2012 IEEE 12th International Conference on Data Mining

Stream Classification with Recurring and Novel Class Detection using Class-Based Ensemble

Tahseen Al-Khateeb*, Mohammad M. Masud†, Latifur Khan*, Charu Aggarwal‡
Jiawei Han§ and Bhavani Thuraisingham*

*Dept. of Computer Sc., University of Texas at Dallas, USA. Email: {tahseen, lkhan, bhavani.thuraisingham}@utdallas.edu
†College of Information Technology, United Arab Emirates University, Al Ain, UAE. Email: m.masud@uaeu.ac.ae
‡IBM T. J. Watson Research Center, Yorktown Heights, New York, USA. Email: charu@us.ibm.com
§Department of Computer Science, University of Illinois at Urbana-Champaign, USA. Email: hanj@cs.uiuc.edu

***Abstract*—Concept-evolution has recently received a lot of attention in the context of mining data streams. Concept-evolution occurs when a new class evolves in the stream. Although many recent studies address this issue, most of them do not consider the scenario of recurring classes in the stream. A class is called recurring if it appears in the stream, disappears for a while, and then reappears again. Existing data stream classification techniques either misclassify the recurring class instances as another class, or falsely identify the recurring classes as novel. This increases the prediction error of the classifiers, and in some cases causes unnecessary waste in memory and computational resources. In this paper we address the recurring class issue by proposing a novel "class-based" ensemble technique, which substitutes the traditional "chunk-based" ensemble approaches and correctly distinguishes between a recurring class and a novel one. We analytically and experimentally confirm the superiority of our method over state-of-the-art techniques.**

***Keywords*-stream classification; novel class; recurring class;**

I. Introduction

The issues of one-pass learning and concept-drift are widely studied in the context of data stream classification. One pass learning is required because of the massive volume and continuous delivery of the data stream. Concept-drift occurs in dynamic streams, and many different techniques have been designed with the same goal of maintaining an up-to-date classification model [1]–[6]. Another issue of considerable recent interest is that of *concept-evolution*, which refers to the emergence of a new class. Several approaches have been proposed to address this issue [7], [8]. These approaches pro-actively detect novel classes in the stream, before being trained with the novel class instances. Most other data stream classification methods fail to detect novel classes because they assume that the number of classes in the data stream is fixed.

In real data streams, the emergence of new classes is a common phenomenon. For example, a new kind of intrusion may appear in network traffic, or a new category of text may appear in a social text stream such as *Twitter*. A special case of concept-evolution is that of a *recurring class*, which occurs when a class reappears after a long disappearance from the stream. This special case is also common because an intrusion in network traffic may reappear after a long time, or social network actors may discuss an interesting topic in *Twitter* at a particular time every year (e.g., Halloween). Another application of the *Twitter* stream is to monitor topics and detect when trends emerge. This includes general changes in topics such as sports or fashion and it includes new quickly emerging trends such as deaths or catastrophes. It is a challenging problem to correctly associate tweet messages with trends and topics. These challenges are best addressed with a streaming model due to the continuous and large volumes of incoming messages and a large number of existing and recurring classes. Most approaches for data stream classification fail to detect such recurring classes. We further elaborate this problem in the following exposition.

Data stream classifiers may either be single model incremental approaches, or ensemble techniques, in which the classification output is a function of the predictions of different classifiers. Ensemble techniques have been more popular than their single model counterparts because of their simpler implementation and higher efficiency [2]. Most of these ensemble techniques use a chunk-based approach for learning [2], [7], in which they divide the data stream into chunks, and train a model from one chunk. We refer to these approaches as "chunk-based" approaches. An ensemble of chunk-based models is used to classify unlabeled data. These approaches usually keep a fixed-sized ensemble, which is continuously updated by replacing an older model with a newly trained model. Some chunk-based techniques, such as [2], cannot detect novel classes, whereas others can do so [7]. Chunk-based techniques that cannot detect novel classes cannot detect recurrent classes as well. This is because when a class disappears for a while, the ensemble eventually discards all models trained with that class. Therefore, when the class reappears as a recurrent class, none of the models in the ensemble can detect it. On the other hand, chunk-

1550-4786/12 $26.00 © 2012 IEEE
DOI 10.1109/ICDM.2012.125

based techniques that can detect novel classes, also cannot detect recurrent classes. This is because a recurrent class is usually identified as a "novel class" by these techniques.

Recurring classes are important to detect, because they create several undesirable effects when falsely interpreted. First, they increase the false alarm rate because when they reappear, they may be either falsely detected as another class, or erroneously identified as novel. Second, when recurring classes are identified as novel, significant computational resources are wasted. This is because novel class detection is a memory and computationally intensive task. It also wastes human efforts, in cases where the output of the classification is used by a human analyst. In such cases, the analyst may have to spend extra effort in analyzing the afore-mentioned false alarms.

In this paper we propose a new ensemble technique to overcome the drawbacks of chunk-based ensembles, referred to as a *class-based* ensemble. For each class c in the stream (seen so far), we keep an ensemble of L micro-classifiers, the details of which will be explained shortly. Therefore, the total number of ensembles is C, where C is the total number of classes seen so far in the stream. The collection of these C ensembles (CL micro-classifiers in total) constitutes the complete classification model.

Next, we briefly discuss the construction and operation of the micro-classifiers. We train r micro-classifiers from a data chunk, where r is the number of classes in the chunk. Each micro-classifier is trained using only the positive instances of a class. During training (see Section IV), a decision boundary is built surrounding the training data. The newly trained micro-classifiers update the existing ensemble of models by replacing the old micro-classifiers. The function of an ensemble of micro-classifiers is two-fold. First, it checks whether a test instance falls within the decision boundary of the ensemble. A test instance x falls inside the decision boundary of an ensemble if it falls inside the decision boundary of the majority micro-classifiers in the ensemble. In this case, the ensemble is called *bounding* ensemble. Second, it outputs a *micro* classification for a test instance. The micro outputs of all the bounding ensembles are then combined to get the final classification (i.e., class prediction) of a test instance (c.f. Section V).

The contributions of this work are as follows. First, to the best of our knowledge, this is the first work that proposes a class-based ensemble technique to address both the recurring class issue and concept-evolution in data streams. Our proposed solution reduces false alarm rates and overall classification error. Second, this technique can be applied to detect periodic classes, such as classes that appear weekly, monthly, or yearly. This will be useful for a better prediction and profiling of the characteristics of a data stream. Third, we analytically show the impact of the ensemble size and concept-drift on error rates and experimentally confirm this finding. Finally, we apply our technique on a number of real and synthetic datasets, and obtain superior performance over state-of-the-art techniques.

The remainder of this paper is organized as follows. Section II discusses the related works in data stream classification and novel class detection. Section III briefly discusses the proposed approach, and Sections IV and V describes the proposed technique in details. Section VI then reports the datasets and experimental results, and Section VII concludes with directions to future work.

