Efficient Handling of Concept Drift and Concept Evolution over Stream Data

Ahsanul Haque*, Latifur Khan*, Michael Baron†, Bhavani Thuraisingham*, and Charu Aggarwal‡
*Department of Computer Science, The University of Texas at Dallas, Richardson, Texas 75080
Email: {ahsanul.haque, lkhan, bhavani.thuraisingham}@utdallas.edu
†Department of Mathematical Sciences, The University of Texas at Dallas, Richardson, Texas 75080
Email: mbaron@utdallas.edu
‡IBM T. J. Watson Research Center, Yorktown NY, USA
Email: charu@us.ibm.com

***Abstract*—To decide if an update to a data stream classifier is necessary, existing sliding window based techniques monitor classifier performance on recent instances. If there is a significant change in classifier performance, these approaches determine a chunk boundary, and update the classifier. However, monitoring classifier performance is costly due to scarcity of labeled data. In our previous work, we presented a semi-supervised framework SAND, which uses change detection on classifier confidence to detect a concept drift. Unlike most approaches, it requires only a limited amount of labeled data to detect chunk boundaries and to update the classifier. However, SAND is expensive in terms of execution time due to exhaustive invocation of the change detection module. In this paper, we present an efficient framework, which is based on the same principle as SAND, but exploits dynamic programming and executes the change detection module selectively. Moreover, we provide theoretical justification of the confidence calculation, and show effect of a concept drift on subsequent confidence scores. Experiment results show efficiency of the proposed framework in terms of both accuracy and execution time.**

***Keywords*—Classifier Confidence; Dynamic Chunk; Concept Drift**

I. Introduction

In today's connected digital world, enormous amount of data are being generated in form of data streams from a variety of sources - social networks, online businesses, sensors, military surveillance to name a few. This is why, necessity of having a robust and fast technique for classifying data steams is becoming increasingly important. However, data stream classification is a challenging task due to its inherent properties, e.g., infinite length, concept drift, concept evolution, limited labeled data, delayed labeling, etc. [1].

Due to the infinite length problem, data streams cannot be stored into main memory for analyzing, e.g., labeling. Existing techniques either divide the stream into fixed-sized chunks (e.g. [2]), or use gradual forgetting (e.g. [3]) to address the problem of infinite length. However, it requires *a priori* knowledge about the time-scale of change to find the correct chunk size or forgetting rate which is not available most of the time. So, these techniques suffer from a trade-off between performance during stable periods due to redundant retraining, or delayed response to a sudden drift of concept. Existing dynamic sliding window based approaches (e.g. [4]) determine chunk boundaries by tracking any major change in error rate of the classifier, but requires true labels of all data instances.

Concept drift occurs in a stream when the underlying concept of data changes over time. So, the classification model needs to be updated regularly to adapt to the most recent concept [2]. The vast majority of concept drift researches are supervised in nature, and assume that true labels of test instances will be readily available to update the classifier as soon as they are tested. However, typically labeling requires annotation by a human expert, which is a costly and time consuming process. More often, true labels are available only for a limited amount of data. Therefore, supervised approaches suffer in this scenario [5].

Another major challenge in classifying data streams is emergence of a totally new class, also known as concept evolution. In real-world data streams, such as intrusion detection, text classification or fraud detection, the number of classes is not fixed, and concept evolution may occur at any time in the stream. If concept evolution is not addressed timely, classifier might misclassify the instances from the new class as existing classes, thereby the classification error increases. However, this problem has been ignored by most of state-of-the-art techniques.

In our previous work, we proposed a semi-supervised sliding window based framework SAND [6] to address all of the above challenges. It maintains an ensemble of classifier models, each trained on a dynamically determined chunk. However, unlike other existing sliding window techniques, it does not depend on error rate of the classifier, which requires true labels for all data instances. Rather, it estimates classifier confidence in classifying each test instance, stores in the sliding window, and tracks any significant change in confidence estimates using a change detection technique (CDT). A change in the classifier confidence indicates occurrence of a concept drift. Therefore, if there is a significant change in confidence estimates, a concept drift is detected, a chunk boundary is determined, and the classifier is updated using a few labeled instances from the latest chunk. Experiment results suggest that SAND achieves good classification performance despite using a limited amount of labeled data. However, it is computationally expensive due to exhaustive invocation of the change detection module.

In this paper, we present an efficient framework based on the principle of SAND. It has an ensemble classifier consisting of k-NN type clustering based models. We propose two estimators, namely *Association* and *Purity* to estimate confidence of the classifier. These estimators are generic as these can be used to estimate confidence of any clustering based model. We theoretically justify the use of these estimators, and show that confidence estimates decrease consistently in presence of a concept drift. SAND invokes the change detection module after estimating classifier confidence on each test instance. A change detection score is calculated each time based on values currently stored in the sliding window. If this score is greater than a pre-fixed threshold, a concept drift is detected. This exhaustive invocation of the change detection module makes SAND computationally expensive. In this paper, we propose a recursive formula, and use dynamic programming to calculate change detection scores. Along with this, we propose selective and sporadic execution of the change detection module based on confidence score on the most recent test instance. These greatly reduces the execution time of our proposed framework, and makes it practically usable.

Our proposed framework uses only a limited amount of labeled data instances from the latest chunk for updating the classifier. Motivated by the principle of Uncertainty Sampling [7], it selects data instances for labeling based on the classifier confidence score calculated during testing. If the confidence in classifying an instance is low, true label for that instance is requested. Otherwise, the label predicted by the classifier is accepted as the label of that instance. Thus, training data is formed only using a partially labeled data. A new model is then trained on the training data to replace the oldest model in the ensemble. Thus, the ensemble is updated using a limited amount of labeled data without any extra overhead by simply using the confidence scores calculated for dynamic sliding window management.

Our framework also incorporates a novel class detector for the purpose of handling concept evolution. The detector assumes appearance of a single novel class at a time in the data stream. Any instance falling outside the decision boundaries of all models in the ensemble is identified as an outlier. The framework interprets the presence of a sufficiently large number of outliers with strong cohesion among themselves as emergence of a novel class.

Primary contributions of our work are as follows: 1) We present a technique to estimate confidence of any clustering based classifier with theoretical justification. 2) We analytically show that classifier confidence decreases consistently in presence of a concept drift. Therefore, we design a suitable change detection technique (CDT) to detect any significant change in classifier confidence. 3) We use the confidence scores to select a limited number of data instances from the latest chunk for labeling. Following detection of a concept drift, the classifier is updated using this limited amount of labeled data instances without any extra overhead. 4) We present a semi-supervised framework, which uses dynamically determined chunks both for classification and novel class detection. We use dynamic programming, and propose sporadic and selective execution of the change detection module to reduce execution time of the proposed framework to a great extent. 5) We evaluate our proposed framework on several benchmark and synthetic data sets. Experiment results indicate that our framework shows good performance in terms of both classification accuracy and execution time.

