novel class. 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 a variation of the k-NN rule, and is applicable to any dataset irrespective of its data distribution, and shape of classes (e.g. convex and non-convex).

Deviation between approximate and exact q-NSC computation: As discussed earlier, we compute q-NSC for each $Fpseudopoint$, rather than each $Foutler$ individually in order to reduce time complexity. The resultant q-NSC is an approximation of the exact value. However, following analysis shows that the deviation of the approximate value from exact value 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 the closest existing class pseudopoint from ϕ_i

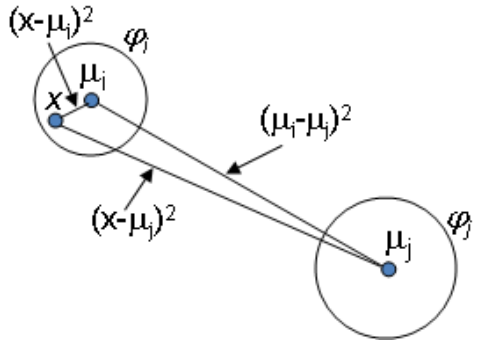


Fig. 5. Illustrating the computation of deviation. ϕ_i is an $Fpseudopoint$, i,e., a cluster of $Foutlier$s, and ϕ_j is an existing class pseudopoint, i.e., a cluster of existing class instances. In this particular example, all instances in ϕ_i belong to a novel class.

(figure 5). We compute q-$NSC'(\phi_i)$, the approximate q-NSC of ϕ_i using the following formula:

$$q\text{-}NSC'(\phi_i) = \frac{D(\mu_i, \mu_j) - \bar{D}_i}{max(D(\mu_i, \mu_j), \bar{D}_i)} \tag{4}$$

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. The exact value of q-NSC follows from equation (2):

$$q\text{-}NSC(\phi_i) = \frac{1}{q_1} \sum_{x \in \phi_i} \frac{\frac{1}{q}\sum_{x_j \in \lambda_{c_{min},q}(x)} D(x, x_j) - \frac{1}{q}\sum_{x_i \in \lambda_{c_{out},q}(x)} D(x, x_i)}{max(\frac{1}{q}\sum_{x_j \in \lambda_{c_{min},q}(x)} D(x, x_j), \frac{1}{q}\sum_{x_i \in \lambda_{c_{out},q}(x)} D(x, x_i))} \tag{5}$$

Where $\lambda_{c_{out},q}(x)$ is the set of q nearest neighbors of x within $Fpseudopoint$ ϕ_i, and $\lambda_{c_{min},q}(x)$ is the set of q nearest neighbors of x within pseudopoint ϕ_j, for some $x \in \phi_i$. Therefore, the deviation from the exact value, $\mathcal{E}_{qnsc}$ = q-$NSC(\phi_i) - q$-$NSC'(\phi_i)$. Applying equations (4) and (5),

$$\mathcal{E}_{qnsc} = \frac{1}{q_1} \sum_{x \in \phi_i} \frac{\frac{1}{q}\sum_{x_j \in \lambda_{c_{min},q}(x)} D(x, x_j) - \frac{1}{q}\sum_{x_i \in \lambda_{c_{out},q}(x)} D(x, x_i)}{max(\frac{1}{q}\sum_{x_j \in \lambda_{c_{min},q}(x)} D(x, x_j), \frac{1}{q}\sum_{x_i \in \lambda_{c_{out},q}(x)} D(x, x_i))} - \frac{D(\mu_i, \mu_j) - \bar{D}_i}{max(D(\mu_i, \mu_j), \bar{D}_i)} \tag{6}$$

In order to simplify the equations, we assume that $q_1 = q_2 = q$, and q-NSC is positive for any $x \in \phi_i$. Therefore, $\lambda_{c_{out},q}(x) = \phi_i$, $\lambda_{c_{min},q}(x) = \phi_j$. 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 can deduce following relationships:

R_1: $max(D(\mu_i, \mu_j), \bar{D}_i) = D(\mu_i, \mu_j)$ - as the q-NSC for each $x \in \phi_i$ is positive, the overall q-NSC of ϕ_i (i.e., q-$NSC'(\phi_i)$) is also positive. Therefore, this relationship follows from eq (4).

