SAND2015-7513J

Adaptive Framework for Classification and Novel Class Detection over Evolving Data Streams with Limited Labeled Data

Ahsanul Haque, Latifur Khan, Michael Baron, and Joe Ingram

Most approaches to classifying evolving data streams either divide the stream of data into fixed-size chunks or use gradual forgetting to address the problems of infinite length and concept drift. Finding the fixed size of the chunks or choosing a forgetting rate without prior knowledge about time-scale of change is not a trivial task. As a result, these approaches suffer from a trade-off between performance and sensitivity. To address this problem, we present a framework which uses change detection techniques on the classifier performance to determine chunk boundaries dynamically. Though this framework exhibits good performance, it is heavily dependent on the availability of true labels of data instances. However, labeled data instances are scarce in realistic settings and not readily available. Therefore, we present a second framework which is unsupervised in nature, and exploits change detection on classifier confidence values to determine chunk boundaries dynamically. In this way, it avoids the use of labeled data while still addressing the problems of infinite length and concept drift. Moreover, both of our proposed frameworks address the concept evolution problem by detecting outliers having similar values for the attributes. We provide theoretical proof that our change detection method works better than other state-of-the-art approaches in this particular scenario. Results from experiments on various benchmark and synthetic data sets also show the efficiency of our proposed frameworks.

1 Introduction

Data stream classification is a challenging task due to its own inherent properties, e.g., infinite length, concept drift, concept evolution, limited labeled data, delayed labeling, etc. [6], [12], [20]. Due to the infinite length, traditional multi-pass data mining techniques are not applicable on stream data as they would require infinite storage. Moreover, class boundaries keep changing due to concept drift. So, the classification model needs to be updated regularly, to make sure that it reflects the most recent concept.

Typically, existing data stream classification algorithms [1], [19], [22], [25] address infinite length and concept drift problems by dividing the stream of data into fixed-size chunks, and updating the classifier once all the instances in the chunk are labeled. These approaches require *a priori* knowledge about the time-scale of change to find the fixed size, which is rarely feasible in the case of data streams [5]. As a result, these techniques suffer from a trade-off between performance during stable periods, and sensitivity to sudden concept drift. Some other available approaches [15], [16], [17] use *gradual forgetting* to address infinite length and concept drift problems. These approaches use various decay functions to assign weight to data instances based on their age. This strategy also suffers from a similar trade-off while choosing the decay rate to match an unknown rate of change.

In this paper, we present two complete frameworks to address the aforementioned problems. These frameworks address the infinite length problem by dividing the data stream into variable-size chunks. It uses a change point detection technique to keep track of any change in the classifier feedback and to determine the chunk size dynamically. So, it can adapt to the changing concept immediately. To address the concept drift problem, our frameworks maintain an ensemble of classifier models, each trained on different dynamically determined chunks.

The first framework presented in this paper uses classifier predictive performance as feedback from the classifier. These predictive performance scores are monitored using a dynamic sliding window. A change detection method is exploited to detect significant changes in the classifier predictive performance. If a change is detected, the chunk boundary is determined dynamically and the existing classifier is updated. This framework requires true labels of the data instances to evaluate predictive performance. However, labeled data instances are scarce and not readily available in real-world data streams [20]. So, a good classifier in the streaming context should be able to use partially labeled training data, i.e., only a portion of the training data is labeled.

The second framework proposed in this paper does not

depend on the classifier predictive performance. Rather, it uses a few heuristics to calculate the classifier confidence in predictions and applies a change detection algorithm on these confidence scores. Hence, this framework does not require labels for all data instances during training. Intuitively, heuristic-based confidence scores follow a skewed $beta$ distribution which is also confirmed by *chi-square* goodness-of-fit tests. We design a suitable change detection algorithm to detect significant changes in the classifier confidence scores. Theoretically, we show that this change detection algorithm is more efficient and applicable to this specific scenario than existing algorithms like *adaptive windowing* (ADWIN) [5].

Compared to concept drift, relatively few approaches in the literature focus on addressing the concept evolution problem in data streams. Concept evolution occurs when a new class emerges in the data stream [22], [26]. In real-world data stream classification problems, such as intrusion detection, text classification and fault detection, concept evolution may occur at any time. As a result, if instances from a novel class appear in the data stream, most approaches which do not handle concept evolution will classify those instances as one of the existing classes. This affects the performance of the classifier by increasing classification error.

To address the concept drift problem, our frameworks maintain an ensemble of t classifier models, each trained on different dynamically-determined chunks. Each of the models in the classifier is based on a semi-supervised K-means algorithm. So, both classification and the confidence value calculation in our framework can work with delayed labeling and partially labeled training data. To address the concept evolution problem, our framework includes a novel class detector. If any instance falls outside of the decision boundaries of all the models in the ensemble, it detects it as an outlier. If there are a sufficiently large number of outliers which share similar values for the attributes among themselves, our framework declares those as instances from a novel class. To the best of our knowledge, our framework is the first semi-supervised approach which uses heuristic-based dynamic sliding window management to address both concept drift and concept evolution.

Primary contributions of our work are as follows: 1) we present two frameworks to address the challenges of data stream classification, based on dynamic sliding window management. 2) To avoid the use of labeled data, we present a heuristic-based technique to estimate classifier confidence in predicting labels of data instances. We theoretically justify the choice of the proposed heuristics. 3) We design suitable change detection algorithms for these specific scenarios which take classifier predictive performance or classifier confidence values as input and determines dynamic chunk boundaries based on significant change in the confidence values. Unlike other adaptive sliding window techniques which detect change in error rates of classifiers, our second framework does not need true labels of all data instances for determining chunk boundaries dynamically. 4) We theoretically prove that, our proposed change detection will work better than the existing approaches in this particular scenario. 5) We integrate a concept evolution module to our frameworks. To the best of our knowledge, this is the first effort to address both concept drift and concept evolution using dynamically determined chunk sizes. 6) We evaluate our proposed frameworks on several benchmark and synthetic data sets. Results from the experiments show that our frameworks outperform other state-of-the-art approaches in data stream classification and novel class detection. We also show that, though our second framework does not use any labeled data instances for determining dynamic chunks, it still achieves competitive performance compared with the first framework, which uses all labeled instances.

The rest of the paper is organized as follows: in Section 2, we present a brief literature survey related to our work. Section 3 describes our approach in detail, including the confidence estimation and change detection methods. We describe the data sets, evaluation metrics and present experimental results in Section 4. Finally, Section 5 concludes the paper with directions to future work.

2 Related Works

Most state-of-the-art approaches (for example [1], [22], [25]) divide data streams into fixed-size chunks to solve the infinite length problem. These approaches use *abrupt forgetting* as only the latest chunk of data instances is kept in memory. Typically to address concept drift, each fixed-size chunk is used to retrain or update the classifier as soon as all the instances in the chunk are labeled. However, setting the fixed size of the chunks is very difficult in the context of an evolving data stream. Approaches using a fixed chunk size cannot capture the concept drift immediately if the chunk size is too large, or suffer from unnecessary frequent retraining during stable time periods if the chunk size is too small [5].

Gradual forgetting is used by [15], [16], [17]; which is a full memory approach for defining a window of the instances for learning. In *gradual forgetting*, each example is associated with a weight rather than discarding it from the memory completely. The weight typically is assigned based on the age of that data instance assuming that importance of an instance should decrease with its age. Old instances have weight close to zero based on the decay function, which is equivalent to discarding those instances from the memory. Various decay techniques are used in the literature. For example, a linear decay technique is used in [16], [17], and exponential decay is used in [15]. However, finding the perfect decay function is a challenge if information on the time-scale of change is not available [5]. In this paper, we determine the chunk size dynamically by using an explicit change point detection method.

Change detection techniques are used to determine the chunk sizes and to address the concept drift problem in [5], [11], [13]. There are two types of techniques in terms of change detection, i.e., detecting changes in the posterior distribution of the classes given the features $P(y|\mathbf{X})$, or detecting changes in the generating distribution $P(\mathbf{X})$ [12]. In the literature, several methods [18], [29] exist to deal with the change of $P(\mathbf{X})$. However, detecting a change in $P(\mathbf{X})$ is a hard problem especially in the case of multi-dimensional data [13]. In this paper, we focus on detecting change points in one-dimensional classifier confidence values.

Various techniques to detect changes in $P(y|\mathbf{X})$ have been proposed in [5], [9], [11], [13]. ADWIN [5] determines the size of the sliding window online according to the rate

of change observed from the windowed data itself. The window size increases until a significant change is detected. Unlike [5], [11] keeps track of the initial subwindow in which the error has been lowest so far. When the error rate over the entire current window significantly exceeds this lowest error rate, a change is declared. *Kruskal-Wallis* analysis and *Kolmogorov-Smirnov* tests are used by [9] to detect changes. The approach presented in [13] is based on obtaining statistics from the loss distribution of the learning algorithm by reusing the data multiple times via re-sampling. None of the above approaches handle concept evolution, delayed labeling or limited labeled data problems. One of our proposed approaches uses predictive performance of the classifier to manage the sliding window. However, unlike the above approaches, it also addresses the concept evolution problem.

The above techniques are mainly based on loss estimation of a predictor's performance. So, these assume that the true labels of the instances are readily available which will be used to evaluate the classifier performance, e.g., accuracy, precision, recall etc. This assumption is not practical in the context of data streams [20]. So, these approaches cannot address the limited labeled data or the delayed labeling problem. The second approach we present in this paper, is a framework which does not have this assumption, yet can detect dynamic chunk boundaries efficiently.