The rest of the paper is organized as follows: In Section II, we present a brief literature survey related to our work. Section III describes our proposed framework in detail. Section IV discusses about proposed performance improvement of the framework. We describe the data sets, evaluation metrics and present experiment results in Section V. Finally, Section VI concludes the paper.

II. Related Work

Approaches that divide data streams into fixed-size chunks, e.g., [2] cannot capture concept drift immediately if the chunk size is too large, or suffer from unnecessary frequent training during stable time period if the chunk size is too small [4]. *Gradual forgetting* is also used in the literature, e.g., [3] to address the infinite length problem of data streams. However, finding the perfect decay function for mining an evolving data stream is a challenge. In this paper, we use an explicit change detection technique (CDT) to detect any change of concept, and to determine the chunk size dynamically.

In data stream mining, CDT is used either to detect a change in the input data distribution, or to detect a change in the classifier feedback. Several methods, e.g., [8]–[10] exist to detect change of the input data distribution in a data stream. However, detecting change in a multi-dimensional space is a hard problem [11]. It introduces error while finding changes in the multi-dimensional input space, hence not efficient in the context of data stream mining. In this paper, we focus on detecting change in one dimensional classifier confidence.

Various CDTs have been proposed in the literature to detect concept drift from any significant change in the classifier feedback. *Adwin* [4] is a sliding window based technique which determines the size of the window according to the rate of change observed from the window data itself. [12] detects a change when the error rate over the whole current window significantly exceeds the lowest error rate recorded. [13] exploits *Kruskal Wallis* analysis and *Kolmogorov Smirnov* tests to detect changes. [11] is based on obtaining statistics from the loss distribution of the learning algorithm by reusing the data multiple times via re-sampling. Concept drift detection in [14] contains two CDTs based on Intersection of confidence intervals (ICI), one to detect change in the input data distribution and another to detect change in the classifier error rate. Considering the high volume and speed of today's data streams, running two CDTs after testing each instance is expensive. Another ICI based approach is ACE [15]. All of the above CDTs detect change in the classifier error rate, requiring true labels of all data instances be readily available. This assumption is not practical in the context of data streams [5]. Our proposed CDT detects changes in classifier confidence which does not require any supervised information.

Existing semi-supervised approaches to classifying data streams use active mining, computational geometry etc. to select important data instances for labeling. For example, complex active learning is used in [16], [17] to select data instances for which true labels are requested. Computational geometry based approach COMPOSE [18] can only address gradual (limited) drift, rather than abrupt drift. Our proposed approach reuses the same confidence scores calculated during prediction to select instances for labeling without adding any extra overhead.

Clustering based *single class* novelty detection methods have been proposed in [19], [20], which are not suitable for multi-class environment. [21] proposes an approach for multi-class novelty detection but uses fixed-size chunks, hence suffers from the trade-off discussed earlier. Unlike these approaches, our proposed approach uses dynamic chunks for multi-class novelty detection. [22] presents an evaluation methodology for multi-class novelty detection algorithms. In this work, we assume that only one novel class may appear at a time in the data stream. So, we use traditional novelty detection evaluation metrics that have been used by most of state-of-the-art approaches.

III. Proposed Approach

We proposed SAND in [6], which is a semi-supervised framework for classifying evolving data streams. It maintains an ensemble classifier consisting of k-NN type models for classification. As soon as a test instance arrives, SAND classifies the instance along with calculating a score indicating confidence of the classifier in this classification. These confidence scores are stored in a sliding window. Next, SAND searches for any change in the distribution of confidence scores. A significant change in confidence scores indicates occurrence of a concept drift. Therefore, SAND determines a chunk boundary immediately and updates the classifier. SAND [6] shows good classification performance despite using only a limited amount of labeled data. However, it is not so efficient in terms of execution time due to invocation of the costly *change detection* module after calculating each confidence score.

In this paper, we propose an efficient semi-supervised framework based on the principle of SAND. We provide theoretical justification of the confidence estimators, and analyze evolution of confidence scores in presence of any concept drift. In addition, we reduce the execution time to a great extent by invoking the change detection module selectively. This framework henceforth will be referred to as "ECHO" (Efficent Concept Drift and Concept Evolution Handling over Stream Data).

In order to make this paper self-contained, we briefly discuss the classification and novel class detection of ECHO, which is similar to SAND. ECHO maintains an ensemble $\mathcal{M}$ of t classification models. At the beginning, the models are built on the initial training data, i.e., warm-up period instances. Once the warm-up period is over, each of the incoming data instances is tested by the ensemble classifier. If any instance falls outside of the decision boundary of $\mathcal{M}$, it is considered as an outlier. The outlier instances are temporarily stored in a buffer. When there are enough instances in the buffer, *novel class detection* module is invoked. A class is defined as *novel* class if none of the models in the ensemble has been trained with any instance from that class. ECHO detects emergence of a novel class if the outliers are close from each other, and far away from the existing class instances. If a novel class is detected, the instances in the outliers are tagged accordingly. Otherwise, these instances are considered from the existing classes, and classified using the ensemble classifier.

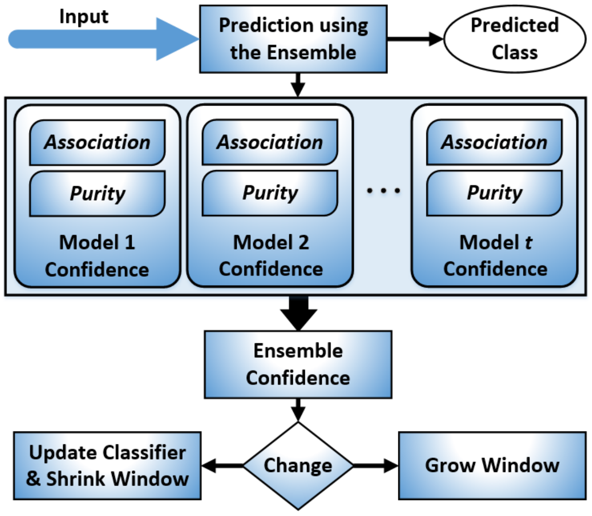


Fig. 1: Sliding window management of ECHO

Sliding window management of ECHO is depicted in Figure 1. If a test instance is inside the decision boundary of the ensemble classifier, or is not detected as instance from a novel class, ECHO classifies it by taking the majority vote from the ensemble. At the same time, it estimates confidence of each model in this classification using a few estimators. These model confidences are combined together to calculate the overall confidence of the ensemble classifier. ECHO maintains a dynamic sliding window W for storing classifier confidence estimates calculated on recent test instances. Following insertion of confidence estimates, ECHO selectively invokes the *change detection* module to search for any significant change of distribution among the scores saved in W. If there is such a change, ECHO detects a concept drift, and W is updated. Moreover, a few instances are selected based on the confidence scores for which the actual labels are requested. Finally, the ensemble classifier is updated following a concept drift by training a new model on limited amount of labeled data, and by replacing the oldest model in the ensemble by it. Based on the memory resource available, we set a maximum allowable size for W denoted by V_m. If W grows beyond V_m, a chunk boundary is determined, and both $\mathcal{M}$ and W are updated. Later in this section, we discuss about classification, novel class detection, and change detection modules. Table I contains a list of frequently used symbols.