R_2: $max(\frac{1}{q}\sum_{x_j \in \lambda_{c_{min},q}(x)} D(x,x_j), \frac{1}{q}\sum_{x_i \in \lambda_{c_{out},q}(x)} D(x,x_i)) = \frac{1}{q}\sum_{x_j \in \lambda_{c_{min},q}(x)} D(x,x_j)$, which follows, since the mean q-NSC of the instances in ϕ_i is positive.

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. Therefore, q-$NSC'(\phi_i)$ can be re-written as:

$$q\text{-}NSC'(\phi_i) = \frac{(\mu_i-\mu_j)^2-\sigma^2}{(\mu_i-\mu_j)^2} = \frac{1}{q}\sum_{x\in\phi_i}\frac{(\mu_i-\mu_j)^2-(x-\mu_i)^2}{(\mu_i-\mu_j)^2} = \frac{1}{q}\sum_{x\in\phi_i} q\text{-}NSC'(x) \quad (7)$$

Where q-$NSC'(x)$ is an approximate value of q-$NSC(x)$. Now we can deduce the following inequalities:

I_1: $(x-\mu_i)^2 \leq (\mu_i-\mu_j)^2$ - since q-$NSC'(x) > 0$ for all $x \in \phi_i$.

I_2: $\sigma_i^2 \leq (\mu_i-\mu_j)^2$ - from equation 7, since q-$NSC'(\phi_i) > 0$.

I_3: $(x-\mu_j)^2 \leq (x-\mu_i)^2 + (\mu_i-\mu_j)^2$ - by triangle inequality (see figure 5)

I_4: $\sigma_j^2 \leq (\mu_i-\mu_j)^2$ - because ϕ_j represents an existing class, and similar inequality as I_2 is applicable to the instances of ϕ_j.

Continuing from equation (6):

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

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:

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

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

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

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

Applying inequalities I_1-I_4, and after several algebraic manipulations, we obtain:

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

Usually, if ϕ_i belongs to a novel class, it is empirically observed that q-$NSC'(\phi_i) \geq 0.9$. Putting this value in equation (7), and solving, we obtain $\sigma_i^2 \leq (1-0.9)(\mu_i-\mu_j)^2$. Therefore, from equation (8), we obtain $\mathcal{E}_{qnsc} \leq 0.1/3 \approx 0.03$. Since the range of q-NSC is -1 to +1, a deviation of 0.03 (3%) from the exact value is really negligible, and does not effect the outcome of the algorithm. Similar reasoning can be carried out for the cases where q-NSC of the instances in ϕ_i is negative.

Time and space complexity: Line 1 of algorithm 3 (clustering) takes $O(KS)$ time, and the for loop (lines 3-8) takes $O(K^2M)$ time. The overall time complexity of algorithm 3 is $O(K^2M + KS) = O(KS)$, since $S >> KM$. Lines 1-5 of algorithm 2 takes $O(S(KM + Mf_c))$ per chunk, where f_c is the time to classify an instance using a classifier, and $O(KM)$ is the time to determine whether an instance is an $Foutlier$. Line 6 takes $O(S)$ time. Line 11 (algorithm 3) is executed at most once in every q time units. Therefore, the worst case complexity of lines 7-14 is $O((KS) * (S/q))$, where $O(KS)$ is the time required to execute line 11 (algorithm 3). So, the overall complexity of algorithm 2 is $O(S(KM + Mf_c + KSq^{-1}))$ per chunk. For most classifiers, $f_c = O(1)$. Also, let $S/q = m$. So, the overall complexity of algorithm 2 becomes $O(KMS+MS+mS) = O(mS)$, since $m >> KM$. 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, and $m << S$.

ECSMiner keeps three buffers: buf, the training buffer $\mathcal{L}$, and the unlabeled data buffer U. Both buf and $\mathcal{L}$ hold at most S instances, whereas U 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 report 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 experiments, t is set to 0.1, 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.

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 using Gaussian distribution having different means (-5.0 to +5.0) and variances (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 times. 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, 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, 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 classifier) = 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: $OLINDDA$ [23], and Weighted Classifier Ensemble (WCE) [27], where the former works as novel class detector, and the latter performs classification. This is done as follows. For each test instance, we delay its classification for T_c time units. That is, $OLINDDA$ is given T_c time units to determine whether the instance is novel. If by that time the test instance is identified as a novel class instance, then it is considered novel and not classified using WCE. Otherwise, the instance is assumed to be an existing class instance, and its class is predicted 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 [23].

