2011 11th IEEE International Conference on Data Mining

Detecting Recurring and Novel Classes in Concept-Drifting Data Streams

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

*Dept. of Comp. Science, Univ. of Texas at Dallas. Email: {mehedy, tahseen, lkhan, bhavani.thuraisingham}@utdallas.edu
†IBM T. J. Watson Research Center, Yorktown Heights, New York. Email: charu@us.ibm.com
‡Department of Computer Science, University of Illinois at Urbana-Champaign. Email: {jinggao3,hanj}@cs.uiuc.edu

***Abstract*—Concept-evolution is one of the major challenges in data stream classification, which occurs when a new class evolves in the stream. This problem remains unaddressed by most state-of-the-art techniques. A recurring class is a special case of concept-evolution. This special case takes place when a class appears in the stream, then disappears for a long time, and again appears. Existing data stream classification techniques that address the concept-evolution problem, wrongly detect the recurring classes as novel class. This creates two main problems. First, much resource is wasted in detecting a recurring class as novel class, because novel class detection is much more computationally- and memory-intensive, as compared to simply recognizing an existing class. Second, when a novel class is identified, human experts are involved in collecting and labeling the instances of that class for future modeling. If a recurrent class is reported as novel class, it will be only a waste of human effort to find out whether it is really a novel class. In this paper, we address the recurring issue, and propose a more realistic novel class detection technique, which *remembers* a class and identifies it as "not novel" when it reappears after a long disappearance. Our approach has shown significant reduction in classification error over state-of-the-art stream classification techniques on several benchmark data streams.**

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

I. Introduction

A major challenge in data stream classification, which deserves attention, but has long been overlooked, is that of *concept-evolution*. Concept-evolution refers to the emergence of a new class. Most existing data stream classifiers assume that the number of classes are fixed [1]–[5]. However, in data streams, new classes may often appear. 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. When a new class emerges, traditional data stream classifiers misclassify the instances of the new class as one of the old classes. In other words, a traditional classifier is bound to misclassify any instance belonging to a new class, because the classifier has not been trained with that class. It is important to be able to proactively detect novel classes in data streams. For example, in an intrusion detection application, it is important to detect and raise alerts for novel intrusions as early as possible, in order to allow for early remedial action and minimization of damage. This problem has been addressed in different ways in the past [6], [7].

A *recurring class* is a special and more common case of concept-evolution in data streams. It occurs when a class reappears after long disappearance from the stream. Recurring classes, when unaddressed, create several undesirable effects. First, they increase the false alarm rate because when they reappear, they may be falsely identified as novel, whereas such classes may observe normal representative behavior. Second, they also increase human effort, in cases where the output of the classification is used by human analyst. In such cases, the analyst may have to spend extra effort in analyzing the afore-mentioned false alarms. Finally, additional computational effort is wasted in running a "novel class detection" module, which is costlier than regular "classification" process.

Prior works in novel class detection, such as [6] are not designed to address the case of recurring classes. In this approach, a fixed size ensemble is used classify the data stream and detect novel classes. When a novel class appears in the stream, it is soon added to the list of classes that the ensemble represents. However, the ensemble is periodically refined with new data, and therefore, if some class disappears for a long time, that class is eventually dropped from the list. Therefore, when the class reappears, it is again detected as a novel class, although it should have been possible to use our prior experience with the class in order to provide a more accurate result. Another work in novel concept detection [7] does not distinguish between novel class and recurrence class, i.e., it considers all classes as novel other than a prespecified "normal" class. Therefore, the problem remains, i.e., once a class has appeared in the stream, how we may "remember" this class as not novel when it appears again after a long absence from the stream.

In this paper, we propose a solution to the recurring class problem in the presence of concept-drift. Our proposed approach is designed to function as a multi-class classifier for concept-drifting data streams, detect novel classes, and distinguish recurring classes from novel classes. We keep an ensemble of size L, and also keep an auxiliary ensemble where at most L^A models per class are stored. This auxiliary

1550-4786/11 $26.00 © 2011 IEEE
DOI 10.1109/ICDM.2011.49

ensemble stores the classes in the form of classification models even after they disappear from the stream. Therefore, when a recurring class appears, it is detected by the auxiliary ensemble as *recurrent*. This approach greatly reduces false alarm rate as well as the overall error. If, however, a completely new class appears in the stream, it is detected as *novel* by the auxiliary ensemble as well.