Besides addressing the infinite length and concept drift problems, our proposed frameworks also address the concept evolution problem in a multi-class environment. Various cluster-based novel concept detection techniques for data streams are proposed in [14], [30]. In these approaches, the instances which are not explained by the current decision model are labeled with an *unknown* profile. If a sufficient number of instances with the *unknown* profile can be found, clustering is applied on these instances. Valid clusters are evaluated as an extension of the normal class or a novelty. So, these are *single class* novelty detection methods, where authors assume that there is only one *normal* class and all other classes are novel. Thus, it is not suitable for a multi-class environment.

A novel class detection algorithm in a multi-class environment is proposed in [22], where each new test instance is considered as an outlier if it falls outside of the decision boundary of the ensemble classifier. A novel class is detected if a sufficient number of outliers have high cohesion among themselves and enough separation from the instances of existing classes. However, this approach divides the data stream into fixed-size chunks, hence suffers from the trade off problem discussed earlier. Unlike these approaches, we present frameworks which can detect novel classes in a multi-class environment using dynamically-determined chunk boundaries.

Three general strategies for transforming block-based ensembles into online learners are investigated in [8]. However, it does not investigate strategies related to concept evolution. An evaluation methodology for multi-class novelty detection algorithms is presented in [10]. In this work, we assume that only one novel class may appear at a time in the data stream. So, we use the traditional novel class detection performance evaluation metrics that have been used by most state-of-the-art approaches.

3 Proposed Approach

As discussed in Section 2, finding the fixed size of chunks or the rate of decay is a non-trivial task without prior knowledge on the rate of change [5], [12]. So, the approaches which use fixed-size chunks [1], [22], [25] suffer from the trade-off between performance during stable period, and sensitivity to sudden concept drift. To address this problem, we present two adaptive window-based approaches to determine chunk boundary dynamically and to handle the concept drift problem. In the first approach, we monitor predictive performance of the classifier on recent data instances. When there is a significant change in the predictive performance, we detect a chunk boundary, and the classifier is updated. We apply a suitable change detection method to detect the change in predictive performance of the classifier.

Dynamic window management used in the first approach is based on loss estimation of predictor performance like other existing sliding window-based approaches in the literature. This estimation requires true labels of data instances to calculate the predictive performance. However, in real-world data streams, labeled data is scarce and not readily available. Our first approach as well as other sliding window-based approaches in the literature might suffer in these scenarios. To address this challenge, we present another unsupervised dynamic window management approach, which uses classifier confidence instead of predictive performance. We propose suitable heuristics for this classifier to estimate the confidence while predicting the label of a test instance. We also design a change detection scheme for this scenario. Therefore, if a significant change is detected in classifier confidence, the chunk boundary is determined dynamically and the classifier is updated using only the recent labeled data.

In this paper, we present two frameworks for classifying and detecting novel classes in data streams by integrating semi-supervised *Classification* and *Novel Class Detection* modules with two different dynamic window management approaches discussed above. We refer to the first framework as "ADCMiner" (Adaptive Concept Drift Handler and Miner for data streams) and the second framework as "SAND" (Semi Supervised Adaptive Novel Class Detection and Classification over Evolving Stream Data). Next, we briefly discuss the classification and novel class detection methods used by both of these frameworks. ADCMiner uses supervised dynamic window management whereas SAND uses unsupervised window management. Since the classification and novel class detection methods we use are semi-supervised, SAND can work with limited labeled data and delayed labeling.

The high level workflow of the proposed frameworks is depicted in the Figure 1. Both of the frameworks maintain an ensemble of t classification models and a dynamic window W containing statistics on the classification, i.e., predictive performance or classifier confidence. Let $\{M_1, ..., M_t\}$ be the models in the ensemble. We define the terms *novel class* and *existing class* as follows:

Definition 1 (Existing class and Novel class) *Let M be the current ensemble of classification models. A class c is an existing class if at least one of the models $M_i \in M$ has been trained with class c. Otherwise, c is a novel class.*

TABLE 1: Commonly Used Symbols and Terms

t : Number of models in the ensemble classifier	$\mathcal{C}^x$: Confidence of the classifier in classifying x
$\mathcal{M}$: The ensemble classifier	τ : Classifier confidence threshold
M_i : i^{th} model in the classifier	$\mathcal{A}_i$: *Association* of model M_i in classification
$\mathcal{L}$: Set of labeled training data	$\mathcal{U}$: Set of unlabeled training data
$D_{ip}(x)$: Distance of instance x from h_{ip}	$\mathcal{P}_i$: *Purity* of model M_i in classification
$\mathcal{L}_{ip}(c)$: Frequency of class c in h_{ip}	y^k : True label of test instance k
h_{ip} : p^{th} pseudopoint in M_i	$\hat{y}_i^k$: Predicted label of test instance k by M_i
$R(h_{ip})$: Radius of h_{ip}	W : Dynamic sliding window

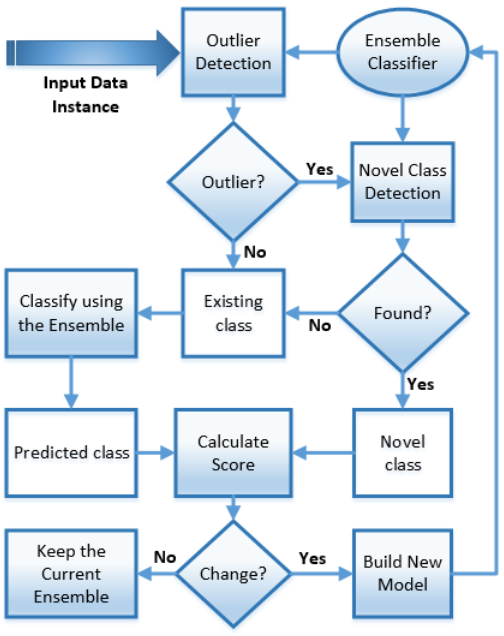


Fig. 1: High level work flow of the frameworks

At the beginning, the ensemble classifier contains models trained on the initial training data, i.e., warm-up period data instances. Once the warm up period is over, each incoming instance in the data stream is first examined by the *Outlier Detection* module to determine whether it is an outlier or not. This module detects an instance as an outlier if the instance falls outside of the decision boundary of the ensemble classifier. If the instance is not an outlier, it is classified as instance of an existing class using majority voting among the models in the ensemble. On the contrary, if the instance is an outlier, it is temporarily stored in a buffer. When there are enough instances in the buffer, the *Novel Class Detection* module is invoked. We define a class as *novel* if none of the models in the ensemble have been trained with any instances from that class. If a novel class is detected, the instances of the novel class are tagged accordingly. Otherwise, the instances in the buffer are considered as part of an existing class and classified using the current ensemble classifier.

In addition to predicting a label for a test instance, the proposed frameworks also calculate a score related to the performance of the classifier on the instance. These scores are stored in the window W. After inserting each score value, the *Change Detection* module searches for any change of distribution in the values stored in W. If it detects a significant change of distribution in the values, the chunk boundary is determined immediately containing instances corresponding to the values stored in W. Subsequently, a new model is trained only on the labeled instances of this chunk and the ensemble is updated by including the newly trained model. On the contrary, if the change detector finds no significant change in the confidence values, the current ensemble is retained and W keeps growing. Details on training, the classifier decision boundary, classification and the novel class detection processes will be discussed in Section 3.1 and 3.2. Table 1 contains the most common symbols and terms used throughout this paper. Next, we will describe the classification and novel class detection methods of our framework.

3.1 Training and Classification

Each model in the ensemble $M_i, i \in 1 \ldots t$ is a k-NN model formed with the training data. Rather than storing the raw training data, K clusters are built using a semi-supervised K-means clustering algorithm, and the cluster summaries (mentioned as *pseudopoints*) of each cluster are retained. These pseudopoints constitute the classification model. The summary contains the centroid, radius, and frequencies of data points belonging to each of the classes. The radius of a pseudopoint is equal to the distance between the centroid and the farthest data point in the cluster. The raw data points are discarded after creating the summary. Therefore, each model M_i is a collection of K pseudopoints. A test instance x is classified using M_i as follows. Let $h \in M_i$ be the pseudopoint whose centroid is the nearest to x. The predicted class of x is the class that has the highest frequency in h. The data point x is classified using the ensemble M by taking a majority vote among all classifiers.

Each pseudopoint corresponds to a "hypersphere" in the feature space with a corresponding centroid and radius. The *decision boundary* of a model M_i is the union of the feature spaces encompassed by all pseudopoints $h \in M_i$. The decision boundary of the ensemble M is the union of the decision boundaries of all models $M_i \in M$.

3.2 Novel Class Detection

Each instance in the data stream is first examined by the ensemble of models to see if it is outside of the ensemble decision boundary. If it is inside the decision boundary, then it is classified normally using the majority vote of the models in the ensemble. Otherwise, it is declared as an *F-outlier*, or filtered outlier.The main assumption behind novel class detection is that any class of the data has the following property.