A. Training and Classification

Each model in the ensemble, $M_i, \quad i \in 1 \ldots t$, is a k-NN type model. However, unlike k-NN, raw data points are not stored in the model. Rather, a number of clusters are built using any clustering algorithm, e.g., K-means, DBSCAN [23] etc. We use an impurity based K-means algorithm (discussed in Section III-D) to build the clusters in our experiments. We assume that only a portion of data instances will be labeled. Once the clusters are created, the raw data points are discarded after summaries (mentioned as *pseudopoints*) of the clusters are saved. Therefore, each model is a collection of K pseudopoints. Summary of a cluster, i.e., a pseudopoint contains the centroid, radius, and number of data points belonging to each

TABLE I: Commonly used symbols and terms

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

of the classes (referred to as frequencies). The radius is equal to the distance between the centroid and the farthest data point in the corresponding cluster.

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 in M_i. The decision boundary of the ensemble $\mathcal{M}$ is the union of decision boundaries of all models $M_i \in \mathcal{M}$.

If a test instance x is inside the decision boundary of $\mathcal{M}$, it is classified using each $M_i \in \mathcal{M}$, $i \in 1 \dots t$ as follows. Let $h \in M_i$ be the pseudopoint whose centroid is the nearest from 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 $\mathcal{M}$ by taking the majority vote among all classifiers.

B. Novel Class Detection

If a test instance x falls outside of the decision boundary of $\mathcal{M}$, it is declared as a filtered outlier, or *F-outlier*. The principle of the novel class detection is that an instance should be closer to instances from the same class (cohesion), and farther apart from instances from other classes (separation). Since any *F-outlier* falls outside of the decision boundary, it is far away from the existing class instances. So, it satisfies the separation property. If it satisfies the cohesion property also, i.e., it is close to other F-outlier instances, ECHO declares emergence of a novel class. To examine this, such instances are stored in a buffer. This buffer is periodically examined by the *novel class detection* module to observe whether there are enough instances in the buffer that are close to each other. This is done by computing the q-Neighborhood Silhouette Coefficient, or q-NSC [21]. This is defined based on q, c-neighborhood of an *F-outlier* x ($q, c(x)$ in short), which is the set of q instances from class c that are nearest to x. Here q is a user defined parameter.

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 [21] of x is given by:

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

q-NSC considers both cohesion and separation, and yields a value between -1 to +1. A positive value of q-NSC indicates that the *F-outliers* are closer to other *F-outlier* instances stored in the buffer (more cohesion), and farther away from the instances from existing classes (more separation). The q-NSC(x) value of an *F-outlier* x must be computed separately for each classifier $M_i \in \mathcal{M}$. ECHO declares emergence of a novel class if it finds at least $q' > q$ *F-outlier*s having a positive q-NSC score for all the classifiers $M_i \in \mathcal{M}$.

C. Calculation of Confidence Scores

Similar to SAND [6], ECHO employs two heuristics, i.e., *association* and *purity* to estimate confidence of each individual model $M_i \in \mathcal{M}$ in classifying any instance x. These individual model confidences are then combined together to calculate confidence of the entire ensemble classifier. Let h_{ip} be the p^{th} pseudopoint in M_i, and c_m be the class having highest frequency in h_{ip}. Assuming the closest pseudopoint from x in model M_i is h_{ip}, the heuristics are calculated as follows:

- *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}. Therefore, smaller the $D_{ip}(x)$, higher the *association*.
- *Purity* is calculated by $\frac{|\mathcal{L}_{ip}(c_m)|}{|\mathcal{L}_{ip}|}$, where $|\mathcal{L}_{ip}|$ is the sum of all *frequencies* in h_{ip}, and $|\mathcal{L}_{ip}(c_m)|$ is the *frequency* of c_m in h_{ip}.

Association and *purity* of the model M_i are denoted by $\mathcal{A}_i$ and $\mathcal{P}_i$ respectively. We theoretically justify use of these heuristics in Section III-D. Both of the heuristics contribute to model confidence according to their estimation capability. This capability is evaluated by the correlation between heuristic values and classification accuracy using the initial training data as follows. Heuristic values for M_i are calculated for each of the 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, a 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. Once the correlation coefficients, i.e., estimation capabilities are evaluated, it is retained for calculating model confidence in classifying future test instances.

To calculate the confidence of M_i in classifying a test instance x (denoted by $\mathcal{C}_i^x$), ECHO first calculates heuristic values $\mathcal{H}_i^x$. Next, $\mathcal{C}_i^x$ is calculated by taking the dot product of $\mathcal{H}_i^x$ and r_i, i.e., $\mathcal{C}_i^x = \mathcal{H}_i^x.r_i$. Similarly, ECHO calculates confidence scores for each of the models in the ensemble. These scores are then normalized between 0 and 1. Finally, ECHO takes the average confidence of the models towards the predicted class to estimate confidence of the entire ensemble $\mathcal{C}^x$.

D. Justification of Confidence Estimators

In this Section, we first define the objective function for semi-supervised K-means clustering that is used to build models in ECHO. Then, based on the objective function, we theoretically justify the choice of the heuristics for estimating classifier confidence.

1) 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, and at the same time by 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, μ_i is the centroid 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 becomes:

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

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.

Next, we justify the use of *association* and *purity* as confidence estimators.

2) Association: Consider two cases: a) 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:

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

If x falls inside h_{ip} but outside of h_{jq}, h_{ip} has a greater *association* value than h_{jq}. Therefore, if other properties remain same, 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}, $||\mu_{jq} - x||^2 > ||\mu_{ip} - x||^2 \Rightarrow \Delta Disp_i^x < \Delta Disp_j^x$, where μ_{ip} and μ_{jq} are centroids of h_{ip} and h_{jq} respectively. 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.

b) 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 properties (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:

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

Therefore, in this case also, 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) 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:

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

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

b) 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

$$p^i_{c_m} = \frac{|\mathcal{L}_{ip}(c_m)|}{|\mathcal{L}_{ip}|} \; ; \; p^j_{c_n} = \frac{|\mathcal{L}_{jq}(c_n)|}{|\mathcal{L}_{jq}|}$$

So,

$$p^i_{c_m} > p^j_{c_n} \Rightarrow \Delta Ent^x_{ip} < \Delta Ent^x_{jq} \qquad (7)$$

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.

