An Adaptive Framework for Multistream Classification

Swarup Chandra[1], Ahsanul Haque[1], Latifur Khan[1] and Charu Aggarwal[2]
[1]Dept. of CS, The University of Texas at Dallas
(src093020, axh129430, lkhan)@utdallas.edu
[2]IBM T. J. Watson Research Center
charu@us.ibm.com

ABSTRACT

A typical data stream classification involves predicting label of data instances generated from a non-stationary process. Studies in the past decade have focused on this problem setting to address various challenges such as concept drift and concept evolution. Most techniques assume availability of class labels associated with unlabeled data instances, soon after label prediction, for further training and drift detection. Moreover, training and test data distributions are assumed to be similar. These assumptions are not always true in practice. For instance, a semi-supervised setting that aims to utilize only a fraction of labels may induce bias during data selection. Consequently, the resulting data distribution of training and test instances may differ. In this paper, we present a novel stream classification problem setting involving two independent non-stationary data generating processes, relaxing the above assumptions. A *source stream* continuously generates labeled data instances whose distribution is biased compared to that of a *target stream* which generates unlabeled data instances from the same domain. The problem, we call *Multistream Classification*, is to predict the class labels of data instances in the target stream, while utilizing labels available on the source stream. Since concept drift can occur asynchronously on these two streams, we design an adaptive framework that uses a technique for supervised concept drift detection in the biased source stream, and unsupervised concept drift detection in the target stream. A weighted ensemble of classifiers is updated after each drift detection on either streams, while utilizing a bias correction mechanism that leverage source information to predict labels of target instances whenever necessary. We empirically evaluate the multistream classifier's performance on both real-world and synthetic datasets, while comparing with various baseline methods and its variants.

Keywords

Data Stream; Classification; Covariate Shift; Concept Drift

CIKM'16 , October 24-28, 2016, Indianapolis, IN, USA
© 2016 ACM. ISBN 978-1-4503-4073-1/16/10. . . $15.00
DOI: http://dx.doi.org/10.1145/2983323.2983842

1. INTRODUCTION

Recent proliferation of Internet technology in every day lifestyle has created streaming sources that generate data continuously. These include social media applications such as Twitter, wearable devices, Internet of Things (IoT) and healthcare systems. Data generated from these sources can be used in applications such as personalized services, human activity recognition, and recommendation systems. The problem of classifying instances (i.e., predicting its class label) in a data stream is challenging due to unknown (and potentially infinite) data size, and changes in data distribution that occur with time. Specifically, a non-stationary data generation process may induce arbitrary changes in population distribution over time. This is known as *data drift* [10]. Consequently, a classifier trained on a particular set of training instances may become obsolete with time.

A plethora of studies [3, 9, 11] in the past decade have proposed techniques to address these challenges. Essentially, classifiers that are trained on minibatches (i.e. subset of sequential instances in a data stream) of finite size are used to predict class label of each new data instance. However, these techniques make two strong assumptions about the ability of applications employing them. First, true labels of data instances along the stream are assumed to be available soon after prediction. These true labels are typically used in adaptation mechanisms either for drift detection or classifier retraining [10]. Often, true class labels are obtained using an oracle or a manual labeling mechanism, which are either impractical or expensive. Second, sample data distribution of the training and test minibatches are assumed to be equal during classification. This assumption may often be violated in many real-world applications, especially in a semi-supervised setting where only a fraction of true labels are utilized. In this case, a mechanism to choose data instances for labeling may be biased. This induces a difference in data distribution between the training and test data. A classifier trained on a biased training dataset may perform poorly on future unbiased test data occurring along the stream.

Consider for example an application using an existing stream sentiment classifier over a Twitter stream [15], where tweets from numerous users are continuously generated. The true sentiment (class label) of each tweet, required by the classifier, may be manually provided by either the user generating it or a supervisor processing it. Typically, one can assume that a user does not provide such labels unless explicitly requested. Therefore, a small set of users may be requested to provide true labels while they generate tweets.

Since the sentiment classification model generalizes to the population, it can be used to predict sentiments of any given tweet. However, labeled tweets from only a small set of users, who may not be a good representation of the population, are now used to train the classifier. This creates a bias in the training data distribution. With unavailability of labels from all users and biased training data, the two strong assumptions of the classifier are clearly violated.

In this paper, we introduce a new problem setting consisting of two types of data streams over the same domain such that these assumptions can be relaxed. A stream of labeled data instances are generated by a non-stationary process from a particular domain. This is referred to as *Source Stream*. Another stream of data instances is assumed to be generated by an independent non-stationary process, referred to as *target stream*, from the same domain. However, class labels of these data instances are unknown. The classification problem is to predict class labels of data instances in the concept drifting target stream while leveraging labeling information available in the concept drifting source stream. Since this classification problem involves two types of data streams, i.e., labeled source stream and unlabeled target stream, we call it as *Multistream Classification*. This setting is illustrated in Figure 1. Naively, one can combine the two data streams to form a single stream over which existing data stream solutions can be utilized. However, unavailability of target data instance labels and occurrence of concept drift asynchronously at different times in the source and target stream, due to independent non-stationary processes, violate assumptions made by these solutions, thereby reducing their performance.

The multistream problem setup partly resonates with that of domain adaptation [2] and transfer learning [19] when considering a set of observations in each stream within a particular time window. Numerous applications involving such problems include text classification [8], sensor based location estimation [20] and collaborative filtering [16]. However, these studies assume stationary environment with known dataset size. Clearly, such applications can be easily augmented to data streams, and also the multistream setting. In the Twitter sentiment classification example mentioned earlier, the biased labeled stream of tweets can be viewed as a source stream, while tweets from rest of the population can be considered as a target stream.

In general, domain adaptation is not possible when the training and test data distributions are unrelated [2]. One type of relationship between them is called *covariate shift* [6]. Here, the training and test distributions are assumed to have equal conditional class distribution with unequal marginal distribution. A correction mechanism is employed to weigh training data instances so that their weighted data distribution resembles that of the observed test data distribution. In the case of multistream classification, one can naively apply covariate shift correction over the source stream to utilize its labeling information towards label prediction on the target stream. However, such methodology may not be efficient in minimizing prediction errors due to asynchronous concept drifts that may disrupt covariate shift assumptions. The correction mechanism employed at a particular time may become outdated, thereby increasing classification error. We address these challenges by proposing an ensemble of weighted classifiers that inculcate covariate shift correction to predict labels of target data instances, while using a change detection technique to detect concept drift over both streams that aid in restoration of covariate shift assumption. Since only the (given) true labels of the source stream are utilized while being completely unsupervised with respect to the target stream, we term this mechanism as quasi-unsupervised classification. The main contributions of the paper are summarized as follows.

1. We introduce a new data stream classification setting consisting of two independent non-stationary streams. Class labels are predicted over one stream (target) whose data instances are unlabeled, using the labeled data instances from another stream (source). We term this problem as Multistream Classification.

2. We design an ensemble of weighted classifiers to address challenges in adapting the source data distribution, which may be biased, towards the target data distribution in a non-stationary environment.

3. We propose an adaptive framework for multistream classification, using limited memory with dynamic size minibatch, by measuring classifier feedback on both streams independently. We perform concept drift detection without requiring true labels from the target stream, and using labels observed on the source stream.

4. We evaluate this proposed approach empirically over numerous datasets, and compare the results with some baseline methods. Furthermore, we also evaluate variants of the framework.