II. Related work

Most data stream classification techniques handle concept-drift using a number of different techniques [1]–[5], [9]–[15]. Two popular alternatives to handle the massive volume of data streams and concept-drift issues are the single-model incremental approach, and hybrid batch-incremental approach. In the single model approach, a single model is dynamically maintained with the new data. For example, [1] incrementally updates a decision tree with incoming data, and [5] incrementally updates micro-clusters in the model with the new data. The batch-incremental approach builds each classification model using a batch learning technique. However, older models are replaced by newer models when the older models become obsolete ([2]–[4], [6], [9]). Some of these hybrid approaches use a single model to classify the unlabeled data (e.g., [9]), whereas others use an ensemble of models (e.g., [2], [4]). The advantage of the hybrid approaches over the single model incremental approach is that the hybrid approaches require much simpler operations to update a model (such as removing a model from the ensemble). However, none of these techniques are capable of detecting novel classes and most of these approaches also fail to detect recurring classes, in which a class disappears from the stream and then reappears after a while.

Another branch of recently developed enhanced data stream classification techniques deals with concept-evolution, in addition to one-pass learning and and concept-drift. Spinosa et al. [16] apply a cluster-based technique to detect novel classes in data streams. Their approach builds a "normal model" of the data using clustering, defined by the hypersphere encompassing all the clusters of normal data. This model is continuously updated with stream progression. If any cluster is formed outside this hypersphere, which satisfies a certain density constraint, then a novel class is declared. However, this approach assumes only one "normal" class, and considers all other classes as "novel". Therefore, it is not directly applicable to multi-class data stream classification, since it corresponds to a "one-class" classifier. Furthermore, this technique assumes that the topological shape of the normal classes in the feature space is convex. This may not be true in real data.

Our previous work [7] proposed a classification and novel class detection technique called *ECSMiner* that is a multi-class classifier and also a novel class detector. However, this

approach does not consider the issue of recurring classes. When a class disappears from the stream for a long time and again reappears, *ECSMiner* identifies it as a novel class. In this paper we show analytically and imperially that our proposed method outperforms *ECSMiner*. In a more recent work [17] we addressed the recurring class problem by distinguishing between novel class and recurring classes. But it incurs high error (see Section VI) due to the complex structure and application of the ensembles.

In this paper, we present a more realistic and accurate approach for novel class detection, which is dramatically different from all of the above approaches. Our approach can be considered a class-based approach as opposed to the afore-mentioned chunk-based approaches. The proposed technique has the advantage of detecting novel classes more efficiently and properly distinguishing between recurring class and novel class in the presence of concept-drift. We compare our technique with the previous approaches and show the effectiveness of our technique in classification and novel class detection on benchmark data streams.

III. Overview of the approach

First we give an informal definition of the data stream classification problem. We assume that a data stream is a continuous flow of data, and that data arrive in chunks, as follows:

$$D_1 = x_1, \ldots, x_S; D_2 = x_{S+1}, \ldots, x_{2S}; \\ ; \quad \ldots; D_n = x_{(n-1)S+1}, \ldots, x_{nS}$$

where x_i is the ith instance (i.e., data point) in the stream, S is the chunk size, D_i is the ith data chunk, and D_n is the latest data chunk. Assuming that the class labels of all the instances in D_n are unknown, the problem is to predict their class labels. Let y_i and $\hat{y}_i$ be the actual and predicted class labels of x_i, respectively. If $\hat{y}_i$=y_i, then the prediction is correct; otherwise it is incorrect. The goal is to minimize the prediction error. The predicted class labels are then used for various purposes depending on the application. For example, in a credit card fraud detection problem, each transaction is an instance (data point) and is classified (i.e., class label is predicted) as either "authentic", or "fraud" by a data stream classification technique. If the predicted class is "fraud", action is taken immediately to block the transaction and notify the card holder. Often, the classification model makes mistakes in prediction, which is discovered when the card holder examines his card statement and reports incorrect predictions (such as "fraud" transactions predicted as "authentic" or vice versa). This feedback from the card holder can be considered as "labeling" of the past data. These labeled data are then used to refine the classification model.

The main concern with data stream classification is building the classification model and keeping it up-to-date by frequently updating the model with the most recent labeled data. Our proposed approach addresses this concern is highlighted in Algorithm 1 and is elaborated in the subsequent sections. But before explaining the algorithm, we define some of the terms that we use throughout the rest of the paper. Recall that we divide the stream into equal sized chunks. When the instances in a chunk are labeled by human experts, we use the chunk for training.

Definition 1 (Micro classifier ($\mathcal{M}^b$)): A Micro classifier $\mathcal{M}^b$ is a partial classifier that has been trained with only positive instances of class b and constructs a decision boundary around the training data during training.

Definition 2 (Micro classifier ensemble ($\mathcal{E}^b$)): Micro classifier ensemble $\mathcal{E}^b$ is an ensemble of L micro classifiers $\{\mathcal{M}_1^b, ..., \mathcal{M}_L^b\}$.
$\mathcal{M}^b$ checks whether a test instance is inside the decision boundary of the classifier. It is a partial classifier in the sense that it outputs only a part of the classification. The partial outputs are combined to produce a final class prediction of x (c.f. Section V-A). Section IV discusses how $\mathcal{M}^b$ is trained and updated. Section V discusses how they classify test instances and detect novel classes.

Definition 3 (Chunk based ensemble): A chunk based ensemble is an ensemble classification approach, which trains one classifier per chunk, and keeps L (constant) classifiers in the ensemble. In other words, the ensemble M contains L models $\{M_1, ..., M_L\}$, where each M_i is trained from one data chunk.

Definition 4 (Existing class): Let $\mathcal{L} = \{l_1, ..., l_C\}$ be the set of all class labels in all the labeled training chunks in the stream. A class b is an existing class if $b \in \mathcal{L}$.
In other words, b is an existing class, if at least one training chunk in the stream contained instances of class b.

Definition 5 (Novel class): Let $\mathcal{L} = \{l_1, ..., l_C\}$ be the set of all class labels in all the labeled training chunks in the stream. A class b is a novel class if $b \notin \mathcal{L}$.
In other words, a class b is novel if it never appeared before in the stream. Also, we note that b remains a novel class until the instances belonging to class b are labeled and used to train a model.

Definition 6 (Recurring class): Let $\mathcal{L} = \{l_1, ..., l_C\}$ be the set of all class labels in all the labeled training chunks in the stream. A class b is a recurring class for a chunk based ensemble M if $b \in \mathcal{L}$, but none of the models $M_i \in M$ had been trained using class b.

In other words, b is a recurring class for a chunk based ensemble if b appeared at some point in the past, but has disappeared for so long that all models trained with class b have been discarded from the ensemble, and reappeared in the latest chunk (which is still unlabeled). Therefore, for the ensemble, b appears to be a new class. In Table I we list the most frequently used symbols in this paper.