E. Effect of Concept Drift on Classifier Confidence

In this Section, we show that classifier confidence decreases 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 be the SSE of pseudopoint h_{ip}, μ_{ip} be the centroid of h_{ip}, and x be an arbitrary data point. Therefore, the SSE_i of a classification model is defined as follows:

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

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:

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

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:

$$\begin{aligned}
SSE'_i &= \sum_p n_{ip} D'_{ip} = \sum_p n_{ip} (\bar{D}_{ip} + \delta_{ip}) \\
&= \sum_p n_{ip} \delta_{ip} + \sum_p n_{ip} \bar{D}_{ip} \qquad (10) \\
&= \sum_p n_{ip} \delta_{ip} + SSE_i > SSE_i \qquad (11)
\end{aligned}$$

Algorithm 1 Detect-Change (α, γ, W)

```
Input: $\alpha$: Sensitivity,
       $\gamma$: Cushion period size,
       $W$: The dynamic sliding window.
Output: The change point if exists; -1 otherwise
 1: $T_h \leftarrow -log(\alpha)$, $n \leftarrow$size of $W$, and $\omega_n \leftarrow 0$.
 2: if $n \leq V_m$ & $mean(X_1 \dots X_n) > 0.3$ then
 3:     for $k \leftarrow \gamma : n - \gamma$ do
 4:         Estimate pre and post-beta distributions, beta($\hat{\alpha_0}, \hat{\beta_0}$) and
            beta($\hat{\alpha_1}, \hat{\beta_1}$) from $X_1 \dots X_k$ and $X_{k+1} \dots X_n$ respectively.
 5:         Calculate $S_{k,n}$ using Equation (12).
 6:     end for
 7:     Calculate $\omega_n$ using Equation (13).
 8:     if $\omega_n \geq T_h$ then
 9:         Return $kmax$, where $S_{kmax} = \omega_n$.
10:     else
11:         Return -1.
12:     end if
13: else
14:     Return $n$.
15: end if
```

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.

F. Change Detection

ECHO maintains a variable size window W to monitor confidence of the classifier on recent data instances. Confidence scores are generated in the range of $[0, 1]$ (discussed in Section III-C). We observe from our experiments on various data sets that generated confidence scores tend to follow a *beta* distribution. We have carried out *Chi-Square* goodness of fit test on the generated confidence scores which also confirm that observation. Moreover, we have shown in Section III-E that confidence scores decrease consistently in presence of a concept drift. Therefore, we propose a CUSUM [24]-type change detection technique (CDT) on beta distribution to use in this context. After inserting each confidence score, our CDT is used to detect any significant change of statistical properties within the values stored in W.

Algorithm 1 sketches the proposed CDT. Let n be the size of W, if at any time overall mean confidence of the classifier falls below 0.3, or the window size exceeds the maximum allowable size, the proposed CDT returns n as the change point. Otherwise, first it divides W into two sub-windows for each k between γ to $n - \gamma$. We know that each of the sub-windows contain a beta distribution, however the parameters are unknown. So, $X_1 \dots X_k$ and $X_{k+1} \dots X_n$ constitute pre and post-beta distributions respectively, where X_i is the i^{th} element in W. We estimate the parameters using the method of moments. So, each sub-window should contain at least γ number of values to preserve statistical properties of a distribution. In the literature, $\gamma = 100$ is widely used, which is also called the *cushion* period. The proposed CDT estimates the parameters at Line 4. Let $(\hat{\alpha_0}, \hat{\beta_0})$ and $(\hat{\alpha_1}, \hat{\beta_1})$ are the estimated parameters for pre and post-beta distributions respectively. Then, sum of the log likelihood ratios is calculated at Line 5 using the following formula:

$$S_{k,n} = \sum_{i=k+1}^{N} log \left(\frac{f\left(X_i \mid \hat{\alpha}_1, \hat{\beta}_1\right)}{f\left(X_i \mid \hat{\alpha}_0, \hat{\beta}_0\right)} \right) \tag{12}$$

Where $f\left(X_i | \hat{\alpha}, \hat{\beta}\right)$ is the probability density function (PDF) of the beta distribution having parameters $\left(\hat{\alpha}, \hat{\beta}\right)$ applied on X_i. Next, the cusum process score ω_n for the values stored in W is calculated at Line 7 using the following formula:

$$\omega_n = \max_{\gamma \le k \le n-\gamma} S_{k,n} \tag{13}$$

Let $kmax$ is the value of k for which the algorithm calculated the maximum $S_{k,n}$ value where $\gamma \le k \le n - \gamma$. Finally, a change is detected at point $kmax$ if ω_n is greater than a pre-fixed threshold. We fix the threshold based on the value of the desired sensitivity α. In our experiments, we use $-log(\alpha)$ as the threshold value. We use $\alpha = 0.05$, which is also widely used in the literature.

Algorithm 2 UpdateClassifier ($\mathcal{M}$, W, τ, cp)

```
Input: $\mathcal{M}$: The current ensemble classifier,
       $W$: The dynamic sliding window,
       $\tau$: Classifier confidence threshold,
       $cp$: Estimated change point.
Output: $\mathcal{M}_u$: The updated ensemble classifier
1: $\mathcal{L} \leftarrow \{<x, y>: \mathcal{C}^x \in W \ and \ \mathcal{C}^x \le \tau\}$ // $y$ is the true label of $x$
2: $\mathcal{U} \leftarrow \{<x, \hat{y}>: \mathcal{C}^x \in W \ and \ \mathcal{C}^x > \tau\}$ // $\hat{y}$ is the predicted label of $x$
3: $\mathcal{T} \leftarrow \mathcal{L} \cup \mathcal{U}$ // $\mathcal{T}$ is the training set
4: $M' \leftarrow$ TrainNewModel $(\mathcal{T})$
5: $\mathcal{M}_u \leftarrow$ Update $(\mathcal{M}, M')$
6: $\mathcal{T} \leftarrow \emptyset$
7: $W \leftarrow [X_{cp+1} \dots X_n]$
```

G. Updating the Ensemble using Limited Labeled Data

A chunk boundary is detected as soon as a significant change is detected in the distribution of confidence scores. Subsequently, the current ensemble classifier is updated using the instances from that chunk as shown in Algorithm 2. However, instead of requiring true labels of all instances, ECHO intelligently selects a few instances for labeling using the classifier confidence scores. Since confidence scores are calculated using *association* and *purity* of the models, it provide useful insight to select important instances for updating the classifier. If the confidence in classifying an instance is below a threshold τ, ECHO requests for its true label and include in the labeled instance set (Line 1). On the contrary, if the confidence is above τ, ECHO uses the predicted label and includes the instance in the unlabeled instance set (Line 2). Labeled and unlabeled set of instances together form the new training set to train a new model (Line 4) as discussed in Section III-A.

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 as 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. Finally, W is updated so that it contains only $X_{cp+1} \dots X_n$, where cp is the change point.

H. Time and Space Complexity

ECHO has four modules, i.e., *Classification*, *Change Detection*, *Novel Class Detection*, and *Update*. Time complexity of invoking the *Novel Class Detection* module once is $O(KV_m)$, 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 $O(KtV_m + tV_m + mKV_m)$, where $m = V_m/q$. Since $m >> Kt$, the total time complexity for the classification is $O(mKV_m)$. The time complexity for invoking *Change Detection* module for a whole chunk is $O(V_m^3)$. So, the overall time complexity for executing ECHO on a chunk of data is $O(mKV_m + V_m^3 + f(V_m))$, where $f(V_m)$ is the time to train a new classifier with V_m training instances. Most often, $V_m >> m$ and $V_m >> K$, so the total time complexity is essentially $O(V_m^3 + f(V_m))$. 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 V_m instances, where V_m is the maximum allowable size for the dynamic sliding window. So, space complexity of our framework is $O(V_m)$.

IV. Performance Improvement

Time complexity of the *change detection* module of ECHO is $O(V_m^3)$ for one chunk of data. Therefore, change detection module is the bottleneck for ECHO. In this section, we propose strategies to reduce execution time of the change detection module.

A. Sporadic Execution

As shown in Equation 12 and 13, we calculate $S_{k,n}$ exhaustively for each k, $\gamma \le k \le n-\gamma$, to get cusum score ω_n on n observations. As stated before, confidence scores are expected to decrease when a concept drift occurs. Hence, we are interested about change points only in the negative direction, i.e., change of distribution in decreasing confidence scores. We use this fact to avoid invoking computationally heavy cusum-type CDT after inserting each confidence score. Rather, we use a computationally inexpensive secondary algorithm to seek at least a certain amount of decrease in the population mean of confidence scores. Since the secondary algorithm can only detect a mere mean shift, it is assumed to produce frequent false alarms. Therefore, when it signals such a change, it is considered as a preliminary detection only. It will be then verified and confirmed by a more accurate but computationally heavier cusum-type CDT. In this paper, we form the inexpensive secondary algorithm simply by comparing mean confidence scores of the sub-windows. It produces a warning and therefore proposed cusum-type CDT is invoked only if α times decrease is observed between the mean scores of the sub-windows, where α is the sensitivity parameter introduced in Section III-F.

B. Recursive Calculation

To calculate the cusum score ω_n, we need to calculate $S_{k,n}$ for each k, $\gamma \le k \le n-\gamma$. Hence, to develop the recursion, we divide the original problems into sub-problems based on the value of k and n. We propose the following recursion to calculate $S_{k,n}$:

$$S_{k,n} = \begin{cases} 0, & \text{if } 0 \le n \le 2\gamma \\ \sum_{i=k+1}^{n} \log \frac{f(x_i|\hat{\alpha}_1,\hat{\beta}_1)}{f(x_i|\hat{\alpha}_0,\hat{\beta}_0)}, & \\ & \text{if for all } m \le n,\ I(k,m)=0 \\ S_{k,m} + \sum_{i=m+1}^{n} \log \frac{f(x_i|\hat{\alpha}_1,\hat{\beta}_1)}{f(x_i|\hat{\alpha}_0,\hat{\beta}_0)}, & \text{otherwise} \\ & \text{where } m = \max\{g : g \le n \ \&\ I(k,g)=1\} \end{cases} \tag{14}$$

Let current size of W be n. Since each of the sub-windows are required to contain at least γ number of observations, $S_{k,n}=0$ for all $n \le 2\gamma$. After receiving each test instance, ECHO predicts the label of the instance and inserts the confidence score into W. Since we only compute $S_{k,n}$ when a secondary algorithm gives a warning, for each $m \le n$ we use $I(k,m)$ as an indicator function to indicate whether $S_{k,m}$ was computed or not. If there is no such $m \le n$ for which $S_{k,m}$ was computed, we calculate $S_{k,n}$ using the original formula shown in Equation 12. These first two cases constitute the base of our recursive formula.

The third case applies if we have at least one $m \le n$ for which $S_{k,m}$ was calculated before, i.e., $I(k,m)=1$. If there exists multiple m like this, we find the largest m and reuse $S_{k,m}$ to calculate $S_{k,n}$ recursively. Parameters for pre and post-beta distributions are estimated as before from $X_1 \dots X_k$ and $X_{k+1} \dots X_n$ respectively. However, in this case, we only calculate log-likelihood ratios for $X_{m+1} \dots X_n$ and add with $S_{k,m}$ to get $S_{k,n}$. Since we do not adjust for log-likelihood ratios for $X_{k+1} \dots X_m$, it introduces a loss in the estimation. On the other hand, it can be shown that mere calculating $S_{k,n}$ recursively reduces the time complexity of invoking change detection module for one chunk to $O(V_m^2)$. Experiment result also suggest that the above recursive calculation reduces execution time significantly while keeping very competitive accuracy.

Algorithm 3 Detect-Change-Revised (α, γ, W, $Stat$)

```
Input: $\alpha$: Sensitivity
       $\gamma$: Cushion period size
       $W$: The dynamic sliding window
       $Ind$: Data structure containing largest $m$ for which $I(k,m)$=1.
       $Stat$: Data structure containing previous cusum calculations.
Output: The change point if exists; -1 otherwise
1: $T_h \leftarrow -log(\alpha)$, $n \leftarrow$size of $W$, and $\omega_n \leftarrow 0$.
2: if $n \le V_m$ & $mean(X_1 \dots X_n) > 0.3$ then
3:     for $k \leftarrow \gamma : n-\gamma$ do
4:         if Secondary algorithm (Section IV-A) produces a warning then
5:             Retrieve pre-beta distribution beta$(\hat{\alpha_0},\hat{\beta_0})$ from $Stat[m]$
               if $Ind[k]=m$: estimate the parameters explicitly if there
               is no such $m$.
6:             Estimate post beta distribution beta$(\hat{\alpha_1},\hat{\beta_1})$ from
               $X_{k+1} \dots X_n$.
7:             Calculate $S_{k,n}$ using Equation (14).
8:         end if
9:     end for
10:    Calculate $\omega_n$ using Equation (13).
11:    Insert pre-beta distribution and $S_{k,n}$ at $State[n]$, and update
       $Ind[k] \leftarrow n$.
12:    if $\omega_n \ge T_h$ then
13:        Return $kmax$, where $S_{kmax} = \omega_n$.
14:    else
15:        Return -1.
16:    end if
17: else
18:    Return $n$.
19: end if
```

Algorithm 3 sketches the revised change detection algorithm using the above stated secondary inexpensive algorithm and recursion. We maintain two data structures *Ind* and *Stat*. *Ind* is used to implement the indicator function used in Equation 14. It returns the largest m for which $S_{k,m}$ has already been computed and -1 if no such m exists. On the other hand, *Stat* is used for memoization of previous cusum scores and distributions.

The algorithm starts with calculating the threshold and size of W. If there is a warning from the secondary algorithm, we use the proposed cusum-type CDT to verify it (Line 4). Pre and post-beta distributions are estimated explicitly if *Ind(k)* returns -1, meaning there is no $m \leq n$ for which $S_{k,m}$ was calculated before. Otherwise, only post-beta distribution is estimated explicitly, and pre-beta distribution is retrieved from *Stat* at Line 5. Next, $S_{k,n}$ is calculated using the Equation 14, and ω_n is calculated using Equation 13. *Ind* and *Stat* are updated using n and cusum calculations respectively. Finally, a change is detected if $\omega_n \geq T_h$.

C. Selective Execution

So far in Section IV-A and IV-B, we have focused on efficiently detecting a change point once the change detection module is invoked. To further reduce time complexity of ECHO, we focus on invoking the change detection module itself selectively instead of executing after inserting each confidence estimate. Since we intend to detect significant decrease of confidence over a period of time, confidence scores calculated on the latest test instance could be useful to decide whether the change detection needs to be executed or not. We can skip execution of the change detection module if the latest confidence score is high. On the contrary, we must execute the change detection if the latest confidence score is low. We examine the following two strategies based on this principle for selective execution of the change detection module:

1) First strategy is to use a pre-fixed sampling threshold on the classifier confidence to decide whether the change detection module will be invoked or not. We use the classifier confidence threshold τ (introduced in Section III-G) as the sampling threshold also. If the classifier confidence on test instance x, i.e., $\mathcal{C}^x$ is below τ, change detection module is invoked and vice versa. This version will be referred to as ECHO-F.

2) According to second strategy, The probability of executing the change detection module after inserting $\mathcal{C}^x$ is determined by $e^{-\mathcal{C}^x}$. In other words, we calculate the probability of invoking the change detection module based on the confidence score itself. Therefore, if the confidence in classifying an instance is high, probability of invoking the change detection module will be low and vice versa. We will refer to this version as ECHO-D.

V. Experiment Results

A. Data Sets

Table II depicts the characteristics of the data sets. The *ForestCover* data set is obtained from the UCI repository as explained in [21]. In order to prepare it for novel class detection, we arrange the data so that at most three and at least two classes co-occur at similar time. For the second data set, we use the Physical Activity Monitoring *(PAMAP)* data set from UCI [25]. In this data set, nine people were equipped with sensors that gathered a total of 52 streaming metrics whilst they performed activities. Third data set is *Powersupply* [26], which 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.

TABLE II: Characteristics of data sets

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

HyperPlane [26] 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 synthetic data sets generated using *RandomRBFGeneratorDrift* of MOA [27] 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 ForestCover and PAMAP data sets for simulating both concept drift and novel classes. Rest of the data sets are used to test only concept drift handling capability of the considered approaches.

B. Experiment Setup

We implement two versions of the proposed framework, i.e., ECHO-F, ECHO-D to analyze performance as discussed in Section IV. We compare classification and novel class detection performance of our framework with *ECSMiner* [21]. We have chosen ECSMiner since it addresses both concept drift and concept evolution. Other than that, we compare performance of ECHO with *OzaBagAdwin* (OBA) and *Adaptive Hoeffding Tree* (AHT) implemented in MOA [27] framework, since these approaches seem to have superior performance than others on the data sets used in the experiments. Both OBA and AHT use *ADWIN* [4] as the change detector. These approaches do not have novel class detection capability. So, we compare ECHO with these approaches only in terms of classification performance.

We evaluate each of the above classifiers on a stream by testing and then training with chunks of data in sequence. To evaluate ECSMiner, we use 50 pseudopoints, and ensemble size 6 as suggested in [21]. We use 100% labeled training data to evaluate ECSMiner, OBA and AHT. On the other hand, we set number of models in the ensemble (t), and q in ECHO using cross validation.

C. Performance Metrics

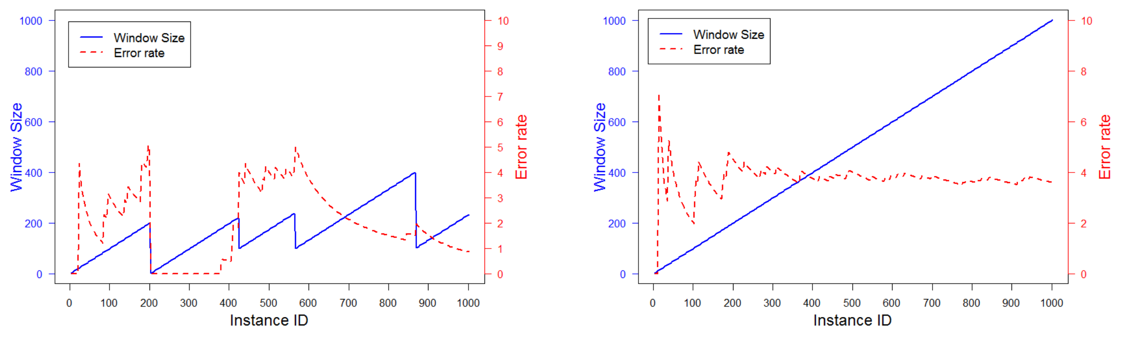
Let FN = total novel class instances misclassified as existing class, FP = total existing class instances misclassified as novel class, TP = total novel class instances correctly classified as novel class, Fe = total existing class instances misclassified (other than FP), N_c = total novel class instances,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

TABLE III: Summary of classification results

Name of	ECHO-F (τ = 0.9)		ECHO-D (τ = 0.9)		ECSMiner	AHT	OBA
Data Set	Error%	% of labeled data	Error%	% of labeled data	Error%	Error%	Error%
ForestCover	3.95	96.54	**3.68**	95.16	4.55	22.89	18.06
PAMAP	4.75	93.64	**3.73**	94.7	35.26	8.76	7.27
Power Supply	0.01	8.93	**0.01**	9.8	0.01	85.59	86.92
HyperPlane	2.79	99.76	**2.18**	100	3.73	46.24	48.55
SynRBF@0.002	**23.98**	99.79	34.01	99.93	63.43	38.75	37.04
SynRBF@0.003	**22.37**	99.51	36.91	99.8	65.39	48.65	46.86

TABLE IV: Comparison of classification performance using limited amount of labeled data

Name of	ECHO-F (τ = 0.4)		ECHO-D (τ = 0.4)		ECSMiner	AHT	OBA
Data Set	Error%	% of labeled data	Error%	% of labeled data	Error%	Error%	Error%
ForestCover	7.96	39.26	**3.88**	35.33	4.55	22.89	18.06
PAMAP	4.77	69.56	**4.73**	68.62	35.26	8.76	7.27
Power Supply	0.01	0.1	**0.01**	0.14	0.05	85.59	86.92
HyperPlane	2.99	32.6	**2.36**	26.9	3.73	46.24	48.55
SynRBF@0.002	53.65	38.4	44.99	28.17	63.43	38.75	**37.04**
SynRBF@0.003	50.51	36.51	**40.65**	54.48	65.39	48.65	46.86



(a) SynRBF@0.002 (b) HyperPlane

Fig. 2: Change of window size and error rate of ECHO-D as the stream progresses

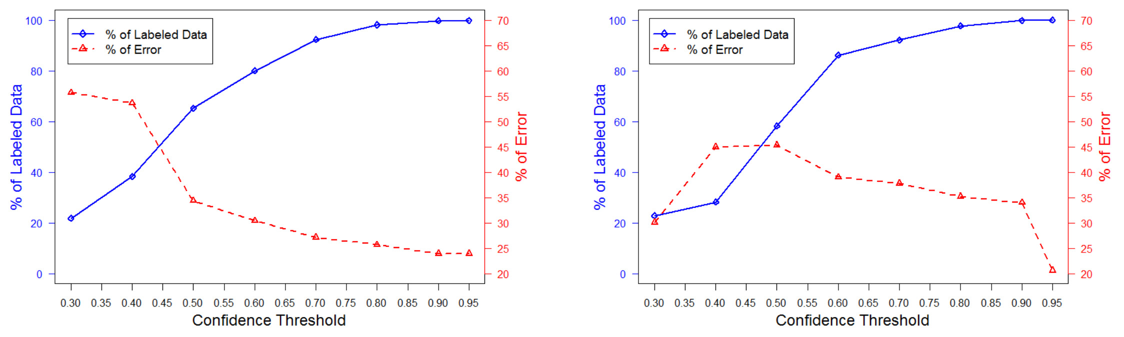
Falsely identified as novel class, i.e., $\frac{FP*100}{N-N_c}$. 3) F_2: F_β score provides the overall performance of a classifier by considering both *precision* and *recall*. In this paper, we use $\beta = 2$, which gives us $F_2 = \frac{5*TP}{5*TP+4*FN+FP}$.

D. Classification

As discussed before, most of the techniques to classifying evolving data stream divide the stream into fixed-size chunks regardless of occurrence or intensity of concept drifts. On the contrary, ECHO determines the chunk size dynamically based on any significant change in classifier confidence. Hence, ECHO avoids unnecessary training during stable period and frequently update the classifier when needed. This characteristic is evident in Figure 2. In this experiment, we have considered the first *1000* instances and plotted the window size and error rate as the stream progresses. In case of SynRBF@0.002 data set, whenever the error rate of the classifier increases significantly, our proposed CDT detects the drift from change in confidence scores, determines the chunk boundary, and updates the classifier. Therefore, the concept drift is accommodated and the error rate drops as expected. On the contrary, Hyperplane data set has less frequent and milder concept drift within the first *1000* instances, and the classifier performance is stable. Hence, ECHO keeps growing the sliding window without updating the classifier. This indicates effectiveness of our proposed semi-supervised concept drift detection technique.

We discussed in Section III-G that ECHO selects instances for labeling based on the confidence threshold (τ). So, percentage of labeled data used to update the classifier depends on the value of τ. Higher value of τ incurs larger amount of labeled data and vice versa. Figure 3 shows how percentage of labeled data and error rate vary with increasing τ on SynRBF@0.002 data set. We observe that percentage of labeled data increases and error rate decreases with increasing τ for both ECHO-F and ECHO-D as expected.

Table III compares performance of ECHO with other considered approaches in terms of classification accuracy. We have used confidence threshold $\tau = 0.9$ in this experiment. In all the cases, ECHO outperforms the other approaches by a large margin. We also observe that, high value of τ in general results in large amount of labeled data. However, this is not always true. For example, in case of Power Supply data set, ECHO versions use labels for less than 10% of instances despite using high τ, still exhibit superior performance. As



(a) ECHO-F on SynRBF@002

(b) ECHO-D on SynRBF@002

Fig. 3: Percentage of labeled data and percentage of error with increasing value of confidence threshold (τ)

discussed in Section III-G, ECHO only requests label for an instance if the confidence is below τ. It implies that, in case of Power Supply data set, ECHO has high confidence in classifying most of the test instances. So, instead of requesting labels for those instances, ECHO uses the predicted labels while updating the classifier.

The number of instances ECHO requests labels for is controlled by τ. We show classification performance of ECHO using $\tau = 0.4$ in Table IV. Although, other approaches considered in this table use 100% labeled data, we have put performance of these approaches for ease of comparison. We observe that, ECHO versions use extremely low amount of labeled data because of using low value for τ. However, ECHO versions still show competitive performance, if not better, than the other considered approaches in case of all the data sets. So, experiment result suggests that ECHO can be used in scenarios where labeled data is very scarce and expensive.

E. Speed Up

ECHO achieves significant *Speed Up* using dynamic programming and other performance improvement techniques discussed in Section IV. To examine the improvement in terms of execution time, we compare performance of ECHO-F and ECHO-D with basic ECHO implementation without any performance improvement. We define Speed Up as T_b/T_o, where T_b is the total time taken by the basic ECHO implementation and T_o is the total time taken by a improved version of ECHO. Figure 4 shows the Speed Up achieved by ECHO-F and ECHO-D on SynRBF@X data sets with increasing τ. As discussed before, higher value of X indicates more frequent and more intense concept drift in SynRBF@X. We observe that improved versions of ECHO achieve as high as 18 times Speed Up than the basic implementation on SynRBF@0.002. In case of SynRBF@0.003, improved versions achieve as high as 120 times Speed Up which is very promising. This indicates the effectiveness of our proposed strategies for performance improvement.

F. Novel Class Detection

ECHO stores the filtered outliers temporarily in a buffer until there are enough instances in the buffer to detect a novel

TABLE V: Novel class detection performance using $\tau = 0.9$

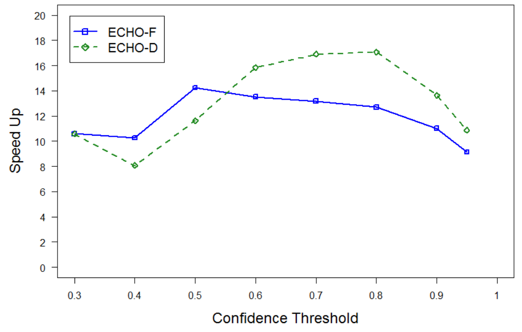
Data set	Method	M_{new}	F_{new}	F_2
ForestCover	ECHO-F	11.37	2.27	0.72
	ECHO-D	14.51	**2.11**	0.71
	ECSMiner	**8.42**	2.13	**0.88**
PAMAP	ECHO-F	0.05	4.22	0.78
	ECHO-D	**0.05**	**3.39**	**0.82**
	ECSMiner	0.05	37.53	0.45

pattern. As the stream progresses, more instances from the novel pattern arrive in the stream which makes it easier to detect the novel pattern. However, similar to ECSMiner, we also consider a maximum allowable time up to which the classifier can wait for enough instances from a emerging class to appear. We set this time constraint as *400* instances as suggested in [21]. Table V compares novel class detection performance of ECHO-D and ECHO-F using $\tau = 0.9$ with ECSMiner on ForestCover and PAMAP data sets. We observe that, both versions of ECHO show competitive performance. ECHO-D shows the best F_{new} and competitive M_{new} and F_2 on ForestCover data set despite using less labeled data than ECSMiner. In case of PAMAP data set, ECHO-D shows better performance than ECSMiner in terms of all M_{new}, F_{new}, and F_2.

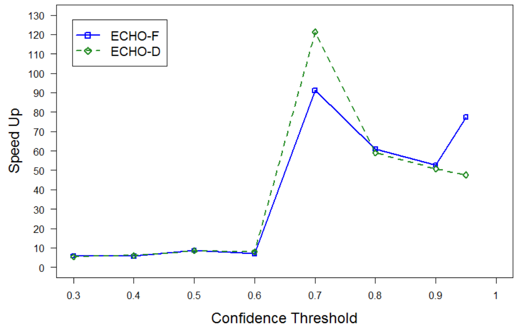
If we consider overall performance (*Error %*, M_{new}, F_{new}, and F_2), ECHO clearly outperforms all the other approaches considered in this paper.

VI. Conclusion

In this paper, we present a semi-supervised framework ECHO for classifying evolving data streams. It detects concept drift and determines chunk boundary dynamically by finding any significant change in classifier confidence. Moreover, it also uses confidence scores to intelligently select limited amount of data instances for labeling from the latest chunk, which is then used to update the classifier. We theoretically justify use of the confidence estimators and effect of concept drift on classifier confidence. We propose several strategies



(a) SynRBF@0.002



(b) SynRBF@0.003

Fig. 4: Confidence threshold (τ) vs Speed Up

including dynamic programming to improve execution time of ECHO. Empirical results show the effectiveness of ECHO.

Acknowledgments

This material is based upon work supported by NSF award no. CNS-1229652, DMS-1322353, and The Air Force Office of Scientific Research under award no. FA-9550-09-1-0468.

References

[1] A. Haque, L. Khan, and M. Baron, "Semi supervised adaptive framework for classifying evolving data stream," in *The 19TH Pacific-Asia Conference on Knowledge Discovery and Data Mining (PAKDD)*, 2015, pp. 383–394.

[2] 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.

[3] R. Klinkenberg, "Learning drifting concepts: Example selection vs. example weighting," *Intell. Data Anal.*, vol. 8, no. 3, pp. 281–300, Aug. 2004.

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

[5] 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*, 2008, pp. 929–934.

[6] A. Haque, L. Khan, and M. Baron, "Sand: Semi-supervised adaptive novel class detection and classification over data stream," in *THIRTIETH AAAI Conference on Artificial Intelligence*, Feb 2016.

[7] B. Settles, "Active learning literature survey," University of Wisconsin–Madison, Computer Sciences Technical Report 1648, 2009.

[8] X. Song, M. Wu, C. Jermaine, and S. Ranka, "Statistical change detection for multi-dimensional data," in *13th ACM SIGKDD*. NY, USA: ACM, 2007, pp. 667–676.

[9] 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.

[10] G. J. Ross, D. K. Tasoulis, and N. M. Adams, "Nonparametric monitoring of data streams for changes in location and scale," *Technometrics*, vol. 53, no. 4, pp. 379–389, 2011.

[11] M. Harel, S. Mannor, R. El-yaniv, and K. Crammer, "Concept drift detection through resampling," in *ICML-14*. JMLR Workshop and Conference Proceedings, 2014, pp. 1009–1017.

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

[13] D. Cieslak and N. Chawla, "Detecting fractures in classifier performance," in *ICDM 2007*, Oct 2007, pp. 123–132.

[14] C. Alippi, G. Boracchi, and M. Roveri, "Just-in-time classifiers for recurrent concepts." *IEEE Trans. Neural Netw. Learning Syst.*, vol. 24, no. 4, pp. 620–634, 2013.

[15] K. Nishida, K. Yamauchi, and T. Omori, "Ace: Adaptive classifiers-ensemble system for concept-drifting environments." in *Multiple Classifier Systems*, ser. Lecture Notes in Computer Science, vol. 3541. Springer, 2005, pp. 176–185.

[16] M. Masud, J. Gao, L. Khan, J. Han, and B. Thuraisingham, "Classification and novel class detection in data streams with active mining," in *Advances in Knowledge Discovery and Data Mining*, ser. Lecture Notes in Computer Science, 2010, vol. 6119, pp. 311–324.

[17] W. Fan, Y. an Huang, H. Wang, and P. S. Yu, "Active mining of data streams," in *in Proceedings of the Fourth SIAM International Conference on Data Mining*, 2004, pp. 457–461.

[18] K. B. Dyer, R. Capo, and R. Polikar, "Compose: A semisupervised learning framework for initially labeled nonstationary streaming data," *IEEE Transactions on Neural Networks and Learning Systems*, vol. 25, no. 1, pp. 12 – 26, 2014.

[19] E. J. Spinosa, A. P. de Leon, and J. . Gama, "Novelty detection with application to data streams," *Intell. Data Anal.*, vol. 13, pp. 405–422, Aug. 2009.

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

[21] 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.*, vol. 23, no. 6, pp. 859–874, 2011.

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

[23] M. Ester, H.-P. Kriegel, J. Sander, and X. Xu, "A density-based algorithm for discovering clusters in large spatial databases with noise," in *Proc. of 2nd International Conference on Knowledge Discovery and*, 1996, pp. 226–231.

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

[25] A. Reiss and D. Stricker, "Introducing a new benchmarked dataset for activity monitoring." in *ISWC*. IEEE, 2012, pp. 108–109.

[26] X. Zhu, "Stream data mining repository," http://www.cse.fau.edu/~xqzhu/stream.html, 2010.

[27] 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*, 2010, pp. 44–50.