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. Therefore, 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 using all the available classes with the first *init_number* instances. 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 all three baseline techniques. Besides, the same base learner is used for WCE and ECSMiner. 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 [23] 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: Let 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. 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}) = $\frac{(F_p + F_n + F_e) * 100}{N}$. 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 *init_number* instances. In our experiments, we set *init_number* = 3S (first three 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 1,937, and 7,053 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.

Table II 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 value 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. In general, XM outperforms the baseline techniques in overall classification accuracy and novel class detection.

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

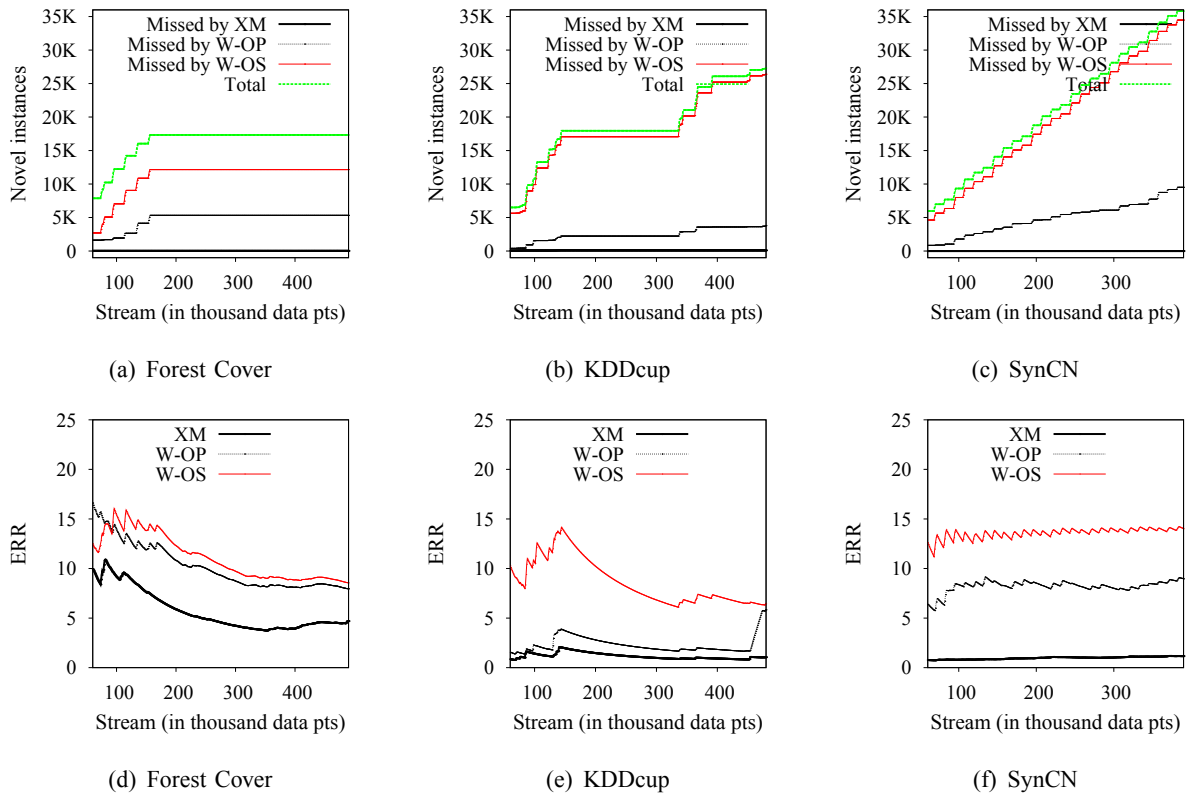


Fig. 6. Top row: novel class instances missed by each method, bottom row: overall error of each method (T_l=1000, T_c=400)

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.