In our proposed approach, we keep an $\mathcal{E}^b$ for each class b that appeared in the Stream. So, the collection of ensembles is $\mathcal{E} = \{\mathcal{E}^1, ..., \mathcal{E}^C\}$, C being the total number of classes

Table I
FREQUENTLY USED SYMBOLS

Symbol	Meaning	Description
$\mathcal{M}^b$	Micro classifier	A partial classifier trained using positive instances of class b.
$\mathcal{E}^b$	Micro classifier ensemble	An ensemble of L Micro-classifiers $\{\mathcal{M}_1^b, ..., \mathcal{M}_L^b\}$.
$\mathcal{E}$	The combined ensemble	Set of ensembles $\{\mathcal{E}^1, ..., \mathcal{E}^C\}$ where C is the total number of classes seen so far in the stream.
L	Ensemble size	
CLAM	CLAss based Micro classifier ensemble	The proposed class based micro classifier ensemble approach

that appeared so far in the stream. When a new data chunk D_n arrives, we first classify the unlabeled data with the $\mathcal{E}$. When D_n is labeled by human experts, new $\mathcal{M}^b$

Algorithm 1 Class based ensemble (CLAM)

```
1: Build initial ensembles E={E^1, ..., E^C} with first init_num chunks
2: while stream not empty do
3:     D_n ← Latest data chunk in stream
4:     for all instance x ∈ D_n do Classify(E,x) /* Section V-A, Algorithm
       2 */
5:     //After the instances in D_n have been labeled by human experts
6:     if there is a novel class in D_n then C = C + 1
7:     for b=1 to C do
8:         s^b ← all class b instances in D_n //Partitioning
9:         if s^b ≠ null then M^b ← Train-classifier (s^b) //Section IV-A
10:        E^b ← Update-ensemble(E^b,M^b,s^b) // Section IV-B
11:    end for
12: end while
```

is trained for each class b in D_n and the existing $\mathcal{E}^b$s are updated. If a novel class c appears in chunk D_n, it will be added to the list of classes and a corresponding $\mathcal{E}^c$ will be added to the collection of ensembles $\mathcal{E}$. However, note that in our approach, classification is a continuous process, i.e., classification of incoming unlabeled data in the stream does not need to be stopped for training and updating of the classifiers. Therefore, even if the instances in D_n are not labeled, the classifier keeps classifying the subsequent chunks D_{n+1},...,D_{n+i}. However, if labeling is delayed, training and updating of the classifier is also delayed, leading to deteriorating prediction accuracy. The overall process is summarized in algorithm 1 and explained in subsequent sections. We will refer to our approach as CLAss-based Micro classifier ensemble or (CLAM).

IV. TRAINING AND ENSEMBLE CONSTRUCTION

This section discusses the *decision boundary*, and the ensemble construction and update (Figure1).

A. Training and building the decision boundary

Recall that each training chunk is first split into r disjoint partitions $\{s^1, ..., s^r\}$ of instances based on class labels, where r is the total number of classes in the chunk (algorithm 1, line 8). Therefore, partition s^i contains only the

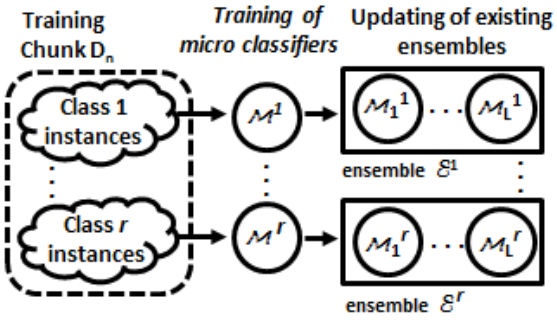

Figure 1. Training approach of micro classifier ensembles

i-th class instances, and so on (see also figure 1). Each such partition s^b is then used to train a $\mathcal{M}^b$ (algorithm 1, line 9) as follows. We use K-means clustering to build K clusters using the instances of a partition s^b. For each cluster H, we compute a summary h, containing i) μ the centroid, ii) r: the radius of H defined as the distance between the centroid and the farthest data point in H, and iii) n: the number of instances in H. The summary h is called a *micro-cluster*. After computing the micro-clusters, the raw data points are removed. The set of micro-clusters for each partition s^b constitute $\mathcal{M}^b$. Therefore, $\mathcal{M}^b$ is a set of K micro-clusters, all built from the same class instances. So, from each chunk, r number of $\mathcal{M}^b$s are built, whereas chunk-based approaches train only one classifier per chunk.

Each micro-cluster in an $\mathcal{M}^b$ corresponds to a "hyper-sphere" in the feature space with a corresponding centroid and radius, these hyper-spheres are usually small enough, and therefore the union of these hyper-spheres can represent both convex and non-convex shapes of classes. The *decision boundary* of $\mathcal{M}^b$ is the union of the feature spaces encompassed by all such hyper-spheres in that $\mathcal{M}^b$. The decision boundary of an $\mathcal{E}^b$ is the union of the decision boundaries of all $\mathcal{M}^b$s in $\mathcal{E}^b$. A test instance is considered to be inside a micro-cluster, if the distance from the test instance to the centroid of the micro-cluster is less than or equal to the radius of the micro-cluster.

The reason for using K-means clustering is two-fold. The first reason is to create the decision boundary. While there are other alternatives such as finding the convex-hull or using density-based clustering to find the clusters that encompass the training data; we choose K-means clustering because of its lower time complexity compared to those alternatives. Essentially, fast running is vital for mining data streams. The second reason for clustering is to reduce space complexity. By storing only the cluster summaries and discarding the raw data, the space complexity per $\mathcal{M}^b$ is reduced from $O(S)$ to constant (i.e., K), where S is the chunk size.

B. Ensemble construction and maintenance

When a new $\mathcal{M}^b$ of class b is trained with the latest labeled chunk, it replaces one of the $\mathcal{M}_i^b$ in $\mathcal{E}^b$ (algorithm 1, line 10). The replacement is chosen by evaluating all

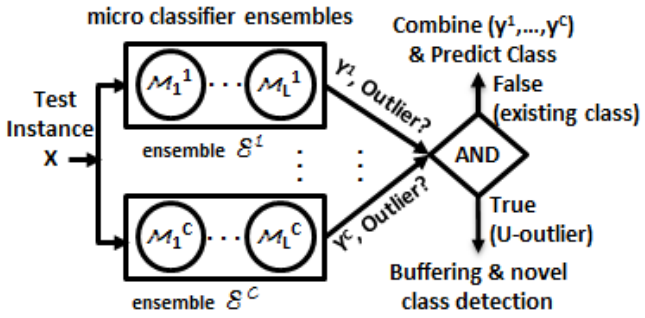

Figure 2. Ensemble classification

$\mathcal{M}_i^b \in \mathcal{E}^b$ on the class b instances of the training chunk (i.e., on partition s^b) and selecting the one with the highest prediction error. In this case, the prediction of $\mathcal{M}_i^b$ is considered erroneous, if a test instance belonging to class b falls outside the decision boundary of $\mathcal{M}_i^b$. Otherwise, the prediction is correct. This ensures that we have exactly L models in each ensemble at any given time. Therefore, only a constant amount of memory is required to store the ensemble. This addresses the limited memory problem. The concept-drift problem is addressed by keeping the ensemble up-to-date with the most recent concept.