The paper is organized as follows. We review relevant studies in Section 2, and formally present the multistream classification problem in Section 3. In Section 4, we describe our solution, and analyze some of its theoretical properties in Section 5. We present our empirical evaluation on various datasets in Section 6, and conclude our paper in Section 7.

2. RELATED WORK

2.1 Stream Classification

Data stream classification is a challenging task due to its inherent properties, e.g., infinite length, concept drift, etc. Infinite length problem is addressed by dividing the whole stream into fixed-size minibatches, e.g., [17] or using gradual forgetting mechanism, e.g., [14]. Recent approaches [3, 9, 11] address this problem by remembering only the instances within two consecutive concept drifts using a dynamic size minibatch. The minibatch size is increased until a concept drift, e.g., change of class boundaries, is detected. Once a drift is detected, the classifier is updated and the instances representing the old concept in the minibatch are removed.

In the literature, concept drift is detected by tracking any change in $P(x)$ or in $P(y|x)$. Since real-world data streams are mostly multivariate, detecting change in $P(x)$ is a hard problem [12]. Concept drift detection in multivariate data streams therefore concentrate on tracking any changes in the posterior class distribution, i.e., $P(y|x)$. Instead of tracking changes in $P(y|x)$ directly, approaches proposed in [3, 9] adopt the principle by Vapnik [24] to detect this change indirectly by tracking drift in the error rate of underlying classifier. However, tracking drift in the error rate requires true labels of test data instances, which are scarce in practice. Recent studies, e.g., [17] have focused on partly addressing this issue by assuming delayed labeling or active learning settings, yet requiring some test data label. Approach proposed in [11] addresses this issue by estimating classifier confidence in classification, and tracking any significant change in confidence scores. However, it focuses on multi-class classification problem in a single stream.

In this paper, we utilize a change point detection mechanism in a unique manner. Since multistream classification involves two streams, change point detection is performed on the labeled source data as well as the unlabeled target data using classifier feedback heuristic. Specifically, error rate is used as classifier feedback on the source stream as labels are available in this stream, while unsupervised classifier confidence is used in case of the target stream.

2.2 Datashift Adaptation

In the Bayesian perspective, a change in data distribution between training and test datasets is viewed as a difference in their joint probability distribution $P(x, y)$. In the realm of domain adaptation, a type of change in distribution can be viewed as $P_{tr}(x) \neq P_{te}(x)$ with $P_{tr}(y|x) = P_{te}(y|x)$. This is known as *covariate shift* [13]. Recent studies focus on developing correction mechanisms for covariate shift by estimating probability density ratio given by $\beta(x) = \frac{P_{te}(x)}{P_{tr}(x)}$. These include Kernel Mean Matching [13] (KMM) and Kullback Leibler Importance Estimation Procedure [23] (KLIEP).

In particular, Kernel Mean Matching (or KMM) aims to reduce mean discrepancy between the weighted training data distribution $\beta(x)P_{tr}(x)$ and the observed test data distribution $P_{te}(x)$ in a Reproducing Kernel Hilbert Space (RKHS) $\phi(x) : x \to \mathcal{F}$. The mean distance is measured using the Maximum Mean Discrepancy (MMD), given by

$$\left\| E_{x \sim P_{tr}(x)}[\beta(x)\phi(x)] - E_{x \sim P_{te}(x)}[\phi(x)] \right\| \quad (1)$$

where $\|\cdot\|$ is the l_2 norm. The main assumptions here are-

Assumption 1. *P_{te} is absolutely continuous with respect to P_{tr}, i.e. $P_{te}(x) = 0$ whenever $P_{tr}(x) = 0$. Additionally, the RKHS kernel ϕ is universal.*

It has been shown that under these conditions, minimizing MMD in Equation 1 converges to $P_{te}(x) = \beta(x)P_{tr}(x)$ [27].

In the case of multistream data classification, estimating the validity of all these necessary conditions is challenging due to asynchronized concept drifts on the source and target streams. Nevertheless, we would want to adapt the source stream distribution towards the target distribution by correcting for covariate shift. In this paper, we devise a mechanism to incorporate covariate shift correction using KMM for multistream classification.

3. PRELIMINARIES

In this section, we formalize the multistream classification problem and present challenges of performing classification over drifting data streams in this context.

3.1 Notations

Table 1 lists frequently used symbols in this paper. A bold symbol or letter is used to denote a set of elements, and a superscript to indicate index of an element in the set. A subscript is used to indicate association of an element or a set to a type such as source or target stream. For example, $x_S^{(i)} \in \mathbf{X}_S$ denotes the i^{th} data instance belonging to a set $\mathbf{X}$ of data instances from the source stream $\mathcal{S}$.

Symbols	Description
$\mathcal{D}$	Domain
$\mathbf{P}$	Set of non-stationary processes
$\mathcal{S} \in \mathbf{P}$	A labeled source stream
$\mathcal{T} \in \mathbf{P}$	An unlabeled target stream
x	v-dimensional features (or covariates)
$y \in \mathbf{Y}$	Class label of a data instance
$\mathcal{M}$	Classifier
P_{tr}	Probability distribution of training minibatch
P_{te}	Probability distribution of test minibatch
β	Probability Density Ratio $\frac{P_{te}}{P_{tr}}$
n	Size of minibatch
E	Ensemble of classifiers
E_{max}	Maximum allowable ensemble size
S_{max}	Maximum allowable minibatch size

Table 1: List of Symbols

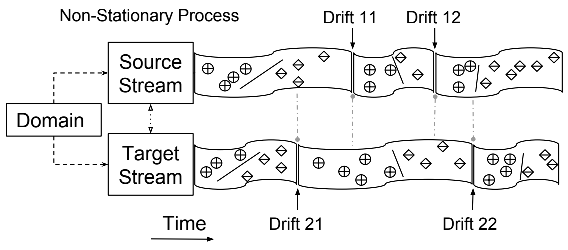


Figure 1: An example illustrating asynchronous data drifts on the source and target streams.

3.2 Problem Statement

Consider a set of independent non-stationary processes (denoted as $\mathbf{P}$), each generating a continuous stream of data instances from a domain $\mathcal{D}$. A data instance is denoted by (x, y), where $x \in \mathcal{D}^v$, is a vector of v covariates, and $y \in \mathbf{Y} = \{1 \dots k\}$, is its corresponding class label. Here, k is the number of classes. In this setting, assume that $\mathbf{P}$ consists of two processes, denoted as $\mathcal{S}$ and $\mathcal{T}$. Throughout the paper, we use the symbol denoting a process interchangeably with its corresponding data stream. In process (or stream) $\mathcal{S}$, both x and y of each data instance are observed. On the contrary, only x of each data instance in stream $\mathcal{T}$ is observed. This indicates that $\mathcal{S}$ is completely labeled while $\mathcal{T}$ is unlabeled. We define the *MultiStream Classification* problem as follows.

Definition 1. *Let $\mathbf{X}_S \in \mathcal{D}$ be a set of v-dimensional vectors of covariates and $\mathbf{Y}_S$ be the corresponding class labels observed on a non-stationary stream $\mathcal{S}$. Similarly, let $\mathbf{X}_T \in \mathcal{D}$ be a set of v-dimensional vectors of covariates observed on an independent non-stationary stream $\mathcal{T}$. Construct a classifier $\mathcal{M}$ that predicts class label of $x \in \mathbf{X}_T$ using $\mathbf{X}_S$, $\mathbf{Y}_S$ and $\mathbf{X}_T$.*

Since $(x, y) \in \mathcal{S}$ are used to predict labels of $x \in \mathcal{T}$, we call $\mathcal{S}$ as the *source stream* and $\mathcal{T}$ as the *target stream.*

3.3 Challenges

As mentioned in Section 2, a biased training distribution can be related to a test distribution by the covariate shift assumption. In the case of multistream setting, an equivalent condition is given by considering sample distributions of $\mathcal{S}$ and $\mathcal{T}$ within a specific time period. Concretely, $P_S^{(t)}(y|x) = P_T^{(t)}(y|x)$ and $P_S^{(t)}(x) \neq P_T^{(t)}(x)$ at time t. Following the motivating example given in Section 1, $\mathcal{S}$ contains biased data compared to $\mathcal{T}$.

However, this assumption may not be true at time $r > t$ due to the non-stationary nature of data streams. Within a data stream, conditional probability distribution may change over time, i.e. $P^{(t)}(y|x) \neq P^{(r)}(y|x)$. This is typically called a *concept drift*. Similarly, a change in covariate distribution with time is called a *covariate drift*. With two independent non-stationary processes generating data continuously from $\mathcal{D}$, the effect of a drift may be observed at different times on these streams. This *asynchronous* data drift between source and target streams is illustrated in Figure 1 as an example. Here, four independent data drifts occurring at different times on $\mathcal{S}$ and $\mathcal{T}$ are indicated. Moreover, the drifting concepts may not be similar between the source and target stream. For example in Figure 1, data distribution after `Drift 21` may not be similar to that after `Drift 11`.

Asynchronous drifts also affect covariate shift correction. In particular, the density ratio estimated, using methods such as KMM, between the target and source distribution may change over time. Consequently, performance of a classifier trained on bias-corrected data is affected. Intuitively, this can be overcome by re-estimating density ratios and training a new bias-corrected classifier.

We address these challenges posed by drifting streams in a multistream setting by designing a fixed size ensemble of weighted classifiers consisting of models trained on data instances in minibatches from $\mathcal{S}$ and $\mathcal{T}$ for predicting class label for data on $\mathcal{T}$. Concept drifts on each stream are detected independently, which triggers a mechanism to update the ensemble with a retrained classifier model, including re-estimation of density ratios whenever necessary.

4. MULTISTREAM CLASSIFICATION

In this section, we describe our proposed framework for multistream classification, referred to as MSC (MultiStream Classifier). As discussed earlier, in multistream classification, covariate shift assumption holds with respect to the target stream until there is a concept drift in any of the streams. The goal of MSC is to use the labeled data from the source stream, and predict labels for the target instances efficiently. To do that, we use an ensemble with two types of classifiers, i.e., source-classifiers and target-classifiers. Source-classifiers are trained only on labeled source stream data. On the contrary, target-classifiers are trained using bias-corrected source data whose distribution resembles the target distribution. If there is a concept drift in any of the streams, covariate shift assumption becomes invalid. Therefore, we use a change detection technique (CDT) to detect concept drifts in source and target stream. If a concept drift is detected in either of the streams, we re-weight training instances and update the ensemble to restore the covariate shift assumption.

Figure 2 illustrates various components that form the overall classification process shown in Algorithm 1. Data instances from the source and target streams are obtained continuously. The ensemble is initialized with a source-classifier and a target-classifier, each trained on an initial minibatch of data instances from appropriate streams. Using this initial-

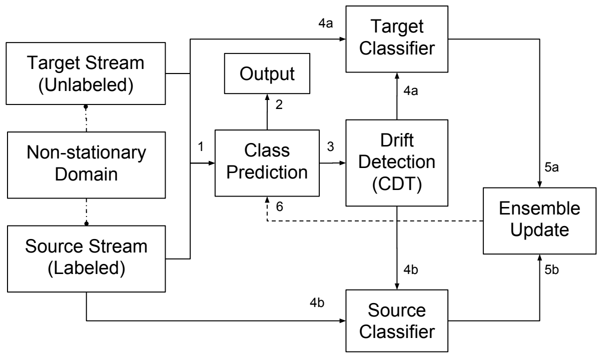


Figure 2: Multistream Classification Overview

ized ensemble, the multistream classification is performed as follows. At a given time period, data instances occur on both $\mathcal{S}$ and $\mathcal{T}$. A data instance from $\mathcal{T}$ is first classified using the ensemble, illustrated by `Class Prediction` in Figure 2 (Step 2). Apart from an estimated class label, the ensemble also provides a value representing its confidence in classifying the data instance. These results are recorded in a buffer. Similarly, any data instance of $\mathcal{S}$ is also classified using the ensemble. In this case, the classification accuracy is recorded since corresponding class label is observed on $\mathcal{S}$. These recorded classifier feedback are used to detect concept drift in the `Drift Detection` phase, as illustrated in the figure (Step 3). Depending on the type of drift detected, a new classifier is trained using the latest minibatch data available. Particularly, a model is trained in a supervised manner using the latest minibatch from $\mathcal{S}$ if a change in distribution of recorded classifier error is detected on $\mathcal{S}$. We call the resulting classifier as a *source-classifier*. This training process is illustrated as `Source Classifier` in the figure (Step 4a). However, if a drift in recorded classifier confidence on $\mathcal{T}$ is detected, then a bias-corrected model is trained using latest minibatches from $\mathcal{S}$ and $\mathcal{T}$. We call the resulting classifier as a *target-classifier*. This training process is illustrated as `Target Classifier` in the figure (Step 4b). Next, newly created classifier is added to the ensemble using an update process, illustrated as `Ensemble Update` in the figure (Step 5a and 5b). The updated ensemble is further used for classification along the stream (Step 6). We now present the details of each component in Algorithm 1.

4.1 Initialization and Classifier Training

Data occurring on $\mathcal{S}$ and $\mathcal{T}$ are stored in a data buffer denoted by B_S and B_T respectively, using *readData* in Algorithm 1. Initially, we consider a warm-up period consisting of I labeled instances from $\mathcal{S}$, and I unlabeled instances from $\mathcal{T}$, which are stored in B_S and B_T respectively. These instances are used to initialize the ensemble of classifiers, denoted as E, using *initializeEnsemble* in the algorithm. Since data instances occur from two different distributions (i.e. $\mathcal{S}$ and $\mathcal{T}$) simultaneously, the ensemble should have the ability to provide good feedback on each data type for future drift detection. Therefore, a source-classifier and a target-classifier are initially added to the ensemble.

Concretely, we train a base model in a supervised manner using B_S, which results in a classifier denoted by $\mathcal{M}_S$

Algorithm 1: MSC : Multistream Classification

```
Data: Data Streams $\mathcal{S}$ and $\mathcal{T}$
Input: Initial Size : $I$
Result: Labels on $\mathcal{T}$
begin
    $B_S, B_T \leftarrow readData(\mathcal{S}, \mathcal{T}, I)$ /*Data Buffer*/
    $\mathcal{M}_S \leftarrow buildSourceModel(B_S)$
    $\mathcal{M}_T \leftarrow buildTargetModel(B_S, B_T)$
    $E \leftarrow initializeEnsemble(\mathcal{M}_S, \mathcal{M}_T)$
    Initialize $W_S$ and $W_T$ /*Feedback Buffers*/
    while $\mathcal{S}$ or $\mathcal{T}$ exists do
        $B_S, B_T \leftarrow readData(\mathcal{S}, \mathcal{T}, 1)$
        /*For Source Stream*/
        $W_S \leftarrow getError(E, B_S)$
        if $z \leftarrow checkDrift(W_S)$ then
            $B_S, W_S \leftarrow updateMinibatch(z, B_S, W_S)$
            $\mathcal{M}_S \leftarrow buildSourceModel(B_S)$
            $updateEnsemble(E, \mathcal{M}_S, B_S, B_T)$
        /*For Target Stream*/
        $W_T, \hat{Y}_T \leftarrow predict(E, B_T)$
        if $z \leftarrow checkDrift(W_T)$ then
            $B_T, W_T \leftarrow updateMinibatch(z, B_T, W_T)$
            $\mathcal{M}_T \leftarrow buildTargetModel(B_S, B_T)$
            $updateEnsemble(E, \mathcal{M}_T, B_S, B_T)$
        print $getAccuracy(\hat{Y}_T, B_T)$
```