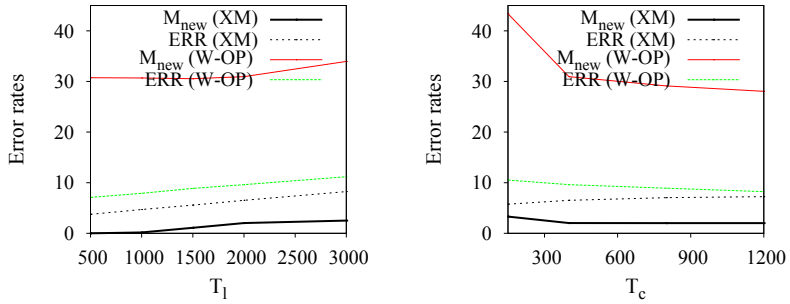
Figures (8(a)-(d)) illustrate how the error rates of XM change for different parameter settings on Forest cover dataset and decision tree classifier. These parameters have similar effects on other datasets, and k-NN classifier. Figure 8(a) shows the effect of chunk size on ERR, F_{new}, and M_{new} rates 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 [26]. But the rate of decrement is diminished gradually. However, M_{new} rate starts increasing after some point

TABLE II

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**



(a) (b)

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

(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 on 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 on 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 novel class is missed by the algorithm if there are less than q instances of the

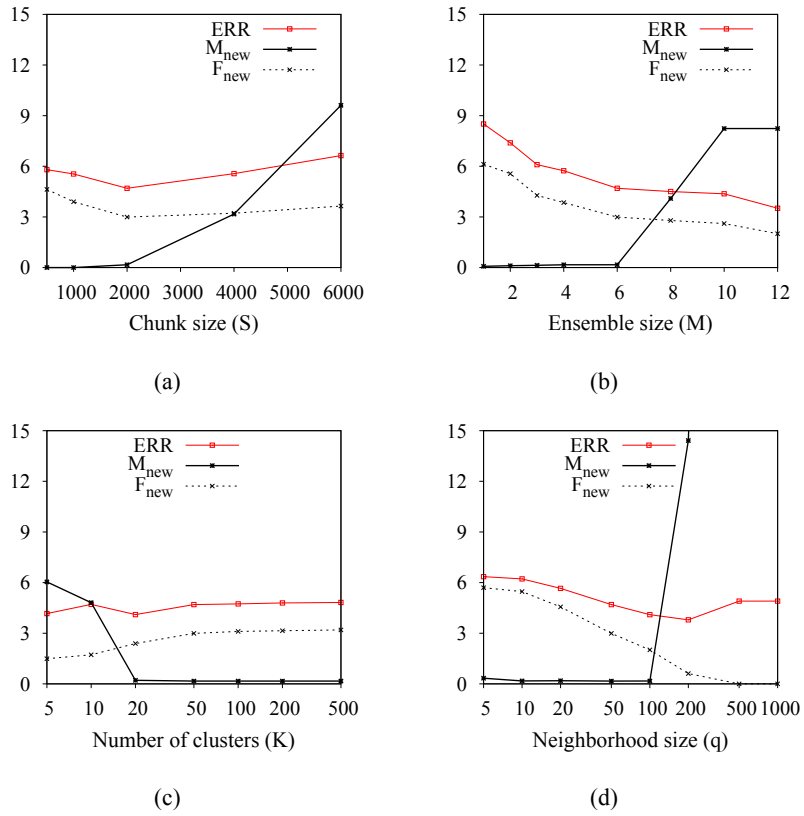


Fig. 8. Parameter sensitivity

novel class in a window of S instances. 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 III. 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

TABLE III

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

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 II). 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. Thus XM outperforms W-OP both in speed and accuracy, and dominates W-OS in 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.

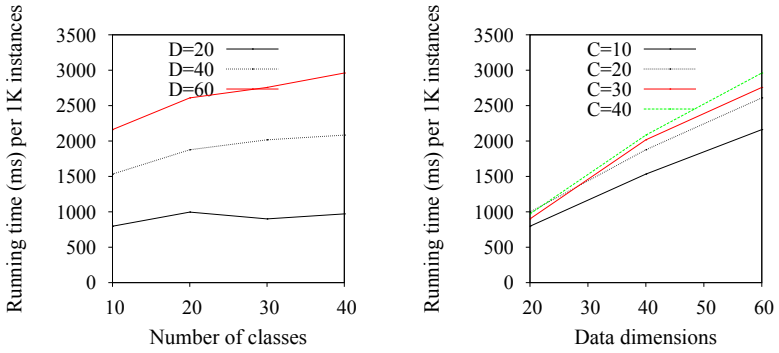


Fig. 9. Scalability test

VI. DISCUSSION AND CONCLUSION

A. Discussion

We observe in the evaluation that XM outperforms both W-OP and W-OS in detecting novel classes. The main reason behind the poorer performance of W-OP and W-OS in detecting novel