Property 1 *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 that class will be far from existing class instances and will be close to other novel class instances. Since *F-outlier*s are outside the decision boundary, they are far from the existing class instances. So, the separation property for a novel class is satisfied by the *F-outlier*s. Therefore, *F-outlier*s are potential 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 *F-outlier*s that are close to each other. This is done by computing the q-Neighborhood Silhouette Coefficient, or q-NSC [21], which is defined as follows:

Definition 2 (q, c-neighborhood) *The q, c-neighborhood (or $q, c(x)$ in short) of an F-outlier x is the set of q class c instances that are nearest to x (i.e., q-nearest class c neighbors of x).*

Here q is a user defined parameter. For example, $q, c_1(x)$ of an *F-outlier* x is q-nearest neighbors of x from class c_1. Let $\bar{D}_{c_{out},q}(x)$ be the mean distance of an *F-outlier* x to its q-nearest *F-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:

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

The expression q-NSC is a unified measure of cohesion and separation, and yields a value between -1 and +1. A positive value indicates that x is closer to the *F-outlier* instances (more cohesion) and farther away from existing class instances (more separation), and vice versa. The q-NSC(x) value of an *F-outlier* x must be computed separately for each classifier $M_i \in M$. A *new class* is declared if there are at least q' ($> q$) *F-outlier*s having positive q-NSC for all classifiers $M_i \in M$.

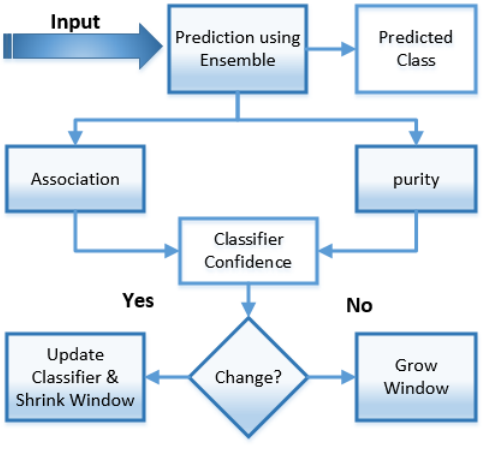


Fig. 2: Change Detection Module

3.3 Supervised Window Management

ADCMiner uses supervised window management by maintaining a variable-size window W to monitor performance of the classifier on recent data instances. The expectation is that the size of W increases during stable periods, i.e., when there is no significant change in the class boundaries and decreases when there is a concept drift. The basic intuition is that concept drift causes a change in class boundaries which worsens performance of the classifier if it is not updated in timely manner [32]. The variable-size window W is maintained as follows: after receiving each test data instance, the current ensemble classifier predicts the label of the instance as discussed in Section 3.1. After receiving the true label of this instance, performance of the classifier is evaluated as $e \in \{0, 1\}$ and inserted into W, where e is an indicator variable based on whether or not the model errors on the given instance. After inserting each performance evaluation value, the change detection algorithm is executed. Since each of the entries of W is either success or failure, each entry follows a Bernoulli distribution. As W is a collection of n such values, the values in the window follow a binomial distribution. It has two parameters, i.e., the number of trials n and the probability of success in each trial p.

3.4 Unsupervised Window Management

Unlike ADCMiner which stores the predictive performance of the classifier in the window W, SAND stores the confidence of the classifier on the prediction of test data instances in W. The confidence score on each test instance is calculated based on two heuristics (to be elaborated next). Therefore, it does not require true labels of the data instances for dynamic window management. We generate the confidence values in the range of $[0, 1]$. We describe calculation of the confidence of classifier and justify the use of the confidence estimators in Section 3.4.1 and Section 3.4.2, respectively. Moreover, in Section 3.4.6, we prove that when there is a concept drift in the data stream, the classifier will have lower confidence in classifying the instances from the stream. As a result, our change detection method is able to detect the change of distribution in confidences, in other words detect the concept drift.

We observe from our experiments on various data sets that generated confidence values tend to follow a *beta* distribution. We have carried out a *chi-square* goodness of fit test on the generated confidence scores in order to confirm that observation. The *beta* distribution forms a rich two-parameter family of distributions on the interval $[0, 1]$. Given parameters $\alpha, \beta \in (0, \infty)$, the expectation and variance of the *beta* distribution are $\mu = \alpha/(\alpha + \beta)$ and $\sigma^2 = \alpha\beta/(\alpha + \beta)^2(\alpha + \beta + 1)$ respectively. Next, we describe the method we propose to calculate the confidence of the classifier.

3.4.1 Calculation of Confidence Scores

Figure 2 shows the workflow of the *Change Detection* module of SAND. Along with predicting the label of each new test data instance, we use two different heuristics, *association* and *purity*, to estimate the confidence of each individual model in the classifier. Finally, we combine the individual model confidence estimates to estimate the overall confidence of the ensemble classifier and store it in W. Subsequently, a change detection method is applied on the values stored in W to detect any change in the distribution of the confidence

values which will be discussed in Section 3.5. If a change is not detected, we keep using the same classifier and W keeps growing. On the contrary, if a change is detected, we update the classifier and then shrink the window.

Let h_{ip} be the p^{th} pseudopoint in M_i and let c_m be the class having highest frequency in h_{ip}. Assuming the closest pseudopoint from instance x in model M_i is h_{ip}, we use the following heuristics to estimate confidence of model M_i:

- *Association* is calculated by $R(h_{ip}) - D_{ip}(x)$, where $R(h_{ip})$ is the radius of h_{ip} and $D_{ip}(x)$ is the distance of x from h_{ip}.
- *Purity* is calculated by $\frac{|\mathcal{L}_{ip}(c_m)|}{|\mathcal{L}_{ip}|}$, where $|\mathcal{L}_{ip}|$ is sum of all the *frequencies* in h_{ip} and $|\mathcal{L}_{ip}(c_m)|$ is the *frequency* of c_m in h_{ip}.

The *association* and *purity* of the model M_i is denoted by $\mathcal{A}_i$ and $\mathcal{P}_i$, respectively. Both of the above heuristics contribute to the confidence score of a model according to their estimation capability. We measure this capability by calculating the correlation coefficient between heuristic values and classification accuracy for each model M_i using the initial training instances as follows: M_i calculates confidence heuristic values for each of the labeled training instances. Let $\mathcal{H}_{ij}^k$ be the value of j^{th} heuristic in M_i's classification of instance k. Since we use two heuristics, $j \in \{1, 2\}$. Let $\hat{y}_i^k$ be the prediction of M_i on instance k and y^k be the true label of that instance. Let v_i is the vector containing v_i^k values indicating whether the classification of instance k by model M_i is correct or not. In other words, $v_i^k = 1$ if $\hat{y}_i^k = y^k$ and $v_i^k = 0$ if $\hat{y}_i^k \neq y^k$. Finally, the correlation vector r_i is calculated for model M_i. It contains r_{ij} values which are Pearson's correlation coefficients between $\mathcal{H}_{ij}$ and v_i for different j.

Correlation coefficients calculated in the training phase are used for classification and confidence estimation during the testing phase as follows. First, our framework SAND calculates confidence heuristic values $\mathcal{H}_i^x$ for a test instance x. Let $\mathcal{C}_i^x$ be the confidence of model M_i in predicting test instance x. $\mathcal{C}_i^x$ is calculated by taking the dot product of $\mathcal{H}_i^x$ and v_i, i.e., $\mathcal{C}_i^x = \mathcal{H}_i^x.v_i$. In the same way, SAND calculates confidence values for each of the models in the ensemble along with the prediction for each test instance. After calculating all the confidence values, we normalize these between 0 and 1. The normalized confidence value $\mathcal{C}_i^x$ is treated as a weight for the prediction $\hat{y}_i$ by model M_i. Finally, to estimate the confidence of the entire ensemble denoted by $\mathcal{C}^x$, SAND takes the average confidence of the models in the ensemble towards the predicted class.

3.4.2 Justification of Confidence Heuristics

In this Section, we first define the objective function for semi-supervised K-means clustering mentioned in Section 3.1. Then, based on the objective function, we theoretically justify the choice of the heuristics for estimating classifier confidence.

3.4.3 Objective Function

Given a limited amount of labeled data, the goal of impurity-based clustering is to create K clusters by minimizing the intra-cluster dispersion (unsupervised K-means has the same goal) and at the same time minimizing the impurity of each cluster. We refer to this problem as K-means with Minimization of Cluster Impurity (MCI-Kmeans). A cluster is completely pure if it contains labeled data points from only one class (along with some unlabeled data). Thus, the objective function should penalize each cluster for being impure. The general form of the objective function is as follows:

$$O_{MCIKmeans} = \sum_{i=1}^{K} \sum_{x \in \mathcal{X}_i} ||x - \mu_i||^2 + \sum_{i=1}^{K} \mathcal{W}_i * Imp_i \quad (2)$$

where $\mathcal{W}_i$ is the weight associated with cluster i, Imp_i is the impurity of cluster i and $\mathcal{X}_i$ is the set of all (both labeled and unlabeled) points in cluster i. To ensure that both the intra-cluster dispersion and cluster impurity are given the same importance, the weight associated with each cluster is chosen to be

$$\mathcal{W}_i = |\mathcal{L}_i| * \bar{\mathcal{D}}_{\mathcal{L}_i} \Rightarrow \mathcal{W}_i = \sum_{x \in \mathcal{L}_i} ||x - \mu_i||^2$$

where $\mathcal{L}_i$ is the set of all labeled data points in Cluster i and $\bar{\mathcal{D}}_{\mathcal{L}_i}$ is the average dispersion from each of these labeled points to the cluster centroid. Any impurity measure can be plugged in to Equation 2. We use the following impurity measure: $Imp_i = ADC_i * Ent_i$, where ADC_i is the "Aggregated Dissimilarity Count" of cluster i and Ent_i is the entropy of cluster i. The Dissimilarity count $DC_i(x, y)$ of a data point x in cluster i having class label y is the total number of labeled points in that cluster belonging to classes other than y. If x is unlabeled (i.e., $y = \emptyset$), then $DC_i(x, y)$ is zero. In other words, $DC_i(x, y) = 0$, if x is unlabeled, and $DC_i(x, y) = |\mathcal{L}_i| - |\mathcal{L}_i(c)|$, if x is labeled and its label $y = c$, where $\mathcal{L}_i(c)$ is the set of labeled points in cluster i belonging to class c. The "Aggregated Dissimilarity Count" or ADC_i is the sum of the dissimilarity counts of all the points in cluster i: $ADC_i = \sum_{x \in \mathcal{L}_i} DC_i(x, y)$. The entropy of a cluster i is computed as: $Ent_i = \sum_{c=1}^{C} (-p_c^i * log(p_c^i))$, where p_c^i is the prior probability of class c, i.e., $p_c^i = \frac{|\mathcal{L}_i(c)|}{|\mathcal{L}_i|}$.

After combining all the above terms, our objective function is

$$O_{MCIKmeans} = \sum_{i=1}^{K} \sum_{x \in \mathcal{X}_i} ||x - \mu_i||^2 + \sum_{i=1}^{K} \sum_{x \in \mathcal{L}_i} ||x - \mu_i||^2 * \sum_{x \in \mathcal{L}_i} (|\mathcal{L}_i| - |\mathcal{L}_i(c)|) * \sum_{c=1}^{C} \left(-\frac{|\mathcal{L}_i(c)|}{|\mathcal{L}_i|} * \log \frac{|\mathcal{L}_i(c)|}{|\mathcal{L}_i|} \right) \quad (3)$$

Let h_{ip} and h_{jq} be the closest pseudopoint from a test data point x in model M_i and M_j respectively. We define M_i will have higher confidence than model M_j in classifying a test data instance x, if including x in h_{jq} increases the objective function more than including x in h_{ip}. We define the following terms:

$\Delta Disp_i^x \leftarrow$ increase in intra-cluster dispersion due to adding x to the closest pseudopoint in the Model M_i.

$\Delta ADC_i^x \leftarrow$ increase in the "Aggregated Dissimilarity Count" due to adding x to the closest pseudopoint in the Model M_i.

$\Delta Ent_i^x \leftarrow$ increase in Entropy due to adding x to the closest pseudopoint in the model M_i.

3.4.4 Association

Consider two cases: 1) x is inside only one of h_{ip} and h_{jq}. Without loss of generality, let assume that x falls inside h_{ip} but outside of h_{jq}. Therefore,

$$R(h_{ip}) > D_{ip}(x) \Rightarrow R(h_{ip}) - D_{ip}(x) > 0 \Rightarrow \mathcal{A}_i(x) > 0$$

and

$$R(h_{jq}) < D_{jq}(x) \Rightarrow R(h_{jq}) - D_{jq}(x) < 0 \Rightarrow \mathcal{A}_j(x) < 0$$

So, from the above equations, we get the following:

$$\mathcal{A}_i(x) > \mathcal{A}_j(x) \tag{4}$$

If x falls inside h_{ip} but outside of h_{jq}, h_{ip} has a greater association than h_{jq}. As a result, M_i will have greater confidence than M_j as expected. Moreover, since the point x falls outside of decision boundary of h_{jq} but inside of h_{ip}, $||\mu_{jq} - x||^2 > ||\mu_{ip} - x||^2 \Rightarrow \Delta Disp_i^x < \Delta Disp_j^x$. So, including x into h_{jq} increases the objective function value (Equation 3) more than that of h_{iq}. So, M_i will have more confidence than M_j in classifying x.

2) Test instance x falls into the same side of both h_{ip} and h_{jq}. Without loss of generality, let assume that, h_{ip} and h_{jq} have similar characteristics (e.g., *centroid*, *radius* etc). Let us also assume that M_i has a higher *association* than M_j in the case of classifying x. Therefore, we can deduce:

$$\begin{aligned}
&\mathcal{A}_i(x) > \mathcal{A}_j(x)\\
&\Rightarrow R(h_{ip}) - D_{ip}(x) > R(h_{jq}) - D_{jq}(x)\\
&\Rightarrow R(h_{ip}) + D_{jq}(x) > R(h_{jq}) + D_{ip}(x)\\
&\Rightarrow D_{jq}(x) > D_{ip}(x) \quad (\text{since } R(h_{ip}) = R(h_{jq}))\\
&\Rightarrow ||\mu_{jq} - x||^2 > ||\mu_{ip} - x||^2\\
&\Rightarrow \Delta Disp_i^x < \Delta Disp_j^x
\end{aligned} \tag{5}$$

So, in case 2, including x into h_{jq} increases the objective function more than including x into h_{ip}. So, in both cases, greater *association* leads to better confidence.

3.4.5 Purity

Assume that h_{ip} predicts x as an instance of c_m and h_{iq} predicts x as an instance of c_n. Let us also assume that h_{ip} and h_{jq} share similar properties except that h_{ip} has a higher *purity* than h_{jq} in predicting data point x. There can be two different cases:

1) Both of the pseudopoints contain an equal number of labeled points, i.e., $|\mathcal{L}_{ip}| = |\mathcal{L}_{jq}|$. Then,

$$\begin{aligned}
&\mathcal{P}_i(x) > \mathcal{P}_j(x)\\
&\Rightarrow \frac{|\mathcal{L}_{ip}(c_m)|}{|\mathcal{L}_{ip}|} > \frac{|\mathcal{L}_{jq}(c_n)|}{|\mathcal{L}_{jq}|}\\
&\Rightarrow |\mathcal{L}_{ip}(c_m)| > |\mathcal{L}_{jq}(c_n)| \quad (\text{Since } |\mathcal{L}_{ip}| = |\mathcal{L}_{jq}|)\\
&\Rightarrow -|\mathcal{L}_{ip}(c_m)| < -|\mathcal{L}_{jq}(c_n)|\\
&\Rightarrow |\mathcal{L}_{ip}| - |\mathcal{L}_{ip}(c_m)| < |\mathcal{L}_{jq}| - |\mathcal{L}_{jq}(c_n)| \ (\text{Since } |\mathcal{L}_{ip}| = |\mathcal{L}_{jq}|)\\
&\Rightarrow DC_{ip}(x, c_m) < DC_{jq}(x, c_n)\\
&\Rightarrow \Delta ADC_{ip}^x < \Delta ADC_{jq}^x
\end{aligned} \tag{6}$$

2) The pseudopoints contain an unequal number of labeled points, i.e., $|\mathcal{L}_{ip}| \neq |\mathcal{L}_{jq}|$. Then,

$$\begin{aligned}
&\mathcal{P}_i(x) > \mathcal{P}_j(x)\\
&\Rightarrow \frac{|\mathcal{L}_{ip}(c_m)|}{|\mathcal{L}_{ip}|} > \frac{|\mathcal{L}_{jq}(c_n)|}{|\mathcal{L}_{jq}|}
\end{aligned}$$

Again since

$$p_{c_m}^i = \frac{|\mathcal{L}_{ip}(c_m)|}{|\mathcal{L}_{ip}|} \ ; \ p_{c_n}^j = \frac{|\mathcal{L}_{jq}(c_n)|}{|\mathcal{L}_{jq}|}$$

So,

$$p_{c_m}^i > p_{c_n}^j \Rightarrow \Delta Ent_{ip}^x < \Delta Ent_{jq}^x \tag{7}$$

Equation 6 and Equation 7 show that in both cases, including x into the closest pseudopoint in model M_j will increase the value of the objective function more than that of model M_i. Thus, a higher value of *purity* leads to higher confidence.

3.4.6 Effect of Concept Drift on Confidence Score

In this Section, we show that classifier confidence can decrease due to a concept drift. First, we will see the relationship between the intra-cluster dispersion in terms of sum of squared error (SSE) and *association*. Let SSE_p denote the SSE of pseudopoint h_{ip}, μ_{ip} be the centroid of h_{ip}, and x an arbitrary data point. Therefore, the SSE_i of a classification model is defined as follows:

$$\begin{aligned}
SSE_i &= \sum_p SSE_p = \sum_p \sum_{x \in h_{ip}} (x - \mu_{ip})^2\\
&= \sum_p n_{ip} \frac{\sum_{x \in h_{ip}} (x - \mu_{ip})^2}{n_{ip}} = \sum_p n_{ip} \bar{D}_{ip}
\end{aligned} \tag{8}$$

where $\bar{D}_{ip}$ is the mean distance between a data point in h_{ip} and the centroid of h_{ip}, and n_{ip} is the number of instances in h_{ip}. Now the Sum of *Association* of the model M_i can be formulated as follows:

$$\begin{aligned}
\mathcal{A}_i &= \sum_p \mathcal{A}_{ip} = \sum_p \sum_{x \in h_{ip}} (R(h_{ip}) - (x - \mu_{ip})^2)\\
&= \sum_p n_{ip} R(h_{ip}) - \sum_p \sum_{x \in h_{ip}} (x - \mu_{ip})^2\\
&= \sum_p n_{ip} R(h_{ip}) - SSE_i
\end{aligned} \tag{9}$$

Equation (9) concludes that total model *association* is inversely proportional to the model SSE because the pseudopoint radii are fixed (can be considered constant) on a given model.

Next, we show that a concept drift increases model SSE. We describe concept drift as the drift of the decision boundary, obtained by drifted pseudopoints. Assume, without loss of generality, that concept drift is quantified by the amount of drift of the pseudopoints, which is obtained by δ_{ip} displacement of the centroid of h_{ip} for all p in the model. Assume that a new window (i.e., chunk) appears in the stream that exhibits the drift. Therefore, the new chunk can be obtained by moving the center of each pseudopoint h_{ip} in the current model by δ_{ip}, while keeping the same distribution of data points in each pseudopoint. Let the drifted pseudopoint be h'_{ip}. Now if we randomly draw n_{ip} data points from h'_{ip}, and calculate the mean distance between each a point x' and the centroid μ_{ip} (of the old pseudopoint), the mean distance would be higher by δ_{ip} amount. Let SSE'_i be the SSE of the new model, and D'_{ip}

be the mean distance between new data points and old pseudopoint. Therefore, we can derive that:

$$SSE'_i = \sum_p n_{ip} D'_{ip} = \sum_p n_{ip}(\bar{D}_{ip} + \delta_{ip}) = \sum_p n_{ip}\delta_{ip} + \sum_p n_{ip}\bar{D}_{ip} = \sum_p n_{ip}\delta_{ip} + SSE_i > SSE_i \quad (10)$$

Since $SSE'_i > SSE_i$, from equation (9) we can follow up that $A'_i < A_i$, meaning that *association*, and as a consequence, confidence in classifying the new data chunk will be lower because of the concept drift. The higher the amount of drift, the lower the confidence will be.

3.5 Change Detection and Updating the Ensemble

As discussed earlier, ADCMiner stores predictive performance scores in a variable-size window W. On the contrary, SAND stores confidence scores of the classifier on recent data instances in W. After inserting each score in W, the *Change Detection* module is invoked to check if there is any significant change of distribution among the values stored in it.

Algorithm 1 Change Detection Algorithm

1: $W \leftarrow \emptyset$
2: $T_h \leftarrow -log(\alpha)$ // α is sensitivity parameter.
3: **while** true **do**
4: $x \leftarrow$ the latest data point in the stream
5: $[\hat{y}, score(x)] \leftarrow$ Classify(x) // $\hat{y}$ is the predicted label and $score(x)$ is the associated score calculated as discussed in Section 3.3 and Section 3.4.
6: $W \Leftarrow score(x)$
7: $N \leftarrow |W|$; $w_n \leftarrow 0$; $ecp \leftarrow -1$ // ecp contains the estimated change point
8: **for** $k \leftarrow \Delta$ **to** $N - \Delta$ **do**
9: $m_b \leftarrow$ mean$(W[1:k])$
10: $m_a \leftarrow$ mean$(W[k+1:N])$
11: **if** $m_a \leq (1-\alpha) * m_b$ **then**
12: $S_k \leftarrow 0$
13: $Dist_b \leftarrow$ estimateParam$(W[1:k])$
14: $Dist_a \leftarrow$ estimateParam$(W[k+1:N])$
15: **for** $i \leftarrow k+1$ **to** N **do**
16: $S_k \leftarrow S_k + log\left(\frac{f(W[i] \mid Dist_a)}{f(W[i] \mid Dist_b)}\right)$
17: **end for**
18: **if** $S_k > w_n$ **then**
19: $w_n \leftarrow S_k$; $ecp \leftarrow k$
20: **end if**
21: **end if**
22: **end for**
23: **if** $w_n > T_h$ **then**
24: retrainClassifier(W)
25: $W \leftarrow W[ecp+1:N]$
26: **end if**
27: **end while**

In both cases, we know the family of underlying distribution of the values stored in W. Moreover, change detection in these cases refers to detecting a change of parameters, not a change of distribution family. Therefore, we propose a CUSUM (CUmulative SUM) type parametric change detection algorithm on corresponding distributions to use in this context. In other words, we design CUSUM based on a *binomial distribution* for ADCMiner and CUSUM based on a *beta distribution* for SAND.

The overview of the change detection method is the following. It starts by dividing W into every possible pair of sub-windows W_b and W_a having at least a minimum number of values. Then it estimates the parameters of the distributions in each of the split from the values stored in the corresponding split. If the number of observations are sufficiently large, the *method of moments* estimations hold *consistency* and *asymptotic normality* properties. Our change point detection method splits W in such a way that each sub-window contains at least $\Delta = 100$ observations. Typically, this number of observations is good enough to make sure that the estimations of the parameters are close to the real parameter values. Finally, it detects the change point based on the sum of likelihood ratios.

Algorithm 1 sketches our change detection method. As soon as a new test instance arrives, the current version of the classifier is used to predict the label along with calculating a score shown at Line 5. Predictive performance and classifier confidence are used as scores respectively by ADCMiner and SAND as discussed in Section 3.3 and Section 3.4. Then, the proposed change detection method divides the window W for each k between Δ to $N - \Delta$ where N is the total number of observations in W. $W_b = W[1:k]$ contains relatively older observations and $W_a = W[k+1:N]$ contains the recent observations. When concept drift occurs, both predictive performance and confidence values are expected to be decreased. So, we want to detect the changes in the negative direction only in both ADCMiner and SAND. In other words, if m_a and m_b are the mean values of the observations in W_a and W_b, we want to detect the change only when $m_a < m_b$. Let α be the sensitivity parameter, we only execute the change point detection if $m_a \leq (1-\alpha) * m_b$ to avoid any discrete noise or sudden short term spikes. In our experiments, we use $\alpha = 0.05$.

After splitting W into two sub-windows having at least Δ number of observations, our change detection method estimates the parameters of the distribution from the values in each of the splits using the *method of moments* technique at Line 13 and 14. The *binomial distribution* has two parameters, i.e., the number of trials n and the probability of success p. In ADCMiner framework, n is estimated by the number of observations in the sub-window. p is estimated by the mean value ($\bar{x}$) in the sub-window. Similarly, the *beta distribution* has two positive shape parameters, i.e., α and β. *Method of moments* estimations of the parameters of the *beta* distribution are the following:

$$\hat{\alpha} = \frac{\bar{X}^2 - \bar{X}^3}{s^2} - \bar{X} \quad (11)$$

$$\hat{\beta} = \hat{\alpha}\left(\frac{1}{\bar{X}} - 1\right) \quad (12)$$

Where $\bar{X}$ is the sample mean and s is the sample standard deviation. Let $Dist_b$ and $Dist_a$ be the estimated distributions from W_b and W_a respectively. The sum of the log

likelihood ratios is calculated in the inner loop between Lines 15 and 17 using the following formula:

$$S_k = \sum_{i=k+1}^{n} log\left(\frac{f\left(W[i] \mid Dist_a\right)}{f\left(W[i] \mid Dist_b\right)}\right) \quad (13)$$

Where $f\left(X_i \mid Dist\right)$ is the probability density function (PDF) of the distribution $Dist$ applied on the data instance X_i. The *CUSUM* process score w_n for the values in the whole window W is calculated in the outer loop between Lines 8 and 22 using the following formula:

$$w_n = \max_{\Delta \leq k \leq N-\Delta} S_k \quad (14)$$

Let k_{max} be the value of k for which the algorithm calculated the maximum S_k value where $\Delta \leq k \leq N - \Delta$. If w_n is greater than a pre-fixed threshold, then a change point is detected at point k_{max}. We fix the threshold based on the value of the sensitivity α. In our experiments, we use $-log(\alpha)$ as the threshold value.

If the change detection method detects a change in the distribution, a new model is built on the data instances corresponding to scores stored in W and the sub-window W_b is subsequently dropped. Once a new model is trained, it replaces the oldest one among the existing models in the ensemble. This ensures that we have exactly L models in the ensemble at any given point of time. In this way, the infinite length problem is addressed because a constant amount of memory is required to store the ensemble. The concept drift problem is addressed by keeping the ensemble up-to-date with the most recent concept.

3.6 Optimal Properties of Proposed Change Detection Technique

Our proposed CUSUM-type change detection technique is based on the sum of log-likelihood ratios rather than the raw observations x_i. For this reason, it is able to detect arbitrary changes in the distribution of x_i, and not only the mean shifts. This is important for our situation, when the range of observed x_i is from 0 to 1. Changes in a location parameter alone are mere shifts, and they necessarily imply changes of the range, hence it is important to detect changes in other parameters. It is proven mathematically that among all the sequential change-detection procedures with a given rate of false alarms, the CUSUM rule has the lowest expected detection delay [2], [24]. Stating this rigorously, the CUSUM procedure minimizes the "worst" mean delay

$$\operatorname{ess\,sup} E_\nu\{(T-\nu)^+ | x_1, \ldots, x_\nu\}$$

among all the stopping rules with the mean time between false alarms

$$E_\infty(T) \geq \gamma,$$

where T is the stopping time; $(T-\nu)^+$ is the detection delay which equals 0 if $T < \nu$, i.e., in case of a false alarm; $\operatorname{ess\,sup}$ represents the longest expected delay given the first ν data points; this essential supremum is attained when the CUSUM process equals 0 at the change-point ν [24]; γ is a chosen positive constant; and $E_\infty(T)$, the expected stopping time given $\nu = \infty$, represents the average time until a false alarm when no change has ever occurred. This mean time until a false alarm is used to measure the rate of false alarms.

As discussed in Section 2, various techniques to detect changes in $P(y|\mathbf{X})$ have been proposed in the literature [5], [11]. Among them, ADWIN is widely used and provides rigorous guarantees of performance, as bounds on the rates of false positives and false negatives [5]. Comparing with ADWIN, the CUSUM procedure is in general more optimal. The ADWIN rule guarantees, using the normal approximation for large window size n, that a false alarm has probability δ/n at any moment inside the window. Then the probability of at least one false alarm in a given window does not exceed n, this rate being preserved by the *Bonferroni* inequality, a standard approach to multiple testing. However, the *Bonferroni* inequality is not sharp and not efficient, especially for large n. The CUSUM algorithm is a sequential procedure that controls the same probability of a false alarm accurately, based on the analysis of the log-likelihood ratio process as a random walk instead of the *Bonferroni* inequality.

The difference between [24] and our approach is the presence of nuisance parameters, e.g., α and β of the *beta* distribution that we estimate before and after the potential change point by their maximum likelihood estimators $\hat{\alpha}$ and $\hat{\beta}$. The resulting generalized-likelihood-ratio (GLR) process appears exponentially close to the original likelihood ratio. That is, the largest difference between the original CUSUM process and the GLR process with inserted estimated parameters can exceed an exponentially small threshold $n^{-\beta}$ only with an exponentially small probability, as $n \to \infty$ [3]. Therefore, the GLR-based CUSUM process remains an asymptotically optimal change detection. While its mean detection delay is a linear function of the chosen threshold, its mean time between false alarms is exponentially long [4].

3.7 Analysis on Time and Space Complexity

Our proposed frameworks have mainly four modules, i.e., *Classification*, *Change Detection*, *Novel Class Detection*, and *Update*. The buffer which stores *outliers* temporarily is examined periodically by calling the *Novel Class Detection* module to see if a novel class has arrived. Time complexity of the *Novel Class Detection* module is $O(KW)$ where K is the number of *pseudopoints*. This module is invoked only when the buffer contains q number of instances in it. Including this periodic call to the *Novel Class Detection* module, the total time complexity for classification is $O(KtW+tW+mKW)$, where t is the number of models in the ensemble classifier and $m = W/q$. Since $m >> Kt$, the total time complexity for the *Classification* module is $O(mKW)$. The time complexity for invoking *Change Detection* module is $O(W^2)$. So, the overall time complexity for executing the frameworks is $O(mkW + W^2 + f(W))$, where $f(W)$ is the time to train a new classifier with W training instances. Most often, $W >> m$ and $W >> k$, so the total time complexity is essentially $O(W^2 + f(W))$. We use four buffers, i.e., *filtered outlier buffer*, *training buffer*, *unlabeled data buffer* and *dynamic sliding window*. All the buffers can contain at most W instances, where W is the maximum allowable size for the dynamic sliding window. So, space complexity for our frameworks is $O(W)$.

4 EXPERIMENT RESULTS

In this section, we evaluate our proposed approaches both on several benchmark real-world data sets and synthetic data sets. We compare performance of our proposed approach against several well known base-line approaches in terms of different performance metrics on both classification and novel class detection.

4.1 Data Sets

We use four real and six different types of synthetic data sets to test performance of SAND along with some other baseline approaches. Table 2 depicts the characteristics of the data sets.

TABLE 2: Characteristics of Data Sets

Name of Data set	Num of Instances	Num of Classes	Num of Features
ForestCover	150,000	7	54
Electricity	45,312	2	8
Power Supply	29,927	2	24
PAMAP	150,000	19	52
SynCN	100,000	20	40
SyntheticLED	100,000	10	7
HyperPlane	100,000	5	10
SynRBF@0.002	100,000	7	70
SynRBF@0.003	100,000	7	70

The ForestCover data set is obtained from the UCI repository as explained in [22]. We normalize the data set, and arrange the data in order to prepare it for novel class detection so that in any chunk at most three and at least two classes co-occur, and new classes appear randomly.

The Electricity [23] data set contains data collected from the Australian New South Wales Electricity Market. In this market, price is affected by demand and supply. The class label identifies the change of the price relative to a moving average of the last 24 hours. The Power Supply [28] data set contains hourly power supply of an Italian electricity company which records the power from two sources: power supply from main grid and power transformed from other grids. The learning task is to predict which hour (1 out of 24 hours) the current power supply belongs to.

We have also collected the Physical Activity Monitoring data set (PAMAP) from the UCI [27] repository. In this data set, nine people were equipped with sensors that gathered a total of 52 streaming metrics features whilst they performed activities. Nineteen total activities were identified as class labels - including one category for miscellaneous or transient activities.

Besides using real-world data sets, we also use synthetically generated data sets. SynCN (Synthetic Data with Concept-Drift and Novel Class) is a synthetic data set generated using the following equation: $\sum_{i=1}^{d} a_i x_i = a_0$ [22]. There are several parameters that simulate concept-drift in these data sets. 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 $\{x_1, \ldots, x_d\}$, where $x_i \in [0, 1]$.

The SyntheticLED data set is generated using the MOA [7] framework which simulates the LED digits classification problem with 10% noise added. In this data set, seven segments correspond to seven binary features, and the digits "0" through "9" represent 10 classes.

HyperPlane is a synthetic data stream which is generated using the equation: $f(x) = \sum_{j=1}^{d-1} a_j \frac{(x_j + x_{j+1})}{x_j}$, where $f(x)$ is the label of instance x and $a_j,\ j = 1, 2, .., d$, controls the shape of the decision surfaces. SynRBF@X are other synthetic data sets generated using RandomRBFGenerator-Drift of the MOA [7] framework where X is the Speed of change of centroids in the model. We generate two such data sets using different X to check how efficiently different approaches can adapt to a concept drift.

We use the ForestCover, PAMAP and SynCN data sets for simulating both concept drift and novel classes. The rest of the data sets are used to test only concept drift handling capability of different approaches.

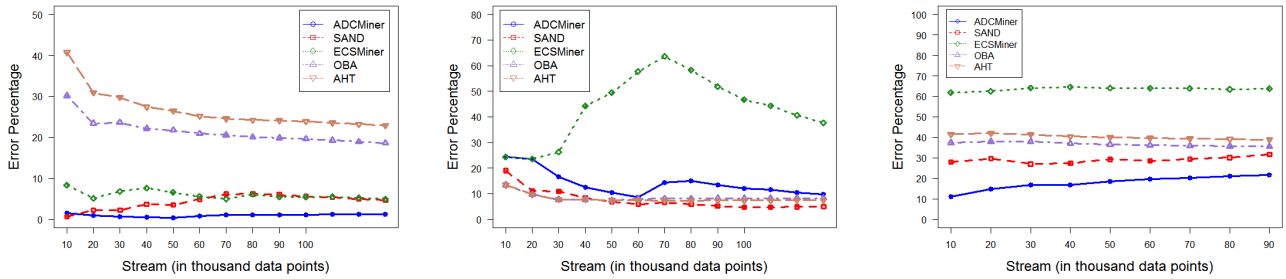
4.2 Experimental Setup

We implemented both ADCMiner and SAND using Java version $1.7.0.51$. To evaluate performance, we used a virtual machine which is configured with 8 cores and 16 *GB* of RAM. The clock speed of each virtual core is 2.4 *GHZ*.

We compare classification and novel class detection performance of our approaches with ECSMiner [22]. We have chosen ECSMiner since it is one of the most robust and efficient frameworks available in the literature for classifying data streams having both concept drift and concept evolution. However, it uses a fixed chunk size where our proposed approach uses a variable chunk size based on the change in classifier performance.

Other than that, we compare classification performance of the proposed approaches with OzaBagAdwin (OBA) and Adaptive Hoeffding Tree (AHT) implemented in the MOA [7] framework, since these approaches seem to have superior performance than others on the data sets used in the experiments. Both OBA and AHT uses ADWIN [5] as the change detector. These approaches do not have a novel class detection feature. So, we compare these approaches with our approach only in terms of classification performance.

Other than ECSMiner, we use two more baseline approaches to compare novel class detection performance of our proposed frameworks. These two baseline approaches are a combination of two techniques: OLINDDA [31] and Weighted Classifier Ensemble (WCE) [33], where the former works as novel class detector and latter performs classification as shown in [22]. 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, two alternative solutions are proposed. First, parallel OLINDDA models are built, one for each class, which evolve simultaneously. Whenever the instances of a novel class appear, a new OLINDDA model is built for that class. A test instance is declared as novel, if all the existing class models identify this instance as novel. We refer to this method as W-OP. Second, initially an OLINDDA model is built using all the available classes with the first few 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 W-OS.



(a) ForestCover (b) PAMAP (c) SynCN

Fig. 3: Number of Instances v.s. Overall Error of Each Method

TABLE 3: Summary of Classification Results

Name of Data set	ADCMiner Error%	SAND Error%	ECSMiner Error%	AHT Error%	OBA Error%
ForestCover	**1.12**	4.46	4.55	22.89	18.06
Electricity	**0.01**	**0.01**	0.022	27.72	22.26
Power Supply	**0.01**	**0.01**	**0.01**	85.59	86.92
PAMAP	9.07	**5.01**	35.26	8.76	7.27
SynCN	**0.01**	0.99	**0.01**	4.81	4.5
SyntheticLED	0.55	**0.41**	**0.41**	26.09	26.09
HyperPlane	**0.01**	**0.01**	3.73	46.24	48.55
SynRBF@0.002	**21.58**	31.36	63.43	38.75	37.04
SynRBF@0.003	**36.47**	39.50	65.39	48.65	46.86

We evaluate each of the above classifiers on a stream by testing and then training with chunks of data in sequence. To evaluate ECSMiner, W-OP and W-OS, we use 50 pseudo-points, ensemble size 6, and 95% of labeled training data as suggested in [22]. On the other hand, we use 100% labeled training data in case of OzaBagAdwin (OBA) and Adaptive Hoeffding Tree (AHT) since training and updating of these approaches are fully supervised.

4.3 Performance Metrics

Let FN = total novel class instances misclassified as existing class, FP = total existing class instances misclassified as novel class, TP = total novel class instances correctly classified as novel class, Fe = total existing class instances misclassified (other than FP), N_c = total novel class instances in the stream, N = total instances the stream. We use the following performance metrics to evaluate our technique:

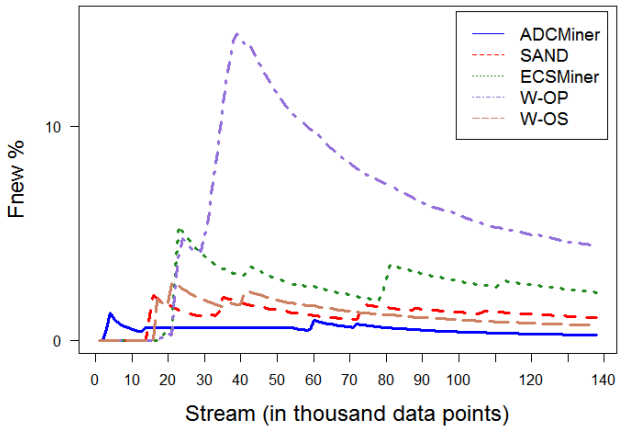
1) *Error%*: Total misclassification error (percent), i.e., $\frac{(FP+FN+Fe)*100}{N}$.
2) M_{new}: % of novel class instances misclassified as existing class, i.e., $\frac{FN*100}{N_c}$.
3) F_{new}: % of existing class instances Falsely identified as novel class, i.e., $\frac{FP*100}{N-N_c}$.
4) F_2: F_β score provides the overall performance of a classifier by considering both $precision$ and $recall$. In this paper, we use $\beta = 2$, which gives us $F_2 = \frac{5*TP}{5*TP+4*FN+FP}$.

In the case of the F_2 measure, a higher value is better. In case of all the other metrics, a lower value is better.

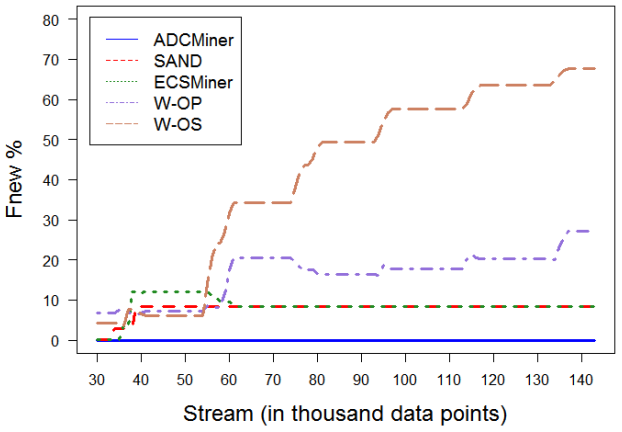
4.4 Classification Performance

As discussed earlier, most of the state-of-the-art techniques to classify evolving data stream, e.g., ECSMiner divide the data stream into fixed-size chunks regardless of occurrence or intensity of concept drift. However, our proposed approaches determine the chunk size dynamically based on the feedback from the classifier, i.e., predictive performance or confidence values of the classifier. In this way, ADCMiner and SAND avoid unnecessary training during stable periods and frequently update the classifier when needed which is also supported by the experimental results. As an instance, with increasing speed of change of centroids X in SynRBF@X data sets, our change detection technique helps SAND to update the ensemble classifier more frequently to cope with more frequent concept drift. SAND creates 231 and 249 number of chunks while classifying SynRBF@0.002 and SynRBF@0.003 data sets respectively where ECSMiner creates the same number of chunks in both of the cases. ADCMiner creates 522 and 550 number of chunks respectively for the same data sets. On the contrary, SAND and ADCMiner update the classifier 3 times only in case of Hyperplane data set comparing with fixed 47 number of updates by ECSMiner approach. This indicates that, SAND and ADCMiner avoid unnecessary training during the stable periods and frequently updates the classifier during concept drift.

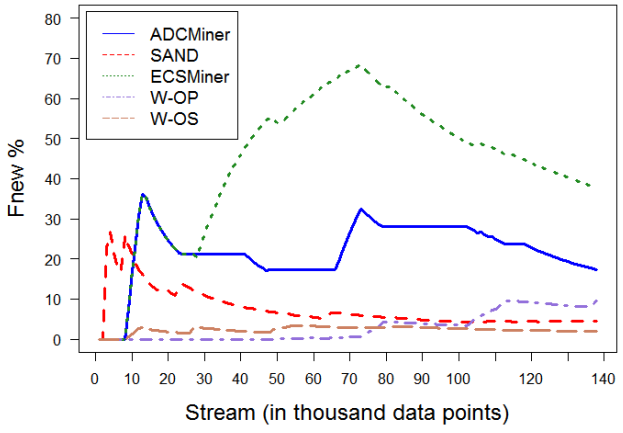
Table 3 summarizes the classification error for each of the techniques on each data set described in Section 4.1. ADCMiner shows the best performance among all the other classifier on most of the data sets. However, it requires labels for all the data instances for maintaining dynamic sliding windows. SAND, on the other hand, does not requires true labels of the data instances for maintaining dynamic sliding



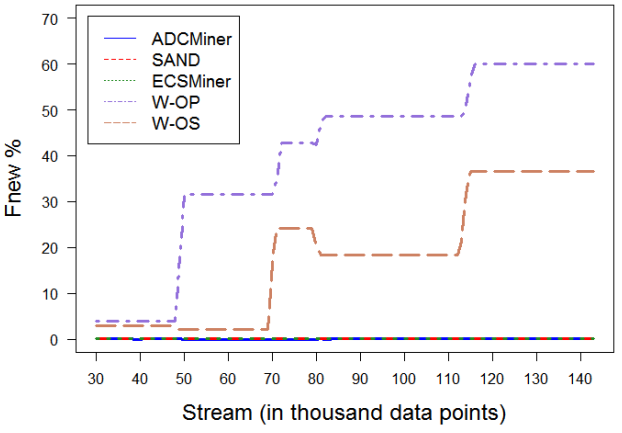
(a) ForestCover Fnew



(b) ForestCover Mnew



(c) PAMAP Fnew



(d) PAMAP Mnew

Fig. 4: Number of Instances v.s. Novel Class Detection Performance of Each Method

window and determining dynamic chunk size. Still, it shows very competitive performance compared with ADCMiner and outperforms other approaches on all the data sets used, except SynCN. Figure 3 shows overall percentage of error of all the approaches compared as the stream progresses on the ForestCover and PAMAP data sets. In the case of the PAMAP data set, ECSMiner, AHT and OBA exhibit 7.03, 1.75 and 1.45 times more error rate than our proposed framework SAND. Similarly, in the case of the ForestCover data set, ECSMiner, AHT and OBA show 1.02, 5.13 and 4.04 times more error rate than SAND.

Only in the case of the SynCN data set does ECSMiner shows slightly better performance than SAND, yet both of the approaches achieve less than 1% error rate. It can be observed that, in the case of the SynCN data set, the other approaches also show comparatively better results than on the rest of the data sets. This indicates that the SynCN data set contains less frequent concept drift compared with other data sets. Since ECSMiner uses a fixed-chunk size, it updates the model more frequently during stable time periods than SAND. From the experiments, we know that SAND updates the ensemble classifier only 30 times compared to the 47 number of updates by ECSMiner. So, ECSMiner gains slightly better accuracy in expense of more frequent training and updating the ensemble. These results indicate that our proposed change point detection algorithm help the framework to detect concept drift efficiently compared to ADWIN which is used in the AHT and OBA approaches.

4.5 Novel Class Detection Performance

Table 4 summarizes the novel class detection performance of SAND and ECSMiner on different data sets. Similar to the classification results, ADCMiner shows the best performance in terms of M_{new}, F_{new} and F_2. However, SAND does not use any labeled instances for determining dynamic chunk sizes, yet shows very competitive results on all the data sets. SAND clearly outperforms ECSMiner in terms of novel class detection on the PAMAP data set. It shows 1.25, 8.28 and 1.71 times better performance than ECSMiner in terms of the M_{new}, F_{new} and F_2 measures. On the contrary, SAND shows very competitive performance compared with ECSMiner in the case of the ForestCover and SynCN data

TABLE 4: Summary of Novel Class Detection Results

Data set	Method	M_{new}	F_{new}	F_2
ForestCover	ADCMiner	**0.0**	**0.225**	**0.998**
	SAND	8.45	1.05	0.83
	ECSMiner	8.42	2.13	0.88
	W-OP	27.206	4.226	0.718
	W-OS	67.727	0.661	0.368
PAMAP	ADCMiner	**0.04**	16.31	**0.96**
	SAND	**0.04**	4.53	0.77
	ECSMiner	0.05	37.53	0.45
	W-OP	60.09	10.86	0.32
	W-OS	36.57	**1.81**	0.64
SynCN	ADCMiner	**0.0**	**0.0**	**1.0**
	SAND	**0.0**	0.01	0.99
	ECSMiner	**0.0**	**0.0**	**1.0**
	W-OP	13.41	1.31	0.86
	W-OS	82.55	0.28	0.20

sets. ECSMiner gains slightly better performance due to more frequent updates as discussed above. Figure 4 shows performance of all the approaches in terms of F_{new} and M_{new} on the ForestCover and PAMAP data sets as the stream progresses. It shows that, ADCMiner and SAND dominate other approaches in novel class detection performance.

So, empirical data show that, both of the proposed approaches are very effective for classifying evolving data streams. Moreover, SAND shows very competitive performance regardless of using no labeled data for dynamic window management.

5 Conclusion

In this paper, we present two complete frameworks, i.e., ADCMiner and SAND which use feedback from the classifier to detect chunk boundaries dynamically. These dynamic chunks are then used to update the existing classifier to adapt to the concept drift or concept evolution problems. ADCMiner uses classifier predictive performance as classifier feedback, which requires labeled data instances. To avoid this, SAND monitors classifier confidence values to determine chunk sizes. We integrate novel class detection capability to the both of the frameworks. We show that, SAND exhibits very competitive performance comparing with ADCMiner, in spite of not using any labeled data instances. In the future, we would like to extend our approach for better semi-supervised learning using active learning technique.

Acknowledgment

Funding for this work was partially supported by the Laboratory Directed Research and Development program at Sandia National Laboratories.

Sandia National Laboratories is a multi-program laboratory managed and operated by Sandia Corporation, a wholly-owned subsidiary of Lockheed Martin Corporation, for the U.S. Department of Energy's National Nuclear Security Administration under contract DE-AC04-94AL85000.

References

[1] C. C. Aggarwal and P. S. Yu. On classification of high-cardinality data streams. In *SDM*, pages 802–813. SIAM, 2010.

[2] I. V. N. Alexander G. Tartakovsky and M. Basseville. *equential Analysis: Hypothesis Testing and Change-Point Detection*. Chapman & Hall/CRC, 2014.

[3] M. Baron. Convergence rates of change-point estimators and tail probabilities of the first-passage-time process. *Canadian J. of Statistics*, 27:183–197, 1999.

[4] M. I. Baron. Nonparametric adaptive change point estimation and on line detection. *Sequential Analysis*, 19(1-2):1–23, 2000.

[5] A. Bifet and R. Gavald. Learning from time-changing data with adaptive windowing. In *SDM*. SIAM, 2007.

[6] A. Bifet, G. Holmes, B. Pfahringer, R. Kirkby, and R. Gavald. New ensemble methods for evolving data streams. In *KDD '09*, pages 139–148, 2009.

[7] A. Bifet, G. Holmes, B. Pfahringer, P. Kranen, H. Kremer, T. Jansen, and T. Seidl. Moa: Massive online analysis, a framework for stream classification and clustering. In *Journal of Machine Learning Research*, pages 44–50, 2010.

[8] D. Brzezinski and J. Stefanowski. Combining block-based and online methods in learning ensembles from concept drifting data streams. *Inf. Sci.*, 265:50–67, May 2014.

[9] D. Cieslak and N. Chawla. Detecting fractures in classifier performance. In *Data Mining, 2007. ICDM 2007. Seventh IEEE International Conference on*, pages 123–132, Oct 2007.

[10] E. Faria, I. Goncalves, J. Gama, and A. Carvalho. Evaluation methodology for multiclass novelty detection algorithms. In *Intelligent Systems (BRACIS), 2013 Brazilian Conference on*, pages 19–25, Oct 2013.

[11] J. Gama, P. Medas, G. Castillo, and P. Rodrigues. Learning with drift detection. In *In SBIA Brazilian Symposium on Artificial Intelligence*, pages 286–295. Springer Verlag, 2004.

[12] J. a. Gama, I. Žliobaitė, A. Bifet, M. Pechenizkiy, and A. Bouchachia. A survey on concept drift adaptation. *ACM Comput. Surv.*, 46(4):44:1–44:37, Mar. 2014.

[13] M. Harel, S. Mannor, R. El-yaniv, and K. Crammer. Concept drift detection through resampling. In *Proceedings of the 31st International Conference on Machine Learning (ICML-14)*, pages 1009–1017. JMLR Workshop and Conference Proceedings, 2014.

[14] M. Z. Hayat and M. R. Hashemi. A dct based approach for detecting novelty and concept drift in data streams. In *SoCPaR*, pages 373–378. IEEE, 2010.

[15] R. Klinkenberg. Learning drifting concepts: Example selection vs. example weighting. *Intell. Data Anal.*, 8(3):281–300, Aug. 2004.

[16] I. Koychev. Gradual forgetting for adaptation to concept drift. In *In Proceedings of ECAI 2000 Workshop Current Issues in Spatio-Temporal Reasoning*, pages 101–106, 2000.

[17] I. Koychev. Tracking changing user interests through prior-learning of context. In *Adaptive Hypermedia and Adaptive Web-Based Systems*, volume 2347 of *Lecture Notes in Computer Science*, pages 223–232. Springer Berlin Heidelberg, 2002.

[18] L. I. Kuncheva and W. J. Faithfull. PCA feature extraction for change detection in multidimensional unlabelled data. *IEEE Transactions on Neural Networks and Learning Systems*, 2013.

[19] M. M. Masud, Q. Chen, J. Gao, L. Khan, J. Han, and B. M. Thuraisingham. Classification and novel class detection of data streams in a dynamic feature space. In *ECML PKDD '10.*, volume II, pages 337–352, 2010.

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

[21] M. M. Masud, J. Gao, L. Khan, J. Han, and B. M. Thuraisingham. Integrating novel class detection with classification for concept-drifting data streams. In *ECML PKDD*, volume II, pages 79–94, 2009.

[22] 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 Trans. Knowl. Data Eng.*, 23(6):859–874, 2011.

[23] MOA. Moa massive online analysis-real time analytics for data streams repository data sets. http://moa.cms.waikato.ac.nz/datasets/, 2015.

[24] G. V. Moustakides. Optimal stopping times for detecting changes in distributions. *Ann. Statist.*, 14(4):1379–1387, 12 1986.

[25] B. Parker and L. Khan. Detecting and tracking concept class drift and emergence in non-stationary fast data streams. In *Twenty-Ninth AAAI Conference on Artificial Intelligence*, Jan 2015.

[26] B. Parker, A. M. Mustafa, and L. Khan. Novel class detection and feature via a tiered ensemble approach for stream mining. In *ICTAI*, pages 1171–1178, 2012.
[27] A. Reiss and D. Stricker. Introducing a new benchmarked dataset for activity monitoring. In *ISWC*, pages 108–109. IEEE, 2012.
[28] Sensor. Stream data mining repository. http://www.cse.fau.edu/~xqzhu/stream.html, 2015.
[29] X. Song, M. Wu, C. Jermaine, and S. Ranka. Statistical change detection for multi-dimensional data. In *13th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 667–676, NY, USA, 2007. ACM.
[30] E. J. Spinosa, A. P. de Leon, and J. . Gama. Novelty detection with application to data streams. *Intell. Data Anal.*, 13:405–422, Aug. 2009.
[31] 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 *ACM SAC*, pages 976–980, 2008.
[32] V. N. Vapnik. *Statistical Learning Theory*. Wiley-Interscience, 1998.
[33] H. Wang, W. Fan, P. S. Yu, and J. Han. Mining concept-drifting data streams using ensemble classifiers. In *Ninth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 226–235. ACM, 2003.

Ahsanul Haque received the BS degree in computer science and Engineering from the Bangladesh University of Engineering and Technology (BUET) in 2011. He is currently working towards the PhD degree in the Department of Computer Science, University of Texas at Dallas. His research interests are in data stream mining, machine learning, and big data analytic. He has published more than 7 research papers in selective peer review conferences including PAKDD, IEEE BigData, IEEE CloudCom, and IEEE Cloud. He is a student member of the IEEE.

Latifur Khan received the BSc degree in computer science and engineering from the Bangladesh University of Engineering and Technology, Dhaka, Bangladesh in 1993, and the MS and PhD degrees in computer science from the University of Southern California in 1996 and 2000, respectively. He is currently an associate professor in the Computer Science Department at the University of Texas at Dallas (UTD), where he has been teaching and conducting research since Sept. 2000. In addition, he 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 the University of Texas at Dallas. His 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. He has published more than 130 papers in journals and conferences. He is a senior member of the IEEE.

Michael Baron is Professor of Statistics at the University of Texas at Dallas. Supported by grants from the National Science Foundation, the National Security Agency, the Actuarial Foundation, and the Semiconductor Technical Council, he conducts research in the areas of sequential analysis, change-point detection, and Bayesian inference, applying obtained results in epidemiology, clinical trials, cyber security, energy finance, and semiconductor manufacturing.

M. Baron has published two books, a number of research articles and book chapters, and gave an even greater number of invited presentations and seminars. In 2007 M. Baron received Abraham Wald prize for the best paper in sequential analysis, and in 2013 he was elected a Fellow of the American Statistical Association. Currently, he serves as Associate Editor of the Journal of Sequential Analysis. He is a member of the American Statistical Association and the International Society for Bayesian Analysis.

Joe Ingram is a Principal Member of the Technical Staff at Sandia National Laboratories. He joined Sandia in 2010 after completing a master's degree in computer science at Southern Illinois University. Joe completed his B.S. in computer science in 2007 at the University of Illinois at Springfield. He has contributed to various research projects focused on applying machine learning to challenging problems in cybersecurity.