from *buildSourceModel* in the algorithm. This forms a source-classifier. On the contrary for $\mathcal{T}$, source bias correction is performed by estimating the density ratios (instance weights) between B_T and B_S using KMM. We then train a base model using B_S along with its instance weights in a supervised manner to form a bias-corrected classifier denoted by $\mathcal{M}_T$, from *buildTargetModel* in the algorithm. This forms a target-classifier. Particularly, the density ratio $\beta(B_S) = \frac{P_T(B_T)}{P_S(B_S)}$ is computed by minimizing the Maximum Mean Discrepancy (MMD) in Equation 1. An optimal β is obtained by minimizing the corresponding empirically equivalent quadratic optimization problem [13], given by

$$\boldsymbol{\beta}^{(t)*} \approx \underset{\beta^{(t)}}{\text{minimize}} \frac{1}{2}\boldsymbol{\beta}^T \mathbf{K}\boldsymbol{\beta} - \boldsymbol{\kappa}^T\boldsymbol{\beta}$$
$$\text{subject to } \beta_i \in [0, B_{kmm}] \;\&\; \left|\frac{1}{n_{tr}}\sum_{i=1}^{n_{tr}}\beta_i - 1\right| \leq \epsilon_{kmm} \tag{2}$$

where $\mathbf{K}$ and $\boldsymbol{\kappa}$ are matrices of a RKHS kernel $k(\cdot)$ with $K_{ij} = k(x_{tr}^{(i)}, x_{tr}^{(j)})$, and $\kappa_i = \frac{n_{tr}}{n_{te}}\sum_{j=1}^{n_{te}} k(x_{tr}^{(i)}, x_{te}^{(j)})$, and B_{kmm} is a parameter greater than 0. These training procedures are used throughout the algorithm while using the latest available data in B_S and B_T.

Once a classifier is trained, it is associated with two types of weights, i.e., source weight w_S and target weight w_T, based on its performance on B_S and B_T respectively. Since B_S instances are labeled, we use the classification error (denoted by η) to calculate w_S, which is given by $\frac{1}{2}\ln\frac{1-\eta}{\eta}$. However, B_T instances are unlabeled. Therefore, we use the classifier's confidence on B_T (denoted by ζ) to calculate w_T in a similar manner. We further analyze the use of such heuristic as a surrogate to error rate in Section 5. The weighted classifier is then added to E.

4.2 Class Prediction

Every classifier in the ensemble predicts a class label for a given data instance. Ensemble output is the class label associated with the highest aggregated classifier weight. For example, consider a data instance $x_S \in \mathcal{S}$. Let the ensemble contain 3 classifiers $[\mathcal{M}^{(1)}, \mathcal{M}^{(2)}, \mathcal{M}^{(3)}]$ with corresponding source weights $W_S = [0.2, 0.9, 0.6]$. If the prediction of these 3 classifiers for B_S are $\hat{Y} = [y^{(1)}, y^{(2)}, y^{(1)}]$ respectively, the ensemble score for class $y^{(k)}$ is $\sum_{i=1}^{|\mathcal{M}|} w_S^{(i)}/|\mathcal{M}|$ where i is the index for which the predicted label is $y^{(k)}$. In this case, the score for $y^{(1)}$ is 0.26 and $y^{(2)}$ is 0.30. Since $0.26 < 0.30$, the ensemble prediction is $y^{(2)}$. This method is used by *getError* and *predict* in Algorithm 1. Moreover, *getError* checks the error on B_S and returns a 1 if the predicted label is equal to the true label of a data instance, and 0 otherwise. Whereas, *predict* returns the ensemble score along with the predicted class label.

4.3 Drift Detection

A drift along $\mathcal{S}$ may affect a source-classifier's performance. We denote this as *within-stream* drift. Meanwhile, a drift in $\mathcal{S}$ or $\mathcal{T}$ may affect a target-classifier's performance. We denote this as *across-stream* drift. In both cases, a drift causes change in the data distribution. This is handled by training a new classifier and updating the ensemble.

4.3.1 Window Management

We maintain two sliding windows (feedback buffers) W_S and W_T that monitor ensemble feedback on recent data instances from $\mathcal{S}$ and $\mathcal{T}$ respectively. It is well known that classifier accuracy decreases due to concept drift [9]. Moreover, we show in Section 5.2 that classifier confidence also decreases due to drifting concepts. Therefore, *checkDrift* in Algorithm 1 is used to detect significant change in W_S or W_T, indicating a concept drift in the corresponding stream.

Supervised Window Management.
Class label of a data instance in $\mathcal{S}$ is predicted using the ensemble, whose resulting error $e \in \{0, 1\}$ is inserted into W_S, as mentioned in Section 4.2. Since each entry in W_S is either success or failure, it follows a Bernoulli distribution, while n such values in W_S follow a binomial distribution.

Unsupervised Window Management.
Unlike W_S, W_T contains ensemble confidence scores resulting from classification of data instances from $\mathcal{T}$, as indicated in Section 4.2. Confidence values are generated within the range of $[0, 1]$. Since classifier confidence scores are typically high until there is a concept drift, and decreases thereafter, these can be modeled by a *beta* distribution. Beta distribution has two parameters, i.e., α and β. The distribution is symmetric if $\alpha = \beta$, or skewed otherwise.

4.3.2 Change Detection

We invoke a change point detection method to check for significant changes in W_S and W_T. Particularly, we propose a CUSUM-type change detection technique [10] (CDT) based on binomial and beta distribution.

W_h (where $h \in \{\mathcal{S}, \mathcal{T}\}$) is divided into every possible pair of sub-windows W_h^b and W_h^a having at least γ number of values, where $h \in \{S, T\}$. Next, distribution of values stored in each split is estimated. At least γ values are needed for a

Algorithm 2: *checkDrift* : Concept Drift Detection

```
Input: $\alpha_d$: Drift Sensitivity, $\gamma$: Cushion period size,
       $W_h$: The dynamic sliding window.
Result: The change point if exists; $-1$ otherwise.
begin
    $Threshold \leftarrow -log(\alpha_d)$, $n \leftarrow$size of $W_h$, and $\omega_n \leftarrow 0$
    if $n \leq \mathcal{S}_{max}$ & $mean(W_h[1:n]) > 0.5$ then
        for $q \leftarrow \gamma : n - \gamma$ do
            Estimate pre and post-change distribution
             parameters, $\boldsymbol{\theta}_a$ and $\boldsymbol{\theta}_b$ from $W_x[1:q]$ and
             $W_h[q+1:n]$ respectively.
            Calculate $s(q,n)$ using Equation (3).
        $\omega_n = \max_{\gamma \leq q \leq n-\gamma} s(q,n)$
        if $\omega_n \geq Threshold$ then
            Return $q_m$, where $s(q,n) = \omega_n$.
        else
            Return $-1$.
    else
        Return 0.
```

good estimation of the distribution of values stored in each sub-window. We use $\gamma = 100$, which is also widely used in the literature due to *consistency* and *asymptotic normality* properties. Finally, a change point is detected based on the sum of log likelihood ratios.

Algorithm 2 sketches the proposed CDT. If at any point of time, average feedback is below 0.5, or size of the window exceeds $\mathcal{S}_{max}$, the ensemble classifier is updated immediately regardless of any change in the distribution. Otherwise, CDT divides the window W_h for each q between γ and $n-\gamma$, where n is the total number of observations in W_h and γ is the cushion size. The cushion ensures that each sub-window contains at least γ number of instances. $W_h^b = W[1:q]$ contains relatively older observations and $W_h^a = W[q+1:n]$ contains recent ones. CDT then estimates corresponding distributions represented by W_h^b and W_h^a. Since values in W_S follow a binomial distribution, two parameters n and p are estimated for values stored in corresponding sub-windows. The number of trials n is the sub-window size, and the probability of success p is the mean value in that sub-window. Values stored in W_T follow a beta distribution, hence corresponding parameters are estimated using method of moments.

Let $\boldsymbol{\theta}_b$ and $\boldsymbol{\theta}_a$ be the estimated distribution parameters from W_h^b and W_h^a respectively. The sum of log likelihood ratios is calculated using:

$$s(q,n) = \sum_{i=q+1}^{n} log\left(\frac{P\left(W_h[i] \mid \boldsymbol{\theta}_a\right)}{P\left(W_h[i] \mid \boldsymbol{\theta}_b\right)}\right) \tag{3}$$

where $P\left(W_h[i] \mid \boldsymbol{\theta}\right)$ is the probability density function, given a set of parameters $\boldsymbol{\theta}$, applied on the i^{th} instance stored in W_h. Finally, using all values in W_h, the CUSUM process score is calculated by $\omega_n = \max_{\gamma \leq q \leq n-\gamma} s(q,n)$. Let q_m be the value of q corresponding to the largest $s(q,n)$ value, where $\gamma \leq q \leq N - \gamma$. If w_n is greater than a user-defined threshold, then a change point is detected at q_m. We fix the threshold based on drift sensitivity α_d. In our experiments, we use $-log(\alpha_d)$ as the threshold value.

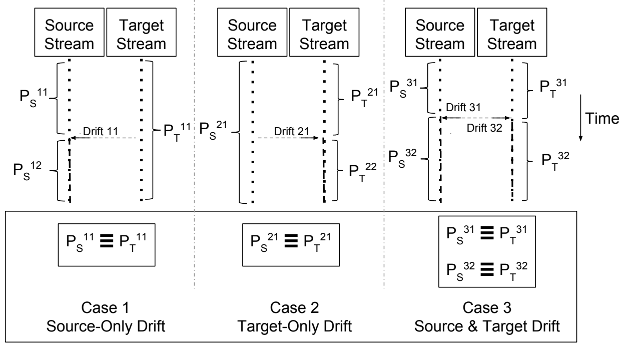


Figure 3: Illustration of Drift Adaptation cases.

4.4 Drift Adaptation

Once a drift is detected on a stream, new classifiers are trained appropriately using a minibatch of recent data instances that represent a new data distribution. However, training both types of classifiers (i.e. source-classifier and target-classifier) at every drift may not be necessary. With the goal of classifying target data instances, we observe three distinct types of changes in data distribution that occurs between training and test data when a drift is detected at time t. Figure 3 illustrates the intuition of deciding the type of classifier needed to be trained at t. We initially assume that the covariate shift assumptions hold between the source and target streams. Any other combinations of data drifts detected along $\mathcal{S}$ and $\mathcal{T}$ can be viewed as an application of these three cases in series with appropriate order.

The first case illustrated shows a drift detected only on $\mathcal{S}$, denoted as `Drift 11`. Data distribution on $\mathcal{S}$ before the drift is denoted as P_S^{11}, and after the drift is denoted as P_S^{12}. Meanwhile, P_T^{11} is the data distribution of $\mathcal{T}$ throughout, indicating no drift. Since future target data instances can be classified using an existing target-classifier in the ensemble, a new target-classifier trained on P_S^{12} and P_T^{11} may not be necessary. The satisfiability of covariate shift assumption is denoted by $P_S^{11} \equiv P_T^{11}$. However, a source-classifier trained on P_S^{12} is required for concept drift adaptation within $\mathcal{S}$, aiding future drift detection. We call this *source-only* drift.

Equivalently, the second case illustrated shows a drift detected only on $\mathcal{T}$, denoted by `Drift 21`. We call this as a *target-only* drift. Here, only a new target-classifier is needed for predicting class labels of newer data instances resulting from distribution P_T^{22}. The target-classifier therefore corrects the bias of P_S^{21} towards P_T^{22}.

Finally, drifts may occur simultaneously on $\mathcal{S}$ and $\mathcal{T}$. This is illustrated as case 3 in the figure. Here, both a source-classifier and a target-classifier may be required to address the issues discussed above in case 1 and case 2. A source-classifier is trained using data from $\mathcal{S}$ having distribution P_S^{32}, whereas a target-classifier is trained using data from $\mathcal{S}$ and $\mathcal{T}$ having distribution P_S^{32} and P_T^{32} respectively.

Furthermore at each detected drift in a stream, the associated data buffer (B_S or B_T) and feedback buffer (W_S or W_T) of the stream are updated by removing data instances before the change point, thereby simulating dynamic sized minibatches. This is performed using *updateMinibatch* in Algorithm 1. Note that the size of a data buffer and a feedback buffer of a stream are equal at all times. Therefore, a change point associated with the feedback buffer has an equivalent index on the corresponding data buffer.

4.5 Ensemble Update

We use a fixed-size ensemble consisting of weighted source and target classifiers. As noted earlier, a new classifier is added to the ensemble once a drift is detected. When the number of classifiers in the ensemble reach a user-defined maximum limit, denoted as E_{max}, the least desirable classifier in the ensemble is replaced by a newly constructed classifier. Intuitively, the least desirable classifier is the one having the least confidence on $\mathcal{T}$ since it may result in higher classification error on $\mathcal{T}$.

The workflow of *updateEnsemble* in Algorithm 1 is as follows. Before adding a new model to the ensemble, weights of existing classifiers are recomputed using the latest B_S and B_T appropriately. These updated weights are then used in selecting the least desired classifier to be replaced if the ensemble size is E_{max}. A priority queue containing W_S with corresponding indicies is formed. An array of indices R_S is obtained from this priority queue which is sorted in a non-increasing order of source weights. Similarly, R_T is obtained from a priority queue containing W_T and its index, sorted in an non-decreasing order of target weights. A classifier with the least target weight while not having the highest source weight is found to be replaced.

5. ANALYSIS

In this section, we analyze time and space requirements of the proposed algorithm, and provide theoretical properties.

5.1 Complexity Analysis

At every iteration of *MSC*, a new data instance is obtained from $\mathcal{S}$ and $\mathcal{T}$ whose class labels are predicted. The time and space complexity of this operation depends on the base classifier used. In general, training complexity of most well-known parametric models such as SVM surpass label prediction complexity. Therefore, we focus on classifier training and drift detection mechanism for complexity analysis.

Source-classifier is essentially a base classifier (e.g. SVM) trained on B_S. Whereas, a target-classifier consists of two parts, KMM and base classifier training. The time complexity of KMM is given by $\mathcal{O}(n_{tr}^3 + n_{tr}^2 v + n_{tr} n_{te} v)$, and its corresponding space complexity is $\mathcal{O}(n_{tr}^2 + n_{tr} n_{te})$ [18]. In the worst case, n_{tr} and n_{te} are at most S_{max}, resulting in $\mathcal{O}(n^3)$ time complexity on $n = |B_S|$. Here, we assume that $\mathcal{O}(n^3)$ dominates time complexity of base classifier training, and $\mathcal{O}(n^2)$ dominates its space complexity.

The drift detection mechanism also has $\mathcal{O}(n^3)$ time complexity [11], and $\mathcal{O}(n)$ space complexity. Therefore, overall time complexity of performing *MSC* is $\mathcal{O}(n^3)$ with $n = S_{max}$ is the size of the largest minibatch along the stream, with space complexity of $\mathcal{O}(n^2)$.

5.2 Base Classifier Properties

Choice of a base classifier for *MSC* should meet two criteria. First, the classifier should be able to perform kernel computations in RKHS since instance weights from KMM in target-classifier can only be used in RKHS [13]. Second, it should be able to provide a probability output that can be used as a confidence measure while predicting class labels on $\mathcal{T}$. Considering these two criteria, we choose Support Vector Machine (SVM) as a base classifier. Here, we show that the probability derived from distance of test instances to the discriminating SVM hyperplane follows the change in error rate due to concept drift, thereby forming a good confidence estimate required by *MSC*.

Let us assume that the stream contains instances from two classes, i.e., a and b. Let $f(x)$ be the unthresholded output of SVM on instance x. The hyperplane that separates class boundaries is defined by $f(x) = 0$. After a concept drift has occurred, more data instances may lie closer to the boundary in a max-margin SVM, thereby indicating a need for margin update.

According to Platt [21], class conditional densities, i.e., $P(f(x) \mid y)$, are exponentially distributed in real-world datasets when $f(x)$ is in the wrong side of the margin. Let us assume that the decision boundary of class a is in the positive side of the hyperplane, and that of class b is in the negative side. Therefore, $P(f(x) \mid y = a) = r_a e^{-r_a(1-f)}, f \le 1$, and $P(f(x)|y = b) = r_b e^{-r_b(f-1)}, f \ge -1$ [21]. Here, r_a and r_b are parameters of corresponding exponential distribution. This has inspired the probability to be estimated using a parametric form of a sigmoid as follows-

$$P(y = a \mid f(x)) = \frac{1}{1 + \exp(Af(x) + B)} \tag{4}$$

where $A = -(r_a + r_b)$ and $B = r_a - r_b + \ln \frac{P(y=b)}{P(y=a)}$. Since $r_a, r_b \ge 0$, $A < 0$. Therefore, monotonicity of (4) in $f(x)$ can be assured according to [21].

Lemma 1. *Significant change in the distribution of confidence scores indicate occurrence of a concept drift.*

Let us assume without loss of generality that $x_a^{(t)}$ is an instance generated from class boundary of a at time t. Let us also assume that there is a $\delta_a^{(t+1)} > 0$ displacement of the class boundary towards the hyperplane at time $t + 1$. Let $x_a^{(t+1)}$ be the point corresponding to $x_a^{(t)}$ after the change of concept. There are two possible cases.

Case 1: Both $x_a^{(t)}$ and $x_a^{(t+1)}$ are from the same side of the hyperplane. Here,

$$\begin{aligned} &f(x_a^{(t)}) = f(x_a^{(t+1)}) + \delta_a^{(t+1)} \Rightarrow f(x_a^{(t)}) > f(x_a^{(t+1)}) \\ &\Rightarrow 1 + \exp\left(Af(x_a^{(t)}) + B\right) < 1 + \exp\left(Af(x_a^{(t+1)}) + B\right) \\ &\Rightarrow P(y = a|f(x_a^{(t)})) > P(y = a|f(x_a^{(t+1)})) \text{ [using (4)]} \end{aligned} \tag{5}$$

Therefore, the classifier will have lower confidence in classifying instance $x_a^{(t+1)}$ than that of $x_a^{(t)}$.

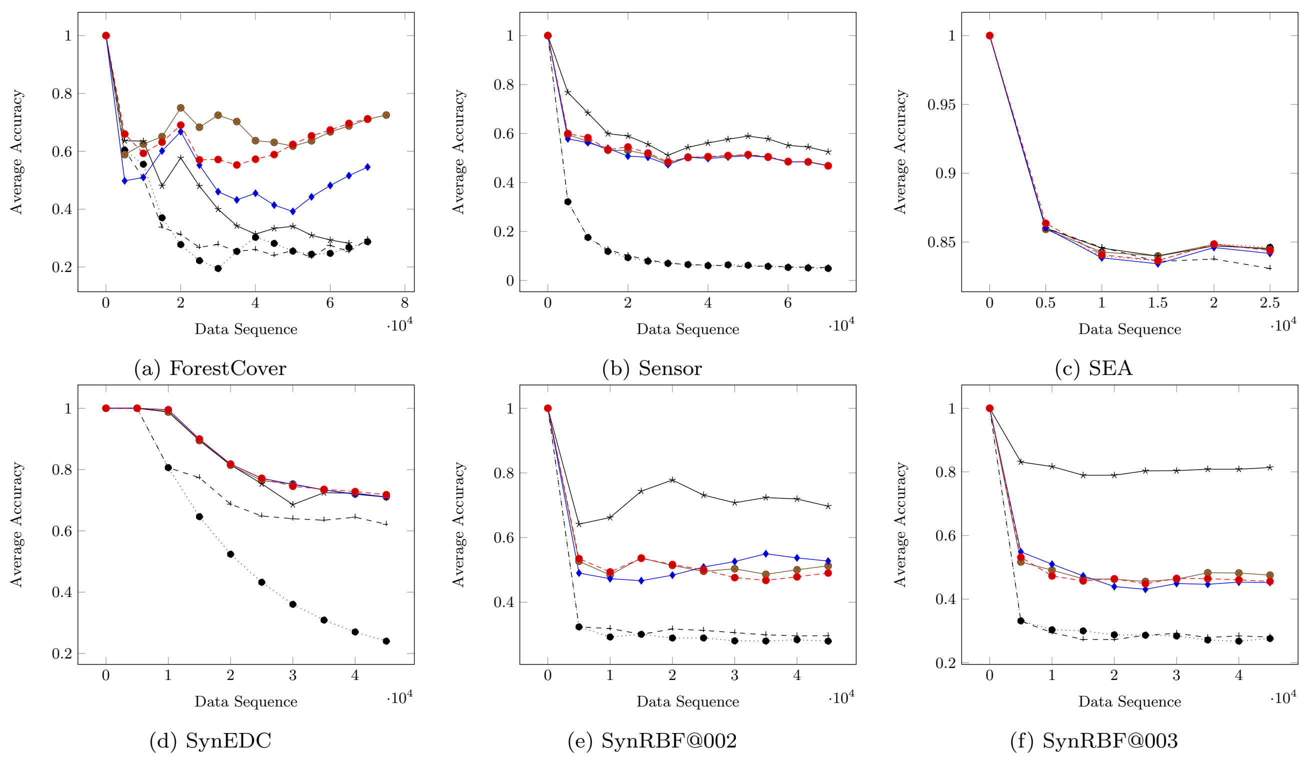
Case 2: If $x_a^{(t)}$ and $x_a^{(t+1)}$ are from different sides of the hyperplane, then,

$$\begin{aligned} &f(x_a^{(t)}) > 0 \;\&\; f(x_a^{(t+1)}) < 0 \Rightarrow f(x_a^{(t)}) > f(x_a^{(t+1)}) \\ &\Rightarrow P(y = a|f(x_a^{(t)})) > P(y = a|f(x_a^{(t+1)})) \text{ [Similar to (5)]} \end{aligned}$$

In both cases, classifier confidence on an instance decreases due to a concept drift. Therefore, a significant change in the distribution of confidence scores indicates occurrence of a concept drift. While this is true for a binary class problem, it can be applied to multiclass SVM which uses binary classification to perform pairwise computations [21, 26]

6. EMPIRICAL EVALUATION

In this section, we describe the experimental setup and present empirical results on various datasets using *MSC*.



(a) ForestCover (b) Sensor (c) SEA

(d) SynEDC (e) SynRBF@002 (f) SynRBF@003

Figure 4: Average accuracy : ⋯●⋯ *sKMM*; - + - *mKMM-5k*; —●— *MSC*; —★— *MSC2*; —◆— *srcMSC*; - ● - *trgMSC*.

Dataset	# features	# classes	# instances
ForestCover	53	7	146,438
Sensor	5	58	150,000
SEA	3	3	58,000
SynEDC	40	20	98,816
SynRBF@002	70	7	98,000
SynRBF@003	70	7	98,686

Table 2: Datasets

6.1 Datasets

We use 3 real-world datasets and 3 synthetic datasets for evaluation of the proposed approach. Table 2 lists these datasets with corresponding properties. The publicly available real-world datasets are ForestCover [1], Sensor [1] and SEA [22]. The other three synthetic data with concept drift were created using the well-known MOA framework [4].

We generate a biased source stream in each dataset, using a method similar to previous studies [13], as follows. First, we detect concept drifts by employing a Naïve Bayes classifier to predict class labels, and monitor its performance using *ADWIN*, similar to [3]. A minibatch is constructed from data instances between the points at which *ADWIN* detects a significant change in performance. Following [13], we first compute sample mean $\bar{x}$ of a minibatch, and divide it into two equal halves such that one half of data instances are selected according to the probability $e^{-|x-\bar{x}|^2}$ to form the biased source minibatch, while the rest forms the target minibatch. Each of these source minibatches are concatenated to form the source stream, and target minibatches concatenated to form the target stream.

[1]http://www.cse.fau.edu/~xqzhu/stream.html

6.2 Experiments

6.2.1 Baseline

Since there exists no previous studies in the multistream setting, we devise two baseline methods by applying a naive single classifier which only performs covariate shift correction between the source and target streams.

In the first baseline method, we train a single target-classifier on an initial set of source and target data instances used during the initialization phase of Algorithm 1. This trained model is then merely used to classify all the available instances in $\mathcal{T}$. We denote this as *single-KMM* or *sKMM*.

For the next baseline method, we follow the initial training setup of sKMM to form a single target-classifier. However, a new target-classifier is trained periodically using the latest S_{max} data instances occurring in $\mathcal{S}$ and $\mathcal{T}$. Every new classifier replaces the existing one and is used for classification of the next S_{max} target instances. Since multiple classifiers are trained along the stream, we denote this method as *multiple-KMM* or *mKMM*.

6.2.2 Variants

We now describe 3 variants of the proposed *MSC* method to evaluate our design decisions. Recall that *MSC* uses two types of classifiers in a single ensemble, i.e., source-classifier and target-classifier. Since each classifier has a source and target weight, they can be used to evaluate a data instance occurring on any data stream, irrespective of their type. We design a multistream classifier that uses only source-classifiers in its ensemble, and refer to it as *source-MSC* and denote it by *srcMSC*. Here, a source-classifier is built at every detected drift, even when a drift is detected on $\mathcal{T}$. Concretely, each *buildTargetModel* in Algorithm 1 is replaced

Symbol	Description
sKMM	A single target classifier trained during initialization.
mKMM-5k	A single target classifier trained periodically after every 5,000 instances along the stream.
MSC	Proposed ensemble framework.
MSC2	MSC using separate source and target ensembles.
srcMSC	MSC using only source-classifiers.
trgMSC	MSC using only target-classifiers.

Table 3: Baseline Methods and Variants

Dataset	Method	Drift Sensitivity			Max Ensemble Size		
		0.05	0.10	0.15	3	5	7
ForestCover	MSC	0.728	0.772	0.724	0.775	0.728	0.718
	MSC2	0.299	0.340	0.614	0.257	0.586	0.511
Sensor	MSC	0.467	0.278	0.272	0.510	0.501	0.491
	MSC2	0.535	0.539	0.539	0.538	0.557	0.540
SEA	MSC	0.842	0.840	0.841	0.844	0.842	0.840
	MSC2	0.840	0.841	0.839	0.837	0.835	0.840
SynEDC	MSC	0.720	0.712	0.727	0.889	0.766	0.726
	MSC2	0.698	0.720	0.719	0.768	0.741	0.718
SynRBF@002	MSC	0.522	0.463	0.466	0.581	0.552	0.564
	MSC2	0.616	0.621	0.636	0.645	0.713	0.682
SynRBF@003	MSC	0.459	0.481	0.476	0.591	0.551	0.516
	MSC2	0.786	0.777	0.773	0.875	0.932	0.876

Table 4: Average Accuracy of MSC to different parameter values.

with a *buildSourceModel* to form *srcMSC*. Note that this is equivalent to performing classification over a single stream with data instances in $\mathcal{S}$ and $\mathcal{T}$ combined. Therefore, this also forms a baseline method. Similarly, we construct *targetMSC* or *trgMSC* by replacing each *buildSourceModel* in Algorithm 1 with *buildTargetModel*. In this case, the ensemble contains only target-classifiers.

In the third variant of *MSC*, we use two finite-size ensembles instead of a single ensemble E. Here, we construct an ensemble E_S which contains only source-classifiers. This is used to classify data instances in $\mathcal{S}$. Similarly, we construct another ensemble E_T which contains only target-classifiers, and is used to classify data instances in $\mathcal{T}$. Since we separate the two types of classifiers using two ensembles and update them accordingly, we call this method as *MSC2*.

6.2.3 Setup

Table 3 summarizes each competing methods. We use the weighted LibSVM library [5] with $C_{svm} = 1.3 \times 10^5$, $\gamma_{svm} = 1 \times 10^{-4}$ and an RBF Kernel as the base classifier. We use KMM for covariate shift correction. The quadratic program of Equation 2 is evaluated using the CVXOPT python library [7]. Since it has a time complexity of $\mathcal{O}(n_{tr}^3)$, we limit the size of B_S to 1000 by sampling uniformly at random whenever a target-classifier is to be trained. Moreover, parameter values of KMM is chosen according to [13]. We use $B_{kmm} = 1000$ and $\epsilon_{kmm} = \frac{\sqrt{n_{tr}}-1}{\sqrt{n_{tr}}}$, and γ_{kmm} as the median of pairwise distances in the training set.

The *MSC* approach (and its variants) involves multiple parameters including ensemble size (denoted by E_{max}) and drift sensitivity (denoted by α_d). We empirically study parameter sensitivity of the framework over all datasets. Particularly, we vary E_{max} by setting it to $\{3, 5, 7\}$, and α_d to $\{0.05, 0.1, 0.15\}$. For comparison with baseline methods, we consider $S_{max} = 5000$, $E_{max} = 10$, and $\alpha_d = 0.001$ appropriately.

6.3 Results

Figure 4 shows the progress of average accuracy along the target stream $\mathcal{T}$ for all competing methods. Clearly, *MSC* and its variants outperform both baseline methods on all datasets by a significant margin, recovering from performance degradation, when necessary, using the drift detection mechanism. For example, average accuracy of both *sKMM* and *mKMM-5k* method on the Sensor dataset drastically reduces to about 10%. Whereas, *MSC* and *MSC2* methods result in an accuracy of around 55%. Note that the baseline methods does not use any drift detection mechanism. Moreover, *srcMSC* and *trgMSC* also result in higher accuracy than baseline methods while using a single type of classifiers, supporting the hypothesis that ensemble methods yield better results than single classifier method in the multistream setting, similar to a single stream setting [25].

Among the variants of the proposed framework, *MSC* and *MSC2* performs better than *srcMSC* and *trgMSC* on most datasets. Particularly, *MSC* performs best in the ForestCover and SynEDC datasets, while *MSC2* performs best in Sensor, SEA, SynRBF@002 and SynRBF@003 datasets. For instance, average accuracy along the stream by *MSC2* in SynRBF@002 dataset is about 70%, compared to 50% resulting from other variants and 30% from the baseline methods. However, simultaneous use of two different types of classifiers in *MSC* and *MSC2* consistently perform better than *srcMSC* and *trgMSC*, which use only a single type of classifier. Asynchronous drifts introduced due to a biased selection method in creating the source and target stream may not always satisfy covariate shift assumptions. Classification using both source and target classifiers are helpful in addressing this uncertainty in data behavior.

In the next set of experiments, we measure parameter sensitivity of the proposed approach. Particularly, the value of ensemble size E_{max} and drift detection sensitivity α_d is varied. For each value of E_{max} and α_d, the average accuracy obtained on each dataset when using *MSC* and *MSC2* is reported in Table 4. From the table, it can be observed that *MSC* and *MSC2* are marginally sensitive to different values of α_d on most datasets. A similar pattern can be observed while varying E_{max} as well. This shows that optimal value of parameters to obtain desired accuracy depends on the dataset properties. Moreover, greedy replacement of non-performing classifiers in the fixed-size ensemble may affect overall accuracy, especially in *MSC* where a single type of classifier may become dominant in the ensemble over time if drifts occur more frequently in one stream ($\mathcal{S}$ or $\mathcal{T}$) than the other. This behavior is reflected in Figure 4 where *MSC2* results in higher accuracy than *MSC* on most datasets. Nevertheless, optimal parameter values can be tuned using cross-validation. We leave this for future work.

On an average over all datasets, the time taken to classify a target instance by *MSC* was $0.45s$, and by *MSC2* was $0.34s$. The cubic time complexity of KMM and CDT can be improved by using distributed systems and dynamic programming [11]. We leave this for future work as well.

7. CONCLUSION

In this paper, we introduce the multistream classification problem consisting of two independent non-stationary data streams. Labeled information available in a source stream is used to classify unlabeled data on the target stream. We discuss challenges in this setting where asynchronous data drift occurs due to independent changes in data distribution within and across the two streams. We proposed a novel framework using covariate shift correction and drift detection techniques to address these challenges. Empirical results show that our approach achieves significantly better performance on real world and synthetic datasets compared to baseline techniques for multistream classification.

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, FA9550-14-1-0173.

8. REFERENCES

[1] S. D. Bay, D. F. Kibler, M. J. Pazzani, and P. Smyth. The uci kdd archive of large data sets for data mining research and experimentation. In *SIGKDD Explorations*, 2000.

[2] S. Ben-David, J. Blitzer, K. Crammer, A. Kulesza, F. Pereira, and J. W. Vaughan. A theory of learning from different domains. *Machine learning*, 79(1-2):151–175, 2010.

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

[4] 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.

[5] C.-C. Chang and C.-J. Lin. Libsvm: a library for support vector machines. *ACM Transactions on Intelligent Systems and Technology (TIST)*, 2(3):27, 2011.

[6] C. Cortes, M. Mohri, M. Riley, and A. Rostamizadeh. Sample selection bias correction theory. In *Algorithmic learning theory*, pages 38–53. Springer, 2008.

[7] J. Dahl and L. Vandenberghe. Cvxopt: A python package for convex optimization. In *Proc. eur. conf. op. res*, 2006.

[8] W. Dai, G.-R. Xue, Q. Yang, and Y. Yu. Co-clustering based classification for out-of-domain documents. In *Proceedings of the 13th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 210–219. ACM, 2007.

[9] J. Gama, P. Medas, G. Castillo, and P. Rodrigues. Learning with drift detection. In *Advances in artificial intelligence–SBIA 2004*, pages 286–295. Springer, 2004.

[10] J. Gama, I. Žliobaitė, A. Bifet, M. Pechenizkiy, and A. Bouchachia. A survey on concept drift adaptation. *ACM Computing Surveys (CSUR)*, 46(4):44, 2014.

[11] A. Haque, L. Khan, and M. Baron. Sand: Semi-supervised adaptive novel class detection and classification over data stream. In *Thirteenth AAAI Conference on Artificial Intelligence*, pages 1652–1658, Feb 2016.

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

[13] J. Huang, A. Gretton, K. M. Borgwardt, B. Schölkopf, and A. J. Smola. Correcting sample selection bias by unlabeled data. In *Advances in neural information processing systems*, pages 601–608, 2006.

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

[15] E. Kouloumpis, T. Wilson, and J. D. Moore. Twitter sentiment analysis: The good the bad and the omg! *Icwsm*, 11:538–541, 2011.

[16] B. Li, Q. Yang, and X. Xue. Transfer learning for collaborative filtering via a rating-matrix generative model. In *Proceedings of the 26th Annual International Conference on Machine Learning*, pages 617–624. ACM, 2009.

[17] 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.

[18] Y.-Q. Miao, A. K. Farahat, and M. S. Kamel. Ensemble kernel mean matching. In *Data Mining (ICDM), 2015 IEEE International Conference on*, pages 330–338. IEEE, 2015.

[19] S. J. Pan and Q. Yang. A survey on transfer learning. *Knowledge and Data Engineering, IEEE Transactions on*, 22(10):1345–1359, 2010.

[20] S. J. Pan, V. W. Zheng, Q. Yang, and D. H. Hu. Transfer learning for wifi-based indoor localization. In *Association for the advancement of artificial intelligence (AAAI) workshop*, page 6, 2008.

[21] J. C. Platt. Probabilistic outputs for support vector machines and comparisons to regularized likelihood methods. In *Advances in Large Margin Classifiers*, pages 61–74. MIT Press, 1999.

[22] W. N. Street and Y. Kim. A streaming ensemble algorithm (sea) for large-scale classification. In *Proceedings of the seventh ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 377–382. ACM, 2001.

[23] M. Sugiyama, S. Nakajima, H. Kashima, P. V. Buenau, and M. Kawanabe. Direct importance estimation with model selection and its application to covariate shift adaptation. In *Advances in neural information processing systems*, pages 1433–1440, 2008.

[24] V. N. Vapnik. *Statistical Learning Theory.* Wiley-Interscience, 1998.

[25] H. Wang, W. Fan, P. S. Yu, and J. Han. Mining concept-drifting data streams using ensemble classifiers. In *Proceedings of the Ninth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '03, pages 226–235, New York, NY, USA, 2003. ACM.

[26] T.-F. Wu, C.-J. Lin, and R. C. Weng. Probability estimates for multi-class classification by pairwise coupling. *J. Mach. Learn. Res.*, 5:975–1005, Dec. 2004.

[27] Y.-l. Yu and C. Szepesvári. Analysis of kernel mean matching under covariate shift. In *Proceedings of the 29th International Conference on Machine Learning (ICML-12)*, pages 607–614, 2012.