classes is the way OLINDDA detects novel class. OLINDDA makes two strong assumptions about a novel class and normal classes. First, it assumes a spherical boundary (or, convex shape) of the normal model. It updates the radius and centroid 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 must have a convex/spherical shape is too strict to be maintained for a 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 require that an existing class must have convex shape, or that the data density of a novel class should match that of the existing 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. On the other hand, XM correctly distinguishes between concept-drift and concept-evolution, avoiding false detection of novel classes in the event of concept-drift. 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.

In general, 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 convex 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 work will inspire more research toward solving real-world stream classification problems.

It might appear to readers that in order to detect novel classes we are in fact examining whether new clusters are being formed, and therefore, the detection process could go on without supervision. But supervision is necessary for classification. Without external supervision, two

separate clusters could be regarded as two different classes, although they are not. Conversely, if more than one novel classes appear simultaneously, all of them could be regarded as a single novel class if the labels of those instances are never revealed.

B. Conclusion

We have addressed several real world problems related to data stream classification. We have proposed a solution to the concept-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, instances belonging to a novel class 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.

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] C. C. Aggarwal, J. Han, J. Wang, and P. S. Yu. A framework for on-demand classification of evolving data streams. *IEEE Transactions on Knowledge and Data Engineering*, 18(5):577–589, 2006.

[3] T. Ahmed, M. Coates, and A. Lakhina. Multivariate online anomaly detection using kernel recursive least squares. In *Proc. IEEE Infocom, Anchorage, Alaska*, May 2007.

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

[5] 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.

[6] 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.

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

[8] 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.

[9] W. Fan. Systematic data selection to mine concept-drifting data streams. In *Proc. ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (SIGKDD)*, pages 128–137, Seattle, WA, USA, 2004.

[10] J. Gao, W. Fan, and J. Han. On appropriate assumptions to mine data streams. In *Proc. Seventh IEEE International Conference on Data Mining (ICDM)*, pages 143–152, Omaha, NE, USA, Oct 2007.

[11] G. Hulten, L. Spencer, and P. Domingos. Mining time-changing data streams. In *Proc. seventh ACM SIGKDD international conference on Knowledge discovery and data mining (SIGKDD)*, pages 97–106, San Francisco, CA, USA, Aug 2001.

[12] 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.

[13] J. Kolter and M. Maloof. Using additive expert ensembles to cope with concept drift. In *Proc. International Conference on Machine Learning (ICML)*, pages 449–456, Bonn, Germany, Aug 2005.

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

[15] P. Mahoney, M. V.and Chan. Learning rules for anomaly detection of hostile network traffic. In *Proc. ICDM*, 2003.

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

[17] 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. International Conference on Data Mining (ICDM)*, pages 929–934, Pisa, Italy, Dec 15-19 2008.

[18] 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, Part II, LNAI 5782*, pages 79–94, Bled, Slovenia, Sep 7-11 2009.

[19] 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.

[20] 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.

[21] 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.

[22] M. Scholz and R. Klinkenberg. An ensemble classifier for drifting concepts. In *Proc. Second International Workshop on Knowledge Discovery in Data Streams (IWKDDS)*, pages 53–64, Porto, Portugal, Oct 2005.

[23] 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.

[24] 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.

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

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

[27] H. Wang, W. Fan, P. S. Yu, and J. Han. Mining concept-drifting data streams using ensemble classifiers. In *Proc. ninth ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 226–235, Washington, DC, USA, Aug, 2003. ACM.

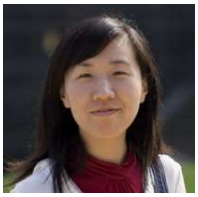
[28] D. yan Yeung and C. Chow. Parzen-window network intrusion detectors. In *Proc. International Conference on Pattern Recognition*, pages 385–388, 2002.