V. Classification, Novel and Recurring Class Detection

Outlier and novel class detection are the two main stages in the classification process (Figure 2). These are detailed in this section.

A. Outlier detection and classification of existing classes

This task is summarized in algorithm 2 and explained here in detail. Each instance in the most recent unlabeled chunk is first examined by $\mathcal{E}$ to see if it is a outlier. A test instance is a outlier for $\mathcal{E}^b$ if it is outside the decision boundary of majority of the classifiers $\mathcal{M}_i^b \in \mathcal{E}^b$. Recall that a test instance is outside the decision boundary of an $\mathcal{M}_i^b$ if for each microcluster $h \in \mathcal{M}_i^b$, the test instance is outside the hypersphere defined by h, i.e., the distance from the test instance to the centroid of h is greater than the radius of h (c.f. Section IV-A). If x is found to be a outlier for *all* $\mathcal{E}^b \in \mathcal{E}$, it is called a universal outlier or U*-outlier* (see figure 2). U-outliers are saved in a buffer for further analysis (algorithm 2, line 2). This buffer is periodically checked to see if the outliers there can constitute a novel class (algorithm 2, lines 10-15).

If x is not a U-outlier, it is classified as one of the existing classes. The classification proceeds as follows. Since x is not a U-outlier, it must be inside the decision boundary of some $\mathcal{E}^b \in \mathcal{E}$. Note that if x is bounded by $\mathcal{E}^b$, it is very likely that it belongs to the class b. However, due to noise or curse of dimensionality, x may be bounded by more than one ensembles. Let $\mathcal{E}'$ be the set of $\mathcal{E}^b$s which identified x as *not* outlier (algorithm 2, line 4). For each $\mathcal{E}^b \in \mathcal{E}'$, we find the minimum distance from x to all micro-clusters $h \in \mathcal{M}^b$ in all $\mathcal{M}_i^b \in \mathcal{E}^b$. This minimum distance is saved as the component output y^b (algorithm 2, line 6, also see figure 2). For simplicity, we use the Euclidean distance metric. The component outputs are combined to get the final prediction (algorithm 2, line 8). The combine function may perform any operation (max / min etc.) based on the component values. For example, if the component outputs are probabilities, we can use the maximum among them. In our case, since the component outputs are distances, we take the minimum of those distances, and assign the predicted class to the corresponding ensemble class label. For example, if y^b is the minimum among all, then the predicted class is b.

Algorithm 2 Classify

Input: x: the latest test instance
$\mathcal{E}$=$\{\mathcal{E}^1, ..., \mathcal{E}^C\}$ the combined ensemble
buf: Buffer (FIFO) to keep potential novel class instances
Output: y, the predicted class label of x
1: **if** U-outlier($\mathcal{E}$,x) = **true then**
2: $buf \Leftarrow x$ //enqueue into buffer
3: **else**
4: $\mathcal{E}' \leftarrow \{\mathcal{E}^b | x$ inside decision boundary of $\mathcal{E}^b\}$
5: **for all** $\mathcal{E}^b \in \mathcal{E}'$ **do**
6: $y^b \leftarrow min_{j=1}^{L}(min_{k=1}^{K}\ Dist(x,\mathcal{M}_j^b.h_k))$
// h_k is the k-th micro-cluster of $\mathcal{M}_j^b$
7: **end for**
8: $y \leftarrow$ Combine($\{y^b | \mathcal{E}^b \in \mathcal{E}'\}$)
9: **end if**
/*Periodic check of buf for novel class*/
10: **if** buf.size $> q$ and Time-since-last-check $> q$ **then**
11: isNovel $\leftarrow$ **DetectNovelClass**($\mathcal{E}$,buf)
//(Section V-B, algorithm 3)
12: **if** isNovel = **true then**
13: identify and tag novel class instances
14: **end if**
15: **end if**

B. Novel and recurring class detection

The main assumption behind novel class detection is that any class of the data has the property that a data point should be closer to the data points of its own class (cohesion) and farther apart from the data points of other classes (separation).

If there is a novel class in the stream, instances belonging to the class will be far from the existing class instances and will be close to other novel class instances. Since U-outliers are outside the decision boundary of all existing classes, they are far from the existing class instances. So, the separation property for a novel class is satisfied by the U-outliers. Therefore, U-outliers are potentially novel class instances, and they are temporarily stored in a buffer buf to observe whether they also satisfy the cohesion property. The buffer is examined periodically to see whether there are enough U-outliers that are close to each other (algorithm 2, lines 10-15). This is done by computing the following metric, which we call the q-Neighborhood Silhouette Coefficient, or q-NSC. We first define the q, c-neighborhood as follows.

Definition 7 ($q, c(x)$): The q, c-neighborhood (or $q, c(x)$ in short) of an U-outliers x is the set of q class c instances that are nearest to x (q-nearest class c neighbors of x). Here q is a user defined parameter. For example, $q, c_1(x)$ of an U-outliers x is the q-nearest class c_1 neighbors of x.

Let $\bar{D}_{c_{out},q}(x)$ be the mean distance of a U-outlier x to its q-nearest U-outlier neighbors. Also, let $\bar{D}_{c,q}(x)$ be the mean distance from x to its $q, c(x)$, and let $\bar{D}_{c_{min},q}(x)$ be the minimum among all $\bar{D}_{c,q}(x)$, $c \in$\{Set of existing classes\}. In other words, q, c_{min} is the nearest existing class neighborhood of x. Then q-NSC of x is given by:

$$q\text{-}NSC(x) = \frac{\bar{D}_{c_{min},q}(x) - \bar{D}_{c_{out},q}(x)}{max(\bar{D}_{c_{min},q}(x), \bar{D}_{c_{out},q}(x))} \quad (1)$$

q-NSC is a combined measure of cohesion and separation, whose value lies within the range [-1,+1]. It is positive when x is closer to the U-outliers instances (more cohesion) and farther away from existing class instances (more separation), and vice versa.