The contributions of this work are as follows. First, to the best of our knowledge, this is the first work that addresses the recurring class issue and concept-evolution in data streams. Our proposed solution, which uses an auxiliary ensemble for recurring class detection, 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 better predicting and profiling the characteristics of a data stream. 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 Section IV describes the proposed technique in details. Section V then reports the datasets and experimental results, and Section VI concludes with directions to future works.

II. Related work

Almost all existing classification techniques [1]–[4], [8]–[11] are designed to handle the large volume of stream and concept-drift aspects of the classification process. These methods use an incremental learning approach for addressing the issues of stream volume and concept drift. There are two typical variations of this incremental approach. The first approach is a single-model incremental approach, where a single model is dynamically maintained with the new data [1], [4]. The other approach is a hybrid batch-incremental approach, in which each model is built using a batch learning technique. However, older models are replaced by newer models when the older models become obsolete ([2], [3], [5], [8]). Some of these hybrid approaches use a single model to classify the unlabeled data (e.g. [8]), whereas others use an ensemble of models (e.g. [2], [3]). 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).

The other category of data stream classification technique deals with concept-evolution, in addition to addressing infinite-length and concept-drift. Spinosa et al. [7] apply a cluster-based technique to detect novel classes in data streams. 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. Masud et al. [6] propose a classification and novel class detection technique called *ECSMiner* that is a multi-class classifier and also a novel class detector. It uses an ensemble of models to classify the unlabeled data and detect novel classes. However, this approach does not consider the issue of recurring class. ECSMiner identifies recurring classes as novel class. In this paper, we present a more realistic and accurate data stream classification approach by distinguishing recurring classes from novel classes and efficiently detecting novel classes.

III. Overview of the approach

Our proposed ensemble model consists of two ensembles, a *primary ensemble* M, and an *auxiliary ensemble* M^A. The ensemble construction and maintenance technique is explained in details in Section IV-A. In short, M is an ensemble of L classification models, $\{M_1, ..., M_L\}$, and M^A is an ensemble of L^AC *auxiliary models* $\{M^A_1, ..., M^A_{L^AC}\}$ where C is the number of classes seen in the stream so far, and L^A is the number of auxiliary models per class. The data stream is divided into equal sized chunks, and the data points in the most recent data chunk are first classified using the ensemble. When the data points in a chunk become labeled (by human experts), that chunk is used for training a classification model. Since the number of models in each ensemble is fixed, the newly trained model replaces the existing model(s) in each ensemble.

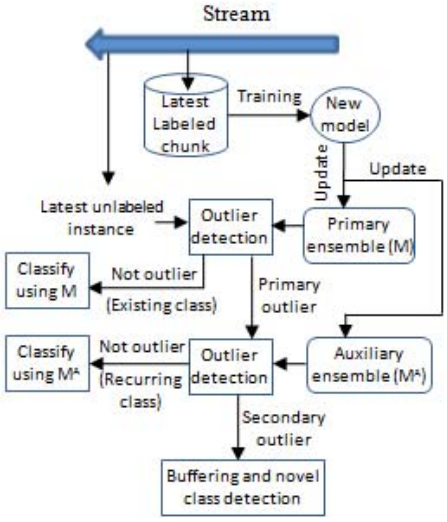

Figure 1. Architecture of the proposed approach

.

Each incoming instance in the data stream is first examined by the *outlier detection* module of the primary ensemble in order to check whether it is an outlier. If it is not an outlier, then it is classified as an existing class using majority voting among the classifiers in the primary ensemble. If it is an outlier, we call it a *primary outlier*, and it is again examined

by the outlier detection module of the auxiliary ensemble. If it is determined as not outlier by the auxiliary ensemble, it is considered as recurring class instance and classified by the auxiliary ensemble. Otherwise, if it is an outlier by the auxiliary ensemble, it is called *secondary outlier*, and it is temporarily stored in a buffer for further analysis. When there are enough instances in the buffer, the *novel class detection* module is invoked. If a novel class is found, the instances of the novel class are tagged accordingly. The classification and novel class detection technique is explained in details in Section IV-B.

Now we define some of the terms that we use throughout the rest of the paper.

Definition 1 (Current existing class): Let M be the current primary ensemble. A class c is a current existing class if any model $M_i \in M$ has been trained with class c.

Definition 2 (Universal existing class): Let M_G= $\{M_1, ..., M_i, ...\}$ be the set of all models trained in the entire stream history starting from the 1st chunk, where M_i is a model trained from the i-th data chunk. A class c is a universal existing class if at least one of the models $M_i \in M_G$ has been trained with class c.

Let C_C be the set of current existing classes and C_U be the set of universal existing classes. It is easy to show that the relationship between C_C and C_U is as follows: $C_C \subseteq C_U$.

Now we define novel and recurring classes.

Definition 3 (Novel class): A class c is a novel class if it is not a universal existing class, i.e., $c \notin C_U$.

In other words, a class c is novel if it never appeared before in the stream.

Definition 4 (Recurring class): A class c is a recurring class if it is not a current existing class, but it is a universal existing class, i.e., $c \notin C_C \wedge c \in C_U$.

In other words, c is a recurring class if c appeared sometime in the past, but now it has disappeared (i.e., it does not belong to any model in the current ensemble).

IV. Implementation details

A. Training and ensemble construction

A *decision boundary* is built during training, which decides whether an instance is outlier. Details of training and building decision boundary is explained in [6]. In short, we cluster the training data using K-means clustering, and the summary of each cluster (centroid, radius etc.) is saved as a *pseudopoint*. A pseudopoint covers a region of the feature space defined by the hypersphere corresponding to the centroid and radius of the pseudopoint. Union of all the hyperspheres constitute the decision boundary.

Each ensemble (primary and auxiliary) undergoes modification when a new model is trained from the training data. Once a new primary model is trained, it replaces one of the existing models in the primary ensemble. The candidate for replacement is chosen by evaluating each model on the latest training data, and selecting the model with the worst prediction error. This ensures that we have exactly L models in the ensemble at any given point of time. Therefore, only a constant amount of memory is required to store the ensemble. Concept-drift is addressed by keeping the ensemble up-to-date with the most recent concept.

The auxiliary ensemble consists of at most L^A models (called *auxiliary model*) per class for each class $c \in C_U$. That is, M^A contains at least one and at most L^A models for each class seen since the beginning of the stream.

When a new model is M_n trained using a data chunk, we first enumerate all the classes in the model, and for each class c, we create an *auxiliary model* $M_n.c$ that contains only class c-pseudopoints from M_n. Now in the auxiliary ensemble, we check how many class c-auxiliary models are present. If the number is less than L^A, we add $M_n.c$ to M^A. Otherwise, we choose one of the existing class c-auxiliary models for replacement and replace with $M_n.c$. We experimented different replacement strategies, such as, least recent, most recent, random, or error based, among which error based gives the best result.

B. Classification with novel and recurring class detection

There are two main stages in the classification process, namely, outlier detection, and novel class detection.

Outlier Detection: Each instance in the most recent unlabeled chunk is first examined in two stages by the ensembles to see if it is an outlier for the ensembles. A test instance is an outlier for the primary ensemble M (i.e., *primary outlier* or P-outlier, in short) if it is outside the decision boundary of each model $M_i \in M$. A test instance is outside the decision boundary of M_i if for each pseudopoint $h \in M_i$, the test instance is outside the hypershere defined by h, i.e., the distance from the test instance to the centroid of h is greater than the radius of h. Each primary outlier is again examined by the auxiliary ensemble M^A. A primary outlier x is considered a outlier for M^A (i.e., *secondary outlier* or S-outlier, in short) if x is outside the decision boundary of $|M^A|-2$ or more models in M^A, where $|M^A|$ is the number of models in M^A. If x is found to be an S-outlier, it is saved in a buffer for further analysis. This buffer is periodically checked to see if the outliers there can constitute a novel class. The main reason for the proposed order of outlier detection (first with M, then M^A) is efficiency. Since most of the instances in the stream is expected to be in the current existing class, they will be filtered out by M, and no testing shall be necessary by M^A. This saves time because M^A is usually larger than M, requiring more testing time.

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). The S-outliers are potential novel class instances, because they are far from the existing class instances (have separation). We need to find

whether the S-outliers are sufficiently close to each other (have cohesion) to form a novel class. We compute a unified measure of cohesion and separation for an S-outlier x, called q-NSC (neighborhood silhouette coefficient), as follows (See [6] for details): $q\text{-}NSC(x) = \frac{\bar{D}_{c_{min},q}(x) - \bar{D}_{c_{out},q}(x)}{max(\bar{D}_{c_{min},q}(x), \bar{D}_{c_{out},q}(x))}$

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 S-outliers instances (more cohesion) and farther away from existing class instances (more separation), and vice versa. The q-NSC(x) value of an S-outliers x is computed separately for each classifier $M_i \in M$. A *novel class* is declared if there are at least q' ($> q$) S-outliers having positive q-NSC for all classifiers $M_i \in M$. Note that if there is any recurring class instance, they should be P-outliers but not S-outliers because the primary ensemble does not contain that class, but secondary ensembles shall contain that class. In this case, those instances are classified by the auxiliary ensembles that identified them as *not* outliers. Algorithm 1 summarizes the proposed classification with novel and recurring class detection technique. We call it SCANR, which stands for Stream Classifier And Novel and Recurring class detector.

Algorithm 1 SCANR

Input: x_n: the latest test instance
M: Current ensemble of L classifiers (primary ensemble)
M^A: Auxiliary ensemble of at most $L^A|C_U|$ classifiers (C_U=set of universal existing classes)
buf: Buffer (FIFO queue) to keep potential novel class instances
Output: Class label of x_n

```
1: if isOutlier(M,x_n) = true then
2:    if isOutlier(M^A,x_n) = true then
3:       buf ⇐ x_n //enqueue into buffer
4:    else
5:       M^A' ← Models in M^A that identified x_n as not outlier
6:       ĉ ← Classify(M^A',x_n)
7:    end if
8: else
9:    ĉ ← Classify(M,x_n)
10: end if
    /*Periodic check of buf for novel class*/
11: if buf.size > q and Time-since-last-check > q then
12:    isNovel ← Check-for-novel-class(M,buf)
13:    if isNovel = true then
14:       identify and tag novel class instances
15:    end if
16: end if
```

Each incoming test instance x_n is first checked with the primary ensemble M (line 1) to see if it is an outlier for M (i.e., P-outlier). P-outliers are passed onto the auxiliary ensemble M^A for further check (line 2). If x_n is not a P-outlier, it is normally classified by M using ensemble voting (line 9). A P-outlier x_n is considered an outlier for M^A (S-outlier) if x_n is outside the decision boundary of $|M^A|-2$ or more models in M^A, where $|M^A|$ is the number of models in M^A. If x_n is found to be an S-outlier, it is saved in a buffer for further analysis (line 3). Otherwise, x_n is classified using the models $M_j^A \in M^A$ that identified x_n as *not* outlier (line 5-6). Finally, buf is periodically checked for novel classes. Novel class check is done sparingly (once in every q time) in order to reduce cost and avoid redundancy.

Time and space compexity: Classification time of SCANR is $O(Tc + P_0 L^A C_U)$, where P_0 is the probability of being a P-outlier and $O(Tc)$ is the classification time of ECSMiner. Therefore, SCANR needs only a constant additional time compared to ECSMiner (Since L^A is constant and C_U usually grows very slowly). The training times of ECSMiner and SCANR are also of the same order. The space complexity of SCANR is $O(L + L^A C_U)$.

V. Experiments

A. Data sets

Synthetic data with concept-drift and concept-evolution: Generated using the same technique as in [6]. We generate three categories of synthetic datasets with 10, 20, and 40 classes respectively. For each category (e.g. 10 classes), we generate different datasets having different number of recurring classes, ranging from 10 to 160 recurring classes per dataset. 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 have used the *Forest cover type (Forest)* from UCI repository and *KDD cup 1999 (KDD)* intrusion detection dataset (10 percent version). We randomly generate 10 different sequences of each dataset, and report the average results. Both these datasets are arranged to have novel and recurring classes.

B. Competitors

SCANR (SC): This is our proposed approach. We will use the notation SC-X (e.g. SC-5) to represent SCANR having X (e.g. 5) auxiliary models per class (L^A=5).

ECSMiner (EM): This is the existing approach [6].

OLINDDA-WCE (OW): This is the same bseline used in [6], which a combination of two baselines: $OLINDDA$ [7], and Weighted Classifier Ensemble (WCE) [2].

C. Parameter settings

EM and SC: i) K (# of pseudopoints / classifier) = 50, ii) q (minimum # of instances reqd. to declare novel class) = 50, iii) L (ensemble size) = 3, iv) S (chunk size) = 1,000.
OLINDDA: Same settings used in [6]. We use the same chunk size, ensemble size, and also the same base learner for all competing approaches.

D. Evaluation

Evaluation approach: We use the following performance metrics for evaluation: M_{new} = % of novel class instances misclassified as existing class, F_{new} = % of existing class instances misclassified as novel class, **OTH** = % of existing class instances misclassified as another existing class, **ERR** = Average misclassification error (%)(i.e., avg of OTH, M_{new} and F_{new}).

We build the initial models in each method with the first *init*(=3) chunks. From the *init* +1st chunk onward, 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 for each method are saved and averaged for producing the summary result.

Figures 2(a)-(d) show the ERR rates for synthetic and real data. The X axes in these graphs correspond to the stream in thousand data points and the Y axes represent aggregate error rates. For example, in figure 2(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 OW,EM,SC-5,SC-15, and SC-25 are 9.1%, 3.2%, 0.8%, 0.4%, and 0.37%, respectively. Figure 2(a) similarly plots ERR rates for SynC20 data for all methods. Figures 2(c) and 2(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, SC has the lowest ERR rates.

Table I summarizes the results on different datasets on 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 see that OW has the highest ERR in all datasets, followed by EM. The main source of error for OW is M_{new}, 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 SC can be contributed to the higher F_{new} rates of EM, which occurs because EM misclassifies all recurring class instances as novel ("false novel" error). Since SC can correctly identify most of the recurring class instances, the F_{new} rates are low.

Figure 3(a) show the effect of number of recurring classes on error rates. This study is done on the synthetic data and have similar effects on the real datasets. Figure 3(a) shows the ERR on synthetic data, where the number of recurring classes ranges from 10 to 150. Here we observe that the ERR rate of EM increase with increasing number of recurring classes. This is because EM identifies the recurring classes as novel. Therefore, more recurring class increases its F_{new} rate, and in turn increases ERR rate. Figure 3(b) shows how the ERR rate of SC reduces with increasing auxiliary ensemble size L^A. If we increase L^A, F_{new} rate drops, and as a consequence, ERR rates drop. This is because larger number of auxiliary models contribute to a more complete decision boundary for any existing class, leading to more precise outlier detection. We also notice that error is higher for datasets having higher number of recurring classes. For example, the curve r=31 shows the ERR rates for a dataset having 31 recurring classes, and so on. The reason is also obvious, i.e., as the number of recurring classes increase, more instances are misclassified as novel.

Table I
SUMMARY RESULT

Dataset	Competitor	F_{new}	M_{new}	ERR
SynC20	OW	4.0	4.9	4.2
	EM	12.4	0.0	4.3
	SC-5	2.4	0.0	1.0
	SC-15	0.8	0.0	0.5
	SC-25	**0.6**	**0.0**	**0.3**
SynC40	OW	1.3	24.0	11.0
	EM	10.1	0.0	3.5
	SC-5	1.9	0.0	0.9
	SC-15	0.7	0.0	0.5
	SC-25	**0.5**	**0.0**	**0.4**
KDD	OW	**0.0**	90.7	30.7
	EM	5.1	33.1	13.0
	SC-5	2.8	28.7	10.9
	SC-15	2.6	28.7	10.8
	SC-25	2.4	**28.7**	**10.7**
Forest	OW	**0.0**	100.0	37.6
	EM	16.4	**63.0**	28.0
	SC-5	5.1	64.1	25.9
	SC-15	3.7	64.4	**25.9**
	SC-25	3.3	64.4	26.0

To analyze the effect of concept-drift on error rates, we generate SynC10 dataset with different magnitudes of drift (*t*=0 to 1). Figure 3(c) shows the effect on the three approaches. For SC, the F_{new} rate increases when drift increases, resulting in increased ERR rate. The F_{new} rate (and ERR) of EM is almost independent of drift, i.e., whether drift occurs or not, it misclassifies *all* the recurrent class instances. However, the F_{new} rate of SC is always less than that of EM. F_{new} rate increases in OW because the drift causes the internal novelty detection mechanism to misclassify shifted existing class instances as novel. Figure 3(d) shows how error rates change with the chunk size, with default values for other parameters. For SC-25, the ERR rate decreases with increasing chunk size upto certain point. This is because larger training data naturally increases the quality of the learned model. However, for EM, we see that 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 L (from 3 to 8) and K (from 25 to 200) but our approach was insensitive to these parameters within these ranges.

VI. CONCLUSION

We have proposed a solution to the recurring class problem, which has been ignored by the existing novel

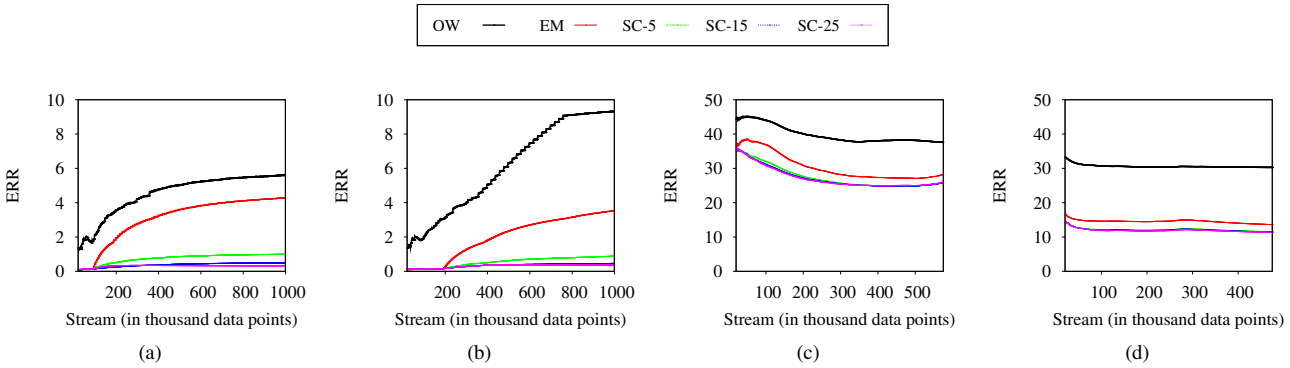

Figure 2. Error rates on (a) SynC20 (b) SynC40, (c) Forest and (d) KDD

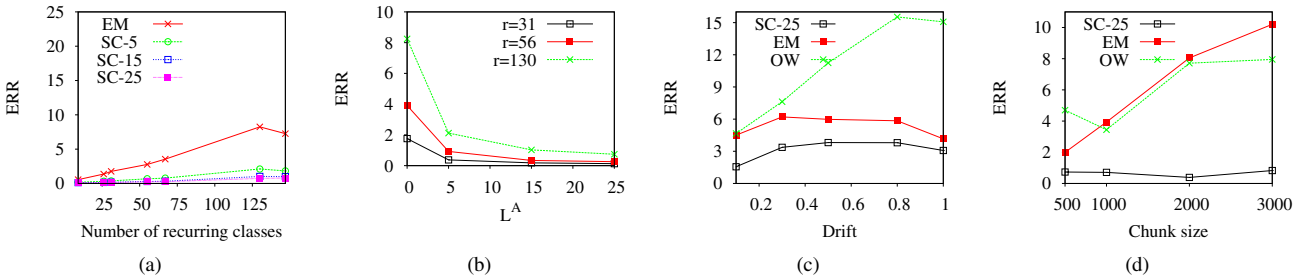

Figure 3. (a) # of recurring class vs ERR; (b) L^A vs ERR; (c) Drift vs ERR; (d) chunk size vs ERR

class detection techniques. This solution utilizes an auxiliary ensemble for keeping track of the classes that appeared in the stream so far. This reduces false alarm rate, as well as overall error rates, which has been demonstrated empirically on synthetic and benchmark data streams. Furthermore, this solution would greatly reduce the human effort that would otherwise be required to identify the recurring classes in data streams. An important application of this technique would be detecting periodic classes, for example, classes that appear every weekend or every winter. This would be useful for better predicting and profiling data streams.

In the future we would explore the possibility of optimizing the classification and outlier detection by the auxiliary ensemble to reduce extra running time. Also, we would like to apply our technique on other real world data streams such as Twitter text streams.

Acknowledgment

This material is based on work supported by the AFOSR under awards FA9450-08-1-0260 and FA9950-10-1-0088 and by NASA under award 2008-00867-01.

References

[1] G. Hulten, L. Spencer, and P. Domingos, "Mining time-changing data streams," in *Proc. KDD*, 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. Kolter and M. Maloof., "Using additive expert ensembles to cope with concept drift." in *Proc. ICML*, 2005, pp. 449–456.

[4] 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.

[5] A. Bifet, G. Holmes, B. Pfahringer, R. Kirkby, and R. Gavald, "New ensemble methods for evolving data streams," in *Proc. ACM SIGKDD '09*, 2009, pp. 139–148.

[6] 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.

[7] 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.

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

[9] 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.

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

[11] 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.