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

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

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

Mohammad M Masud is a Post Doctoral Fellow at the University of Texas at Dallas (UTD). He received his Ph.D. degree from UTD in December 2009. He graduated from Bangladesh University of Engineering and Technology with MS and BS in Computer Science and Engineering degree in 2004, and 2001, respectively. He was also an Assistant Professor of the same university. His research interests are in data stream mining, machine learning, and intrusion detection using data mining. His recent research focuses on developing data mining techniques to classify data streams. He has published more than 20 research papers in journals, and peer reviewed conferences including ICDM, ECML/PKDD and PAKDD. He is also an award-winning programmer at the ACM-International Collegiate Programming Contests (ICPC) World Finals-1999, held in Eindhoven, the Netherlands.



Jing Gao (ews.uiuc.edu/~jinggao3) received the BEng and MEng degrees, both in Computer Science from Harbin Institute of Technology, China, in 2002 and 2004, respectively. She is currently working toward the Ph.D. degree in the Department of Computer Science, University of Illinois at Urbana Champaign. She is broadly interested in data and information analysis with a focus on data mining and machine learning. In particular, her research interests include ensemble methods, transfer learning, mining data streams and anomaly detection. She has published more than 20 papers in refereed journals and conferences, including KDD, NIPS, ICDCS, ICDM and SDM conferences.

Latifur Khan is currently an Associate Professor in the Computer Science department at the University of Texas at Dallas (UTD), where he has taught and conducted research since September 2000. He received his Ph.D. and M.S. degrees in Computer Science from the University of Southern California, in August of 2000, and December of 1996 respectively. He obtained his B.Sc. degree in Computer Science and Engineering from Bangladesh University of Engineering and Technology, Dhaka, Bangladesh in November of 1993. His research work is supported by grants from NASA, the Air Force Office of Scientific Research (AFOSR), National Science Foundation (NSF), IARPA, the Nokia Research Center, Raytheon, Alcatel, and the SUN Academic Equipment Grant program. In addition, Dr. Khan is the director of the state-of-the-art DBL@UTD, UTD Data Mining/Database Laboratory, which is the primary center of research related to data mining, and image/video annotation at University of Texas-Dallas. Dr. Khan's research areas cover data mining, multimedia information management, semantic web and database systems with the primary focus on first three research disciplines. He has served as a committee member in numerous prestigious conferences, symposiums and workshops including the ACM SIGKDD Conference on Knowledge Discovery and Data Mining. Dr. Khan has published over 130 papers in journals and conferences.

Jiawei Han is a Professor in the Department of Computer Science at the University of Illinois. He has been working on research into data mining, data warehousing, stream data mining, spatiotemporal and multimedia data mining, information network analysis, text and Web mining, and software bug mining, with over 400 conference and journal publications. He has chaired or served in over 100 program committees of international conferences and workshops and also served or is serving on the editorial boards for Data Mining and Knowledge Discovery, IEEE Transactions on Knowledge and Data Engineering, Journal of Computer Science and Technology, and Journal of Intelligent Information Systems. He is currently the founding Editor-in-Chief of ACM Transactions on Knowledge Discovery from Data (TKDD). Jiawei has received ACM SIGKDD Innovation Award (2004), and IEEE Computer Society Technical Achievement Award (2005). He is a Fellow of ACM and IEEE. His book ”Data Mining: Concepts and Techniques” (Morgan Kaufmann) has been used worldwide as a textbook.

Bhavani Thuraisingham is a Professor of Computer Science and the Director of Cyber Security Research Center at the University of Texas at Dallas (UTD). Prior to joining UTD, she was a program director for three years at the National Science Foundation (NSF) in Arlington, VA. She has also worked for the Computer Industry in Mpls, MN for over five years and has served as an adjunct professor of computer science and member of the graduate faculty at the University of Minnesota and later taught at Boston University. Dr. Thuraisinghams research interests are in the area of Information Security and data management. She has published over 300 research papers including over 90 journals articles and is the inventor of three patents. She is also the author of nine books in data management, data mining and data security. She serves on the editorial board of numerous journals including ACM Transactions on Information and Systems Security and IEEE Transactions on Dependable and Secure Computing. She is an elected Fellow of three professional organizations: the IEEE (Institute for Electrical and Electronics Engineers), the AAAS (American Association for the Advancement of Science) and the BCS (British Computer Society) for her work in data security. She was educated in the United Kingdom both at the University of Bristol and at the University of Wales.