Now we explain algorithm 3 to describe how q-NSC is computed and novel classes are detected. In order to reduce the time complexity of computing q-NSC, we first cluster the U-outliers in buf into K_0 clusters using the K-means clustering (algorithm 3, lines 1-2), where K_0 is proportional to K (number of micro-clusters per $\mathcal{M}^b$). Also, as we do for training, we save only the cluster summaries as micro-clusters, which we will call U-micro-cluster. Rather than computing q-NSC for each U-outlier, we now compute q-NSC of U-micro-clusters using a variant of equation 1. This reduces the computation time from $O(n^2)$ to $O(K_0^2)$, where n is the size of buf. For each U-micro-cluster h, we find its nearest micro-cluster h' in all $\mathcal{M}_j^b$ for (j= 1 to L, b=1 to C). The distance between h and h' is defined as the distance between their centroids. If h' belongs to class c, then c is the nearest class for h and store it as $h.nc$ (algorithm 3, line 4). The inner loop (lines 7-10) runs for every U-micro-cluster h. We take the $\mathcal{M}_j^c$ where $c = h.nc$, i.e., the j-th micro classifier of $\mathcal{E}^c$ corresponding to the nearest class of h. We then apply a variant of equation 1 to compute the q-NSC of h (line 9). Exiting from the inner loop, we compute the sum of weights (i.e., number instances) of all U-micro-clusters having positive q-NSC (line 12). If this weight is greater than q, then one vote is cast for a novel class. The outer loop (lines 6-16) runs for each index (1 to L) of each ensemble. If after exiting the loop, we find majority classifiers voted for a novel class, then a novel class is declared (line 17).

Our proposed approach does not distinguish between recurring class and existing classes. Therefore, recurring classes are normally identified as an existing class. If more than one novel classes appear at once, CLAM considers them simply as "novel", without distinguishing one from the other. However, as soon as the new chunk is labeled, the distinction becomes clear and each of the novel classes are added to the model as seperate classes.

Algorithm 3 DetectNovelClass

Input: buf: List of outliers
$\mathcal{E}$=$\{\mathcal{E}^1, ..., \mathcal{E}^C\}$ the combined ensemble
Output: **true**, if novel class is found; **false**, otherwise
1: $K_0 \leftarrow$ (K * $|buf|$/Chunk Size)
2: $\mathcal{H} \leftarrow$ Kmeans(buf, K_0)) /* clustering and saving micro-clusters */
3: **for all** $h \in \mathcal{H}$ **do**
4: h.nc $\leftarrow$ nearest-class(h) //nearest majority class
5: **end for**
6: **for** j = 1 to L **do**
7: **for all** $h \in \mathcal{H}$ **do**
8: $c \leftarrow h.nc$
9: $h.s \leftarrow q$-NSC(h,M_j^c) //Compute q-NSC
10: **end for**
11: $\mathcal{H}p \leftarrow \{h | h.s > 0\}$ //set of h with positive q-NSC
12: $w(\mathcal{H}p) \leftarrow Sum_{h \in Hp} w(h)$
//$w(h)$ = total number of instances in h
13: **if** $w(\mathcal{H}p) > q$ **then**
14: NewClassVote++
15: **end if**
16: **end for**
17: **return** NewClassVote $> L$ - NewClassVote /*Majority voting*/

C. Summary and analysis

Now we analyze the impact of the recurring classes on the classification error. In particular, we will show that if the number of recurring classes in the stream increases, the error difference between ECSMiner [7] and CLAM increases, the error of CLAM would be lower than that of ECSMiner.

Lemma 1: Let P_r be the probability of an instance in the existing class being in a recurring class, E_n be the probability of missing a novel class instance by ECSMiner, and L be the number of micro classifiers in each ensemble of CLAM. Also, let F_L be the false alarm rate of CLAM and F_0 be the false alarm rate of ECSMiner. Then the difference $(F_0 - F_L)$ increases with increasing P_r.

Proof: First, we show that F_0 alone increases with increasing P_r. Let P_n be the probability of an instance being in a novel class. Therefore, the probability of being an existing class instance is 1-P_n. Among these 1-P_n instances, P_r instances are recurring class. For ECSMiner, both recurring and novel classes are novel classes, i.e., for ECSMiner, probability of novel class is P_n+P_r. Since ECSMiner misses a novel class with probability E_n, it detects ($P_n + P_r$)*(1-E_n) instances as novel, among which (P_r)*(1-E_n) instances belong to the recurring class. Therefore, the false alarm rate of ECSMiner,

$$F_0 = \frac{(1 - E_n)P_r}{1 - P_n} \quad (2)$$

Which increases with P_r.

Recall that each $\mathcal{M}^b$ in the CLAM contains a decision boundary for the existing classes. So, it is obvious that when more models are added, the union of their decision boundaries keeps growing, which essentially filters out more recurring class instances by detecting them as *not* outliers. This reduces the false alarm rate because fewer recurring class instances are identified as novel. However, after a

certain value of L, this reduction may not be evident. Let L_m be such maximum value of L. Let $f() \in [0,1]$ be defined as a *filtering factor* of recurring class instances such that $f(0)$=0, i.e., filters none of the recurring class instances when L=0, and $f(L_i) > f(L_j)$ for $L_i < L_j \leq L_m$.

It follows from 2, that:

$$F_0 = (1 - f(0))\frac{(1-E_n)P_r}{1-P_n} \quad \text{and also}$$

$$F_L = (1 - f(L))\frac{(1-E_n)P_r}{1-P_n} = (1 - f(L))F_0; \quad L \leq L_m$$

$$\Rightarrow F_0 - F_L = F_0 - (1 - f(L))F_0$$

$$= f(L)F_0 = f(L)P_r\frac{1-E_n}{1-P_n}; \quad L \leq L_m$$

Therefore, if either L is increased or P_r is increased, or both, the difference between F_0 and F_L increases. ■

The trends implied by the afore-mentioned results are also reflected in our experiments.

Another obvious question is the impact of concept-drift on error for CLAM. Because of drift, the underlying concept of a class may change, and as a result, the $\mathcal{M}^b$s may not recognize a recurring class properly. However, the following lemma suggests that no matter how much the drift is, misclassification rate of recurring classes by CLAM is never higher than the misclassification rate of recurring classes by a similar stream classifier that use chunk based ensemble, such as ECSMiner.

Lemma 2: Let CHE be a chunk-based ensemble approach. Let d be the magnitude of drift, $E_c^M(d)$ and $E_c^B(d)$ be the misclassification rates of CLAM and CHE respectively, on a recurrence class c due to drift d. Then for any value of $d \geq 0$, $E_c^M(d) \leq E_c^B(d)$.

Proof: First, we will assess the impact of concept-drift on class characteristics in terms of changes in the feature space covered by a class. Let $\mathcal{R}_c^0$ be the region of the feature space defined by the decision boundary of class c just before the class disappeared from the stream. Also, let $\mathcal{R}_c(d)$ be the new region of space defined by the decision boundary of class c when it reappeared as a recurring class with drift d. We compute a term called *Regional Disparity* (RD) as follows: $RD_c(d) = 1 - \frac{\mathcal{R}_c^0 \cap \mathcal{R}_c(d)}{\mathcal{R}_c^0 \cup \mathcal{R}_c(d)}$ such that $\mathcal{R}_c(0) = \mathcal{R}_c^0$; $\mathcal{R}_c(d) \cap \mathcal{R}_c^0 = \phi$ for $d \geq d_{max}$; and $RD_c(d_i) \geq RD_c(d_j)$ for $d_{max} \geq d_i \geq d_j$. Therefore, $RD_c(d)$ has a value between 0 (when d=0) and 1 (when $d \geq d_{max}$). In other words, the disparity is zero (i.e., both regions are exactly the same) if the drift is zero, and disparity is maximum (regions are disjoint) when drift reaches a certain value. The disparity increases as the overlapping area (i.e., intersection) between the two regions decreases and/or the area of the union of the regions increases. Let $P_c(d)$ be the probability of correctly classifying class c instances after drift by a perfect classifier trained with class c before drift. In other words, $P_c(d)$ is the probability that after drift occurs, an instance belongs to class c if and only if it is in region $\mathcal{R}_c^0$. Also, let $f()$ be a monotonic increasing function with both its domain and range in [0,1], such that $f(0) = 0$ and $f(1) = 1$. Therefore, we can express $P_c(d)$ in terms of $RD_c(d)$ as follows: $P_c(d) = 1 - f(RD_c(d))$.

Second, we will quantify $P_c(d)$ in terms of a real classification model as opposed to a perfect model assumed for $P_c(d)$. The $\mathcal{M}^c$s approximate a decision boundary for the region $\mathcal{R}_c^0$, which defines the approximate region $\hat{\mathcal{R}}_c^0$. The corresponding approximation error by CLAM, $\hat{H}_c^M$, can be estimated by the disparity between these two regions: $\hat{H}_c^M = 1 - \frac{\hat{\mathcal{R}}_c^0 \cap \mathcal{R}_c^0}{\hat{\mathcal{R}}_c^0 \cup \mathcal{R}_c}$. Obviously, it must be the case that $\hat{H}_c^M < 1$, or otherwise the $\mathcal{M}^b$s would have 100% error. Let $Pr(c)$ be the probability of an instance being in the recurring class c after the drift. Therefore, the expected error of CLAM in classifying class c instances can be defined as follows:

$$E_c^M(d) = (1 - (1 - \hat{H}_c^M)P_c(d))Pr(c)$$

Therefore

$$E_c^M(0) = (1 - (1 - \hat{H}_c^M)P_c(0))Pr(c)$$

$$= (1 - (1 - \hat{H}_c^M)1)Pr(c) = \hat{H}_c^M Pr(c) < Pr(c) \quad (3)$$

Because $P_c(0) = 1 - f(RD_c(0)) = 1 - f(0) = 1 - 0 = 1$
Also

$$E_c^M(d' \geq d_{max}) = (1 - (1 - \hat{H}_c^M)P_c(d'))Pr(c)$$

$$= (1 - (1 - \hat{H}_c^M)0)Pr(c) = Pr(c) \quad (4)$$

Because $P_c(d' \geq d_{max}) = 1 - f(RD_c(d')) = 1 - f(1) = 1 - 1 = 0$. Similarly, the expected error of CHE is as follows:

$$E_c^B(d) = (1 - (1 - \hat{H}_c^B)P_c(d))Pr(c)$$

$$= (1 - (1 - 1)P_c(d))Pr(c) = Pr(c) \quad (5)$$

where $\hat{H}_c^B$ is the approximation error of CHE: $\hat{H}_c^B = 1 - \frac{\phi \cap \mathcal{R}_c^0}{\phi \cup \mathcal{R}_c} = 1 - 0 = 1$; because CHE does not have any model trained with class c, and therefore, the approximate region for CHE is ϕ. Following from equation 3, 4 and 5, we reach the conclusion: $E_c^M(d) \leq E_c^B(d)$ ■

Two important conclusions can be drawn from Lemma 2. First, the expected recurring class misclassification rate of CLAM increases with drift, and reaches its maximum value ($= Pr(c)$) for a certain drift magnitude. Second, a chunk-based ensemble (such as ECSMiner [7]) always observes the maximum error rate irrespective of drift magnitude. The misclassification occurs because of not being able to recognize the recurring classes, and falsely detecting them as novel class instances. However, as the lemma suggests, the expected error of CLAM never exceeds that of ECSMiner. The experimental results (Section VI-D) confirm the findings of our analysis.

VI. Experiments

In this section we describe the datasets, experimental setup, and the results.

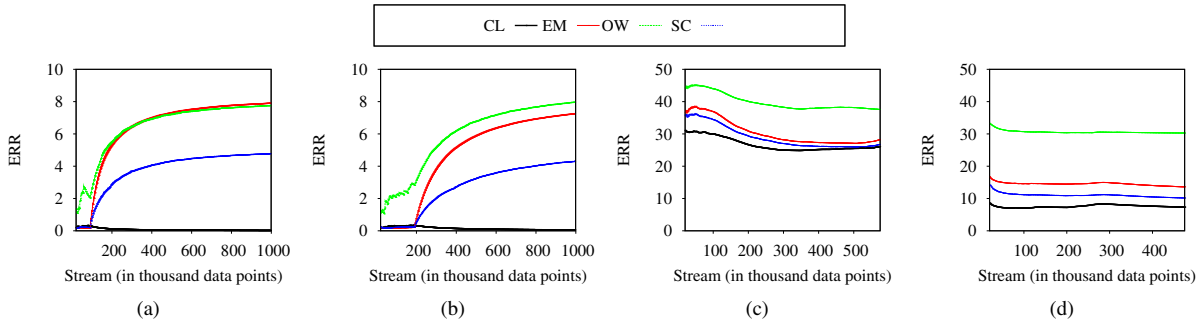

Figure 3. Error rates on (a) SynC20 (b) SynC40 (c) Forest (d) KDD

A. Data sets

Synthetic data with concept-drift and concept-evolution: This synthetic data simulates recurring and novel classes with concept-drift (details can be found in [7]). The dataset was normalized so that all attribute values were drawn from $[0, 1]$. We generated three categories of synthetic datasets with 10, 20, and 40 classes respectively. Each dataset has 40 real valued attributes, and 1 million data points. We will denote a synthetic data having X (e.g. 10) classes as SynCX (e.g. SynC10).

Real data: We used the *Forest cover type (Forest)* from the UCI repository and the *KDD cup 1999 (KDD)* intrusion detection dataset (10 percent version). Forest contains 581,000 instances, 7 classes, and 54 numeric attributes, and KDD contains 490,000 instances, 23 classes, and 34 numeric attributes. We randomly generate 10 different sequences of each dataset, and report the average results.

B. Competing approaches

CLAM (CL) this is our proposed approach. SCANR (SC) this approach proposed in [17]. ECSMiner (EM) this approach proposed in [7]. OLINDDA-WCE (OW) this is a combination of two baselines (parallel version): $OLINDDA$ [16], and Weighted Classifier Ensemble (WCE) [2]. The former works as novel class detector, and the latter performs classification.

C. Parameter settings

CL, EM, and SC: i) K (# of microclusters per classifier) = 50, ii) q (minimum # of instances required to declare novel class) = 50, iii) L (ensemble size) = 3, iv) S (chunk size) = 1,000. OLINDDA: Number of data points per cluster (N_{excl}) = 30, least number of normal instances needed to update the existing model = 100, least number of instances needed to build the initial model = 100. These parameters are chosen either according to the default values used in OLINDDA, or by trial and error to get an overall satisfactory performance. WCE. We used the same chunk size and ensemble size. The same base learners are used in all competing approaches.

D. Evaluation

Evaluation approach: We used the following performance metrics for evaluation: M_{new} = % of novel class instances misclassified as existing class, F_{new} = % of existing class instances Falsely identified as novel class, OTH = % of existing class instances misclassified as another existing class, ERR = Average misclassification error (%)(i.e., average of OTH, M_{new} and F_{new}). We build the initial models in each method with the first three chunks. Starting from chunk four, we first evaluate the performances of each method on that chunk, then use that chunk to update the existing models. The performance metrics for each chunk are saved and averaged for each method to produce the summary result.

Figures 3(a)-(d) show the ERR rates for synthetic and real data. In each of these charts, the X-axis represents stream progression starting from chunk four and the Y-axis represents the error. Recall that the first three chunks in the stream have been used to construct initial ensembles for all methods and testing starts from chunk four. For example, in figure 3(b), the Y values at X=800 represent the ERR of each method from the beginning of the stream upto the current point, (i.e., 800K instances) on the synthetic data SynC40. At this point, the ERR of CL, EM, OW and SC are 0.07%, 6.9%, 7.7%, and 4.04%, respectively. Figure 3(a) similarly plots ERR rates for SynC20 data for all methods. Figures 3(c) and 3(d) show the ERR rates for Forest, and KDD data respectively. These curves are computed by averaging ten runs of each method on ten different randomly generated sequences of each of these data sets. In all these curves, OW has the highest ERR rate, followed by EM; and CL has the lowest ERR rate.

Table II summarizes the results on all datasets for all methods. Here the result for each dataset (e.g. KDD) for a particular method (e.g. EM) is obtained by running the method (EM) on all versions (10 versions for KDD) of the dataset, and averaging the result. The ERR, M_{new} and F_{new} rates reported in the table are the error rates obtained over the entire streams. We omit reporting the OTH error because it can be derived from the other three. OW has the highest ERR in all datasets, followed by EM. Mainly M_{new} contributes

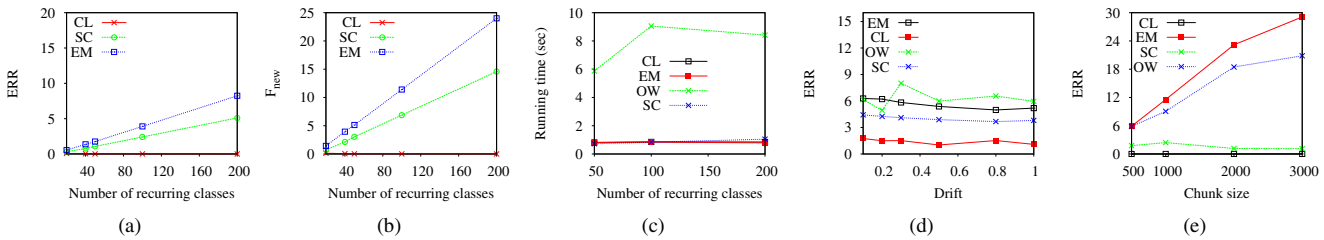

Figure 4. No. of recurring classes vs (a) ERR,(b) F_{new}, and (c) Running time. (d) Drift vs ERR (e) chunk size vs ERR

Table II
SUMMARY RESULT ON ALL DATA SETS

Metric	Competitor	SynC10	SynC20	SynC40	KDD	Forest
F_{new}	OW	0.9	1.0	1.4	**0.0**	**0.0**
	EM	24.0	23.0	20.9	5.8	16.4
	SC	14.6	13.6	11.8	3.1	12.6
	CL	**0.01**	**0.05**	**0.13**	2.3	5.0
M_{new}	OW	3.3	5.0	7.1	89.5	100
	EM	0.0	0.0	0.0	34.5	63.0
	SC	0.0	0.0	0.0	30.1	61.4
	CL	**0.0**	**0.0**	**0.0**	**17.5**	**59.5**
ERR	OW	7.5	7.7	8.0	30.3	37.6
	EM	8.2	7.9	7.2	13.7	28.0
	SC	5.1	4.8	4.3	11.5	26.7
	CL	**0.01**	**0.02**	**0.05**	**7.3**	**26.0**

to the error for OW, since it fails to detect most of the novel class instances. Therefore, the F_{new} rates of OW are also low.

The main source of higher error for EM compared to CL can be contributed to the higher F_{new} rates of EM. The main reason for the higher F_{new} rates for EM is that it cannot identify any of the recurring classes, and misclassifies them as novel. Therefore, all the recurring class instances become "false novel" for EM. SC has slightly lower F_{new} rates than EM because of its recurring class detection approach using an auxiliary ensemble. Yet SC has higher F_{new} and ERR rates than CL because SC uses the auxiliary ensemble to detect outliers but use primary ensemble to detect novel class, which is prone to more error than the novel class detection by CL ensemble. Besides, CL also exhibits lower M_{new} rates than SC because of the same reason, i.e., better detection by the CL ensemble. All other baselines also miss more of the novel class instances (higher M_{new}) than CL.

Figures 4(a) and 4(b) show the impact of varying the number of recurring classes on ERR and F_{new} rates, respectively on synthetic data. The number of recurring classes ranges from 20 to 200. In fact, the number of recurring classes is plotted on the X axes and the Y axes show the error rates for different approaches. Here we observe that both the ERR rate and F_{new} rate of EM and SC increase with increasing the number of recurring classes. This is consistent with our analysis (Section V, Lemma 1) that EM identifies the recurring classes as novel, which increases its F_{new} rate, and in turn increases ERR rate. Therefore, if there are more recurring class instances, both F_{new} rate and ERR rate of EM will increase. This is true for SC too, because of the erroneous novel class detection by the primary ensemble. Note that error rates of CL are independent of the number of recurring classes because of the superior novel class detection technique inherent in the CL-ensemble approach.

Next we analyze the effect of concept-drift on error rates. We generate SynC10 dataset with different magnitude of drift (t=0 to 1). Figure 4(d) shows the effect on the three approaches. It seems that in general, with increasing drift error rates slightly decrease (except OW). The main reason for this decrement is reduced M_{new} rate, which occurs because of the following reason. As drift increases, the older models in the ensemble become outdated more rapidly and discarded more frequently. As a result, the ensemble contains a larger number of newer models when drift is high, compared to when drift is low. Newer models in the ensemble improves the novel class detection, reducing M_{new} rate and ERR rates. However, the ERR rate of CL is always less than other baselines. This result verifies our analysis about the effect of drift (Section V-C, Lemma 2). In OW, F_{new} increases because drift causes the internal novelty detection mechanism to misclassify shifted existing class instances as novel.

Parameter sensitivity: Figure 4(e) shows how error rates change with the chunk size, with default values for other parameters. For EM, ERR increases with increasing chunk size. The reason is that F_{new} increases with increasing chunk size. This happens because when chunk size is increased, the time delay between ensemble update also increases (e.g. 500 vs 1000). Therefore, if a recurrent class appears, it will be misclassified as novel class for a longer period of time (i.e., more instances will be misclassified), which increases the F_{new} rate. For OW, on the contrary, the main contributor to ERR is the M_{new} rate. It also increases with the chunk size because of a similar reason, i.e., increased delay between ensemble update. We also varied other parameters such as primary ensemble size (L: from 1 to 8), number of microclusters per chunk (K: from 25 to 200), and the number of U-outliers with positive silhouette coefficient (NSC) needed to declare a novel class (q: from 50-200), but our approach is insensitive to these parameters within these ranges. So, we recommend using values within these ranges for these parameters. Detailed discussion about the

sensitivity of these parameters can be found in [7]

Figure 4(c) shows the running time of all approaches with varying number of recurring classes in SynC10 dataset. The running time of CL is comparable to SC and EM, and much faster than OW. For example, for 200 recurring classes, running times of CL, EM, SC, and OW are 846, 789, 1048, and 8412 milliseconds, respectively. Also, note that the running time of CL is not sensitive to the number of recurring classes in the dataset, but running time of SC increases with the number of recurring classes. This is because more time is wasted in the primary ensemble of SC when a recurring class arrives, but CL has only one ensemble that works much efficiently. OW is much slower than all other approaches because of the repeated clustering process, which is inefficient.

The space complexity of CL is $O(CL)$ compared to $O(L)$ of EM. However, this is only an insignificant increase (i.e., C) because the number of classes in the stream can be considered as near constant (increases very slowly).

VII. Conclusion

We have proposed a novel ensemble technique, which is superior to other data stream classification techniques because of its ability to detect novel class, and distinguish a recurring class from novel class. A recurring class is a class that disappears from the stream for a long time and reappears. Existing data stream classification techniques misclassifies a recurring class as another class, or identifies it as a novel class, they forget the class during its absence.

Our proposed approach can be considered a class-based ensemble, as opposed to the chunk-based ensemble that is more popular in data stream classification. The class-based ensemble creates an ensemble of models for each class and each such model is called a micro-classifier. The proposed approach CLAM has been very successful in detecting novel classes, and preventing recurring classes from being detected as a novel class, thereby increasing the classifier accuracy. We have shown both analytically and empirically that the CLAM approach outperforms the chunk-based ensemble approaches both in classification and novel class detection. In the future, we will enhance our approach to other base learners. We will also experiment with other real benchmark datasets, including text streams.

Acknowledgment

This material is based on work supported by the AFOSR under award FA9550-08-1-0260, NASA under award 2008-00867-01, NSF CNS-0931975, CCF-0905014, IIS-1017362, and ARL under Cooperative Agreement No. W911NF-09-2-0053 (NS-CTA).

References

[1] G. Hulten, L. Spencer, and P. Domingos, "Mining time-changing data streams," in *Proc. SIGKDD*, 2001, pp. 97–106.

[2] H. Wang, W. Fan, P. S. Yu, and J. Han, "Mining concept-drifting data streams using ensemble classifiers," in *Proc. KDD '03*, 2003, pp. 226–235.

[3] J. Gao, W. Fan, and J. Han., "On appropriate assumptions to mine data streams." in *Proc. ICDM*, 2007, pp. 143–152.

[4] J. Kolter and M. Maloof., "Using additive expert ensembles to cope with concept drift." in *Proc. ICML*, Aug 2005, pp. 449–456.

[5] C. C. Aggarwal, J. Han, J. Wang, and P. S. Yu, "A framework for on-demand classification of evolving data streams," *IEEE TKDE*, vol. 18, no. 5, pp. 577–589, 2006.

[6] A. Bifet, G. Holmes, B. Pfahringer, R. Kirkby, and R. Gavald, "New ensemble methods for evolving data streams," in *Proc. SIGKDD*, 2009, pp. 139–148.

[7] M. M. Masud, J. Gao, L. Khan, J. Han, and B. M. Thuraisingham, "Classification and novel class detection in concept-drifting data streams under time constraints," *IEEE TKDE*, vol. 23, no. 1, pp. 859–874, 2011.

[8] M. M. Masud, Q. Chen, L. Khan, C. C. Aggarwal, J. Gao, J. Han, and B. M. Thuraisingham, "Addressing concept-evolution in concept-drifting data streams," in *Proc. ICDM '10*, pp. 929–934.

[9] Y. Yang, X. Wu, and X. Zhu, "Combining proactive and reactive predictions for data streams," in *Proc. SIGKDD*, 2005, pp. 710–715.

[10] S. Hashemi, Y. Yang, Z. Mirzamomen, and M. Kangavari, "Adapted one-versus-all decision trees for data stream classification," *IEEE TKDE*, vol. 21, no. 5, pp. 624–637, 2009.

[11] P. Zhang, X. Zhu, and L. Guo, "Mining data streams with labeled and unlabeled training examples," in *Proc. ICDM '09.*, 2009, pp. 627–636.

[12] G. Widmer and M. Kubat, "Learning in the presence of concept drift and hidden contexts," *Machine Learning*, vol. 23, no. 1, pp. 69–101, April 1996.

[13] P. Wang, H. Wang, X. Wu, W. Wang, and B. Shi, "A low-granularity classifier for data streams with concept drifts and biased class distribution," *IEEE TKDE*, vol. 19, pp. 1202–1213, 2007.

[14] P. Zhang, J. Li, P. Wang, B. J. Gao, X. Zhu, and L. Guo, "Enabling fast prediction for ensemble models on data streams," in *Proc. SIGKDD*, 2011, pp. 177–185.

[15] P. Zhang, X. Zhu, J. Tan, and L. Guo, "Classifier and cluster ensembles for mining concept drifting data streams," in *Proc. ICDM '10*, pp. 1175–1180.

[16] 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. ACM SAC*, 2008, pp. 976–980.

[17] M. M. Masud, T. M. Al-Khateeb, L. Khan, C. C. Aggarwal, J. Gao, J. Han, and B. M. Thuraisingham, "Detecting recurring and novel classes in concept-drifting data streams," in *Proc. ICDM '11*, Dec. 2011, pp. 1176–1181.