FUSION - An Online Method for Multistream Classification

Ahsanul Haque
The University of Texas at Dallas
axh129430@utdallas.edu

Zhuoyi Wang
The University of Texas at Dallas
zxw151030@utdallas.edu

Swarup Chandra
The University of Texas at Dallas
src093020@utdallas.edu

Bo Dong
The University of Texas at Dallas
bxd130630@utdallas.edu

Latifur Khan
The University of Texas at Dallas
lkhan@utdallas.edu

Kevin W. Hamlen
The University of Texas at Dallas
hamlen@utdallas.edu

ABSTRACT

Traditional data stream classification assumes that data is generated from a single non-stationary process. On the contrary, multistream classification problem involves two independent non-stationary data generating processes. One of them is the source stream that continuously generates labeled data. The other one is the target stream that generates unlabeled test data from the same domain. The distribution represented by the source stream data is biased compared to that of the target stream. Moreover, these streams may have asynchronous concept drifts between them. The multistream classification problem is to predict the class labels of target stream instances by utilizing labeled data from the source stream. This kind of scenario is often observed in real-world applications due to scarcity of labeled data. The only existing approach for multistream classification uses separate drift detection on the streams for addressing the asynchronous concept drift problem. If a concept drift is detected in any of the streams, it uses an expensive batch technique for data shift adaptation. These add significant execution overhead, and limit its usability. In this paper, we propose an efficient solution for multistream classification by fusing drift detection into online data shift adaptation. We study the theoretical convergence rate and computational complexity of the proposed approach. Moreover, empirical results on benchmark data sets indicate significantly improved performance over the baseline methods.

KEYWORDS

Multistream Classification; Data Shift adaptation; Direct Density Ratio Estimation; Asynchronous Concept Drift

1 INTRODUCTION

Data stream mining has attracted researchers due to its importance in today's connected digital world. Streams of data are continuously generated by a variety of sources - social networks, online businesses, military surveillance, to name a few. Efficient extraction of knowledge from these streams may help in taking important decisions in (near) real time, and unveil hidden opportunities. However, data stream mining is a challenging task due to its properties, such as change of underlying class boundaries (also known as concept drift), limited labeled data, delayed labeling, etc. [15].

Data stream mining researchers have so far focused on mining a single stream of data. Even if data is received from more than one stream simultaneously, all of them are assumed to be generated from a non-stationary data generating process [3]. Any change in the data generating process would affect data distributions in these streams simultaneously. Therefore, all such streams can be combined into a single stream, as individual streams represent the same distribution. However, combining streams may not be effective in particular scenarios, especially if individual streams represent different distributions with asynchronous and independent concept drifts among them. This type of scenarios may arise if data is generated by two different, but related non-stationary processes.

For example, consider building a model for predicting sentiment of tweets [13]. Typically, the sentiment is not provided as the ground truth along with a tweet. So, in order to collect training data, a few users may agree to provide tweets with sentiment label information. On the contrary, tweets on which the model needs to analyze the sentiment may come from any Twitter user. Users providing the training data may represent only a small portion of the population. Therefore, if we assume two streams of data, one from the Twitter users providing labeled data, another from the whole population of Twitter users, a sampling bias may exist between the distributions represented by these streams of data. This type of data shift between streams of data is typically caused due to limited supervision, or lack of control over the data generating process [3].

A new problem setting called *Multistream Classification* has been introduced in [3] to address these scenarios. It involves two simultaneous streams of data. One of the streams, called the *source stream*, provides only labeled training data. The other stream, called the *target stream*, provides unlabeled test data. The classification task is to use the labeled data from the source stream for classifying unlabeled data from the target stream efficiently. As pointed out before, combining the two streams may result in a different overall distribution that inhibits available data patterns when they are considered individually. Moreover, independent and asynchronous concept drifts may occur in either of the streams over time. Therefore, traditional techniques for data stream mining may not be effective if applied to the combined stream.

The main challenge of multistream classification is to address data shift and independent asynchronous concept drifts between the source and target streams. In this paper, we propose an efficient approach for multistream classification. The approach uses two sliding windows for storing recent instances from the source and target streams. Data shift between the source and target stream is

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than ACM must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.
CIKM'17 , November 6–10, 2017, Singapore, Singapore
© 2017 Association for Computing Machinery.
ACM ISBN 978-1-4503-4918-5/17/11…$15.00
https://doi.org/10.1145/3132847.3132886

addressed by weighing each source instance based on the density ratio. Let $P_S(\cdot)$ and $P_T(\cdot)$ be the distributions represented by recent source and target data instances respectively. The density ratio for an instance $\boldsymbol{x}$ is defined by $\beta(\boldsymbol{x}) = \frac{P_T(\boldsymbol{x})}{P_S(\boldsymbol{x})}$. A Gaussian kernel model is used in the proposed approach for direct density ratio estimation. The model is updated online with incoming instances. An ensemble classifier is used for classification, where each model is trained on weighted source stream instances.

The proposed approach has an inherent capability of addressing asynchronous concept drifts in multistream classification. In addition to addressing data shifts, the proposed approach uses density ratios estimated by the Gaussian kernel model for detecting any change between distributions represented by weighted source and target stream data. If a significant change is detected, the Gaussian kernel model is updated. Subsequently, weights for the source stream labeled data are re-evaluated using the updated kernel model, and the ensemble classifier is updated. The efficiency of the proposed approach stems from the fact that it uses the same kernel model for addressing both data shift and asynchronous concept drift in multistream classification. Empirical results on benchmark data sets show that the proposed approach outperforms the existing multistream classification method in terms of both accuracy and execution time.

The main contributions of our work are as follows. (1) We present an efficient method for direct density ratio estimation in the multistream setting. The model used in this method is updated online. The density ratios are used for data shift adaptation between the streams. We provide theoretical convergence rate for the proposed method. (2) We present a technique for detecting asynchronous concept drifts between source and target stream data using direct density ratios. (3) We propose an efficient approach for multistream classification by fusing asynchronous concept drift adaptation into data shift adaptation. We derive the time and space complexity of the proposed approach. (4) We use benchmark real-world and synthetic data sets to evaluate our approach, and compare the performance with the baseline methods. In addition to the only existing method for multistream classification, we use some of the acclaimed state-of-the-art data stream mining techniques as baseline methods.

The rest of the paper is organized as follows. We present a brief background on multistream classification in Section 2. We present the proposed approach in Section 3. In Section 4, we provide theoretical analysis on the approach, and present experiment results in Section 5. Finally, we conclude our discussion in Section 6.

2 BACKGROUND

In this section, we present a brief discussion on covariate shift adaptation and multistream classification. We also discuss some of the related work in this area.

2.1 Data Shift Adaptation

A fundamental assumption in data mining, known as the "stationary distribution assumption", is that both the training and test data represent the same data distribution [21]. However, it may be violated in real-world applications due to limited supervision, or lack of control over the data gathering process. Traditional techniques based on this assumption greatly suffer in this scenario.

Addressing an arbitrary difference between training and test distribution is a very difficult problem [9]. Hence, most approaches addressing this challenging assume that the training and test data distributions, denoted by $P_{tr}(\cdot)$ and $P_{te}(\cdot)$ respectively, are related by a covariate shift assumption. More specifically, the relationship between the training and test data distributions is such that $P_{tr}(y|\boldsymbol{x}) = P_{te}(y|\boldsymbol{x})$ and $P_{tr}(\boldsymbol{x}) \neq P_{te}(\boldsymbol{x})$, where $\boldsymbol{x}$ and y denote the set of covariate values and label of a data instance respectively.

In general, covariate shift between training and test data distributions is accounted by computing an importance weight, $\beta(\boldsymbol{x}) = \frac{P_{te}(\boldsymbol{x})}{P_{tr}(\boldsymbol{x})}$, for each training instance $\boldsymbol{x}$, and using them in the learning process. KMM [9], KLIEP [19], and uLSIF [10] are among the techniques that are available in the literature for handling covariate shift. However, these approaches work only on fixed-size training and test data. Although Kawahara and Sugiyama extended KLIEP for direct online density ratio estimation and sequential change point detection [11], it works on a single stream of data, where set of training/reference and test data instances are determined by a sliding window. In this paper, we consider multiple streams of data, where new data instance may arrive arbitrarily at any stream.

2.2 Multistream Classification

The vast majority of existing data stream classification techniques make two strong assumptions. First, true labels of data instances along the stream become available soon after prediction, which are then used for updating the existing classifier. In practice, labeled data are scarce as obtaining true labels is costly [7]. Furthermore, it is assumed that the training and the test data represent the same distribution. As mentioned in Section 2.1, this assumption may also be violated due to scarcity of labeled data. This may induce a sampling bias between the training and test data distribution. A new problem setting, called *Multistream Classification*, has been introduced in [3], where two data streams over the same domain are considered by relaxing the above assumptions.

2.2.1 *Problem Statement.*

Let us consider that two different but related processes generate data continuously from a domain $\mathcal{D}$. The first process operates in a supervised environment, i.e., all the data instances that are generated from the first process are labeled. On the contrary, the second process generates unlabeled data from the same domain. The stream of data generated from the above processes are called the *source stream* and the *target stream*, and are denoted by $\mathcal{S}$ and $\mathcal{T}$ respectively. Each data instance is denoted by $(\boldsymbol{x}, y)$, where $\boldsymbol{x} \in \mathcal{D}^d$ is the set of d covariates, and y is the true label of the instance. As mentioned before, only $\boldsymbol{x}$ for each instance is observed in $\mathcal{T}$, where $\mathcal{S}$ also provides y in addition to $\boldsymbol{x}$ for each instance. The *Multistream Classification* is defined as follows.

Definition 2.1. Let $\mathrm{X}_S \in \mathcal{D}$ be a set of d-dimensional vectors of covariates and Y_S be the corresponding class labels observed on a non-stationary stream $\mathcal{S}$. Similarly, let $\mathrm{X}_T \in \mathcal{D}$ be a set of d-dimensional vectors of covariates observed on another independent non-stationary stream $\mathcal{T}$. Let P_S and P_T denote covariate distribution from $\mathcal{S}$ and $\mathcal{T}$ respectively. Data generated from S and T are related by a covariate shift, i.e., $P_S(y|\boldsymbol{x}) = P_T(y|\boldsymbol{x})$ and $P_S(\boldsymbol{x}) \neq P_T(\boldsymbol{x})$. Construct a classifier $\mathcal{M}$ that predicts class label of $\boldsymbol{x} \in \mathrm{X}_T$ using X_S, Y_S and X_T.

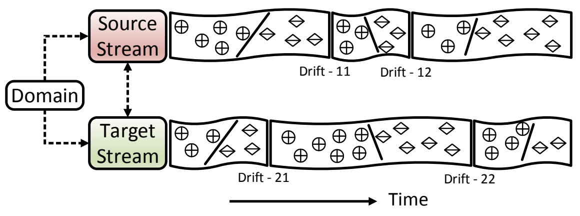


Figure 1: An example illustrating asynchronous data drifts

2.2.2 Challenges. As mentioned before, we assume that initially at time t, data distributions of $\mathcal{S}$ and $\mathcal{T}$ are related by a covariate shift, i.e., $P_S^{(t)}(y \mid \boldsymbol{x}) = P_T^{(t)}(y \mid \boldsymbol{x})$ and $P_S^{(t)}(\boldsymbol{x}) \neq P_T^{(t)}(\boldsymbol{x})$. However, this assumption may not be true at time $r > t$ due to the non-stationary nature of data streams. The reason is that, individually within each data stream, the conditional probability distribution may change over time due to concept drift, i.e. $P^{(t)}(y \mid \boldsymbol{x}) \neq P^{(r)}(y \mid \boldsymbol{x})$. Similarly, a *covariate drift*, i.e., a change in the covariate distribution, may also occur with time in each stream.

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, referred to as *asynchronous* drift. Figure 1 illustrates asynchronous data drifts between the source and the target stream. In this illustration, four independent data drifts occur at different times on $\mathcal{S}$ and $\mathcal{T}$. The drifts *Drift-11* and *Drift-21* are similar, and occur on the source and target stream respectively but at a different times, which by definition is an asynchronous drift. Similarly, *Drift-12* and *Drift-22* represent another example of an asynchronous concept drift. This type of scenario may occur in a real-world application when the factor causing a drift affects the streams at different times. To summarize, the main challenges in multistream classification are handling data shift and asynchronous concept drift between the source and the target stream simultaneously and efficiently.

2.2.3 Prior Work. MSC (MultiStream Classifier) [3] is a framework that has recently been proposed for multistream classification. MSC uses Kernel Mean Matching (KMM) for covariate shift adaptation between the source and the target distributions by weighing labeled source instances in the learning process. However, since source or target stream may have asynchronous concept drifts, weights of the training instances may become outdated if there is a drift in any of these streams. To address this challenge, an ensemble of classifiers is maintained. The ensemble contains classifiers built on both source stream and target stream data. The ensemble is updated by calculating new weights for recent instances from the source data stream if a concept drift is detected in either of these streams of data. Concept drift is detected by monitoring any significant change in classifier feedback following similar approach proposed in [8]. Since source stream generates all labeled data, a concept drift in it is detected by tracking any major change in classifier error rate. On the contrary, classifier confidences are monitored to detect concept drifts in the target stream generating only unlabeled data.

While being the first approach for addressing the challenges of multistream classification, MSC suffers from a number of limitations. First, it uses an ensemble classifier consisting of models trained on both source and target data. Once there is a concept drift in either stream, the ensemble is updated using a model trained on data from the corresponding stream. As a result, the overall ensemble management is complex in MSC. Second, it executes change detection algorithm for detecting concept drift after receiving any new data instance either in the source or target stream. The change detection algorithm used in MSC has time complexity cubic in the number of data instances in the current window, which makes MSC extremely slow. Third, once a concept drift is detected in any of the streams, MSC uses Kernel Mean Matching for covariate shift adaptation, which also has cubic time complexity. The above overheads adversely affect the execution time of MSC.

3 THE PROPOSED APPROACH

In this paper, we propose a simple but efficient solution for multistream classification by fusing drift detection into covariate shift adaptation. We refer to this approach as FUSION (eFficient mUltiStream classification using direct densIty ratio estimatiON).

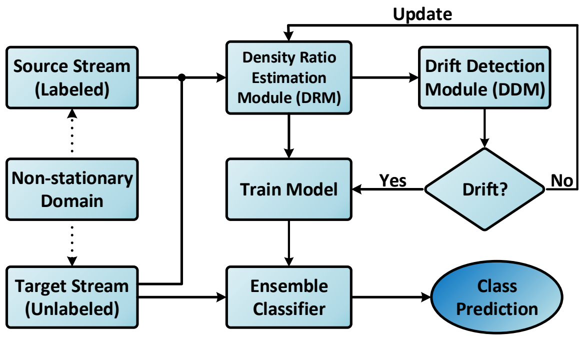


Figure 2: Overview of FUSION

Figure 2 and Algorithm 1 illustrate the core components in FUSION. It has four main modules, i.e., *Density Ratio Estimation (DRM)*, *Drift Detection (DDM)*, *Classification*, and *Update*. As mentioned in the problem statement, both source and target streams are generated from the same non-stationary domain, having asynchronous data drift between them. We use two fixed-size sliding windows to store recent instances from $\mathcal{S}$ and $\mathcal{T}$, referred to as the source and the target sliding window, and denoted as $\mathcal{W}_S$ and $\mathcal{W}_T$ respectively. The size of the sliding windows is denoted by N_m.

FUSION uses an ensemble classifier for classification. The first model in the ensemble is trained on the initial instances from the source sliding window. However, to correct possible covariate shift between $\mathcal{S}$ and $\mathcal{T}$, each instance from the sliding window is associated with an importance weight. FUSION uses density ratios estimated from the Density Ratio Estimation Module (DRM) as importance weights. Label for any new instance arriving in $\mathcal{T}$ is predicted by taking the majority voting from the ensemble classifier. DRM updates the model for density ratio estimation incrementally with each incoming instance in either sliding windows.

Table 1: Frequently used symbols

$\mathcal{D}$: Domain	$\mathcal{M}$: Ensemble Classifier
$\mathbb{P}$: Set of non-stationary processes	P_S: Source stream probability distribution
$\mathcal{S} \in \mathbf{P}$: A labeled source stream	P_T: Target stream probability distribution
$\mathcal{T} \in \mathbf{P}$: An unlabeled target stream	$\beta(\boldsymbol{x})$: Importance weight for $\boldsymbol{x}$
$\boldsymbol{W}_S$, $\boldsymbol{W}_T$: Source and Target Sliding window	L: The maximum allowable ensemble size
$\boldsymbol{x}$: d-dimensional features (or covariates)	$\boldsymbol{\alpha}$: The set of parameters of the Gaussian kernel model
$y \in \mathbf{Y}$: Class label of a data instance	N_m: The size of $\boldsymbol{W}_S$ and $\boldsymbol{W}_T$

Algorithm 1 FUSION: Multistream Classification

```
Input: Labeled source stream data $\mathcal{S}$, The size of sliding windows $N_m$.
Output: Labels predicted on $\mathcal{T}$ data.
1:  Read first $N_m$ instances from $\mathcal{S}$ and $\mathcal{T}$ into $\boldsymbol{W}_S$ and $\boldsymbol{W}_T$ respectively.
2:  Learn $\boldsymbol{\alpha} = \{\alpha_i\}_{i=1}^{N_m}$ using LearnAlpha (Algorithm 2).
3:  Estimate $\left\{\beta(\boldsymbol{W}_S^{(i)})\right\}_{i=1}^{N_m}$ (Section 3.2).
4:  Initialize $\mathcal{M}$ by learning a base model from $\boldsymbol{W}_T$, $\boldsymbol{W}_S$, and $\left\{\beta(\boldsymbol{W}_S^{(i)})\right\}_{i=1}^{N_m}$.
5:  repeat
6:      Receive a new instance $\boldsymbol{x}$.
7:      if $\boldsymbol{x} \in \mathcal{T}$ then
8:          Predict label for $\boldsymbol{x}$ by taking the majority voting.
9:      end if
10:     Slide the corresponding window ($\boldsymbol{W}_S$ or $\boldsymbol{W}_T$) for including the new instance.
11:     Update $\boldsymbol{\alpha}$ using UpdateAlpha (Algorithm 3).
12:     Check for any drift in data using DetectDrift (Algorithm 4).
13:     if DetectDrift returns True then
14:         Recalculate $\boldsymbol{\alpha} = \{\alpha_i\}_{i=1}^{N_m}$ using LearnAlpha
15:         Update $\mathcal{M}$ using UpdateClassifier (Algorithm 5).
16:     end if
17: until $\mathcal{T}$ exists
```

As the system keeps receiving new instances in $\mathcal{S}$ or $\mathcal{T}$, the Drift Detection Module (DDM) detects drift between the distributions represented by data in the sliding windows using density ratios estimated by DRM. If there is a drift, a new model is trained on source sliding window instances along with their updated importance weights. Moreover, the ensemble classifier and the sliding windows are updated.

Table 1 lists frequently used symbols in this paper. Throughout the paper, typically a bold symbol or letter is used to denote a set of elements, and a superscript is used to indicate the index of an element in the set. A subscript is used to indicate the association of an entity to a type. For example, $\boldsymbol{W}_S^{(i)}$ denotes the i^{th} data instance in the source sliding window. We present a detailed discussion about different modules of FUSION in rest of this section.

3.1 Density Ratio Estimation Module (DRM)

The Density Ratio Estimation module (DRM) of FUSION uses a Gaussian kernel model for direct density estimation. The model is updated incrementally as new instances appear in $\mathcal{S}$ or $\mathcal{T}$. In this section, we describe the DRM, and its online update procedure.

3.1.1 Gaussian Kernel Model. At a particular time, we define the source distribution P_S, and the target distribution P_T by the distributions represented by data instances in $\boldsymbol{W}_S$ and $\boldsymbol{W}_T$ respectively at that moment. Density ratio for an instance $\boldsymbol{x}$ is defined by $\beta(\boldsymbol{x}) = \frac{P_T(\boldsymbol{x})}{P_S(\boldsymbol{x})}$. If $\boldsymbol{x}$ is an instance from $\mathcal{S}$, $\beta(\boldsymbol{x})$ is used as its importance weight in the learning process. We will discuss more on training later in this section.

Using Gaussian kernel model, $\beta(\boldsymbol{x})$ is modeled as follows:

$$\hat{\beta}(\boldsymbol{x}) = \sum_{i=1}^{N_m} \alpha_i K_\sigma\left(\boldsymbol{x}, \boldsymbol{W}_T^{(i)}\right) \tag{1}$$

where $\boldsymbol{\alpha} = \{\alpha_i\}_{i=1}^{N_m}$ are parameters to be learned from data samples, and $K_\sigma(\cdot,\cdot)$ is a Gaussian kernel with kernel width σ, i.e., $K_\sigma(\boldsymbol{x}, \boldsymbol{x}') = \exp\left\{-\frac{\|\boldsymbol{x}-\boldsymbol{x}'\|^2}{2\sigma^2}\right\}$. The target sliding window instances, $\boldsymbol{W}_T$, works as the Gaussian centers. Each parameter α_i is associated to the i^{th} Gaussian kernel, i.e., i^{th} instance in $\boldsymbol{W}_T$. We choose kernel width σ by *likelihood cross validation* following [19]. Any other basis functions can also be used in place of Gaussian kernels in Eq. (1).

3.1.2 Learning Parameters. The target distribution is estimated by the weighted training distribution, $\hat{P_T}(\boldsymbol{x}) = \hat{\beta}(\boldsymbol{x})P_S(\boldsymbol{x})$. The parameters $\boldsymbol{\alpha} = \{\alpha_i\}_{i=1}^{N_m}$ in model (1) are learned so that the Kullback-Liebler divergence from $P_T(\boldsymbol{x})$ to $\hat{P_T}(\boldsymbol{x})$ would be minimized. This leads to the following convex optimization problem-

$$\begin{aligned}
&\underset{\{\alpha_i\}_{i=1}^{N_m}}{\text{maximize}} \left[\sum_{j=1}^{N_m} \log\left(\sum_{i=1}^{N_m} \alpha_i K_\sigma\left(\boldsymbol{W}_T^{(j)}, \boldsymbol{W}_T^{(i)}\right)\right)\right] \\
&\text{subject to } \frac{1}{N_m}\sum_{j=1}^{N_m}\sum_{i=1}^{N_m} \alpha_i K_\sigma\left(\boldsymbol{W}_S^{(j)}, \boldsymbol{W}_T^{(i)}\right) = 1, \\
&\text{and } \alpha_1, \alpha_2, \ldots, \alpha_{N_m} \geq 0.
\end{aligned} \tag{2}$$

Algorithm (2) outlines the steps for learning the parameters $\boldsymbol{\alpha}$ for the model in Eq. (1). First, Gaussian kernels are calculated for all $\boldsymbol{W}_T$ instances at Line 1. Next, gradient ascent is performed until convergence while satisfying the constraints at Lines 5-6. Once the set of parameters $\boldsymbol{\alpha}$ is learned from data, importance weight for any instance $\boldsymbol{x}$ is calculated using Eq. (1).

3.1.3 Updating Parameters Online. Since data continuously arrives in source and target streams, the model in Eq. (1) needs to be updated also by updating $\boldsymbol{\alpha}$. Kawahara and Sugiyama have

Algorithm 2 **LearnAlpha: Learn DRM Parameters**

```
Input: Source instances $\mathcal{W}_S = \left\{\mathcal{W}_S^{(i)}\right\}_{i=1}^{N_m}$, target instances $\mathcal{W}_T = \left\{\mathcal{W}_T^{(i)}\right\}_{i=1}^{N_m}$, the learning rate $\epsilon$, and the kernel width $\sigma$.
Output: DRM parameters $\boldsymbol{\alpha} = \{\alpha\}_{i=1}^{N_m}$.
1: $\mathbf{K}^{(i,j)} = K_\sigma(\mathcal{W}_T^{(i)}, \mathcal{W}_T^{(j)}); i,j = 1, \dots, N_m$.
2: $\mathbf{p}^{(j)} = \frac{1}{N_m}\sum_{i=1}^{N_m} K_\sigma(\mathcal{W}_S^{(i)}, \mathcal{W}_T^{(j)}); j = 1, \dots, N_m$.
3: Initialize $\boldsymbol{\alpha}$.
4: repeat
5:     Gradient ascent step:
       $\boldsymbol{\alpha} \leftarrow \boldsymbol{\alpha} + \epsilon\mathbf{K}(1./\mathbf{K})$.
6:     Satisfy constraints:
       $\boldsymbol{\alpha} \leftarrow \boldsymbol{\alpha} + (1 - \mathbf{p}^T\boldsymbol{\alpha})\mathbf{p}/(\mathbf{p}^T\mathbf{p})$,
       $\boldsymbol{\alpha} \leftarrow max(0, \boldsymbol{\alpha})$,
       $\boldsymbol{\alpha} \leftarrow \boldsymbol{\alpha}/(\mathbf{p}^T\boldsymbol{\alpha})$.
7: until convergence
8: Return $\boldsymbol{\alpha} = \{\alpha\}_{i=1}^{N_m}$.
```

proposed an online update method of $\boldsymbol{\alpha}$ in [11]. However, unlike multistream classification scenario, this method assumes only one stream of data with a sliding window to define the set of reference and test data instances. In this paper, we adapt this method for multistream classification scenario.

As mentioned before, $\mathcal{W}_T$ instances act as the Gaussian kernel centers. For each $K_\sigma(\cdot, \mathcal{W}_T^{(i)})$, there is a corresponding parameter α_i in the set $\boldsymbol{\alpha}$, which works as the weight for that Gaussian function. Therefore, if there is a new instance in the target stream, it affects the optimization problem in Eq. (2). So $\boldsymbol{\alpha}$ needs to be updated while satisfying constraints.

The online update method is based on the online learning technique for kernel methods proposed in [12]. Assuming that β is searched within a reproducing kernel Hilbert space $\mathcal{H}$, the following reproducing property holds-

$$\left\langle \beta(\cdot), K(\cdot, \boldsymbol{x}')\right\rangle = \beta(\boldsymbol{x}') \tag{3}$$

Let $E_i(\beta)$ be the empirical error for $\mathcal{W}_T^{(i)}$, $E_i(\beta) = -\log\beta(\mathcal{W}_T^{(i)})$. It can be observed from Eq. (2) that estimated density ratio $\hat{\beta}$ is calculated by minimizing $\sum_{i=1}^{N_m} E_i(\beta)$ under the constraints. Let $\tilde{E}_i(\beta)$ be the regularized empirical error, that is-

$$\tilde{E}_i(\beta) = -\log\beta(\mathcal{W}_T^{(i)}) + \frac{\lambda}{2}\|\beta\|_{\mathcal{H}}^2 \tag{4}$$

where $\lambda(> 0)$ is the regularization parameter, and $\|\beta\|_{\mathcal{H}}$ denotes the norm in $\mathcal{H}$ space.

Considering the reproducing property in Eq. (3), and the regularized empirical error shown in Eq. (4), the estimated density ratio ($\hat{\beta}$) can be updated using a new instance in the target stream, denoted by ($\mathcal{W}_T^{(N_m+1)}$) as follows-

$$\hat{\beta}' = \hat{\beta} - \eta\partial_\beta\tilde{E}_{N_m+1}(\hat{\beta}) \tag{5}$$

where η is the learning rate, and ∂_β denotes partial derivative with respect to β. Since we consider Gaussian kernel model (Eq. (1)), replacing the partial derivative in Eq. (5), we get-

$$\hat{\beta}' = \hat{\beta} - \eta\left(-\frac{K_\sigma\left(\cdot, \mathcal{W}_T^{(N_m+1)}\right)}{\hat{\beta}\left(\mathcal{W}_T^{(N_m+1)}\right)} + \lambda\hat{\beta}\right) \tag{6}$$

Using the Eq. (1), values in $\boldsymbol{\alpha}$ should therefore be updated as follows-

$$\begin{cases} \hat{\alpha}_i' \leftarrow (1-\eta\lambda)\hat{\alpha}_{i+1} & i = 1, \dots, N_m - 1 \\ \hat{\alpha}_i' \leftarrow \frac{\eta}{\hat{\beta}\left(\mathcal{W}_T^{(N_m+1)}\right)} & i = N_m \end{cases} \tag{7}$$

Algorithm 3 **UpdateAlpha: Update DRM Parameters**

```
Input: Source instances $\mathcal{W}_S = \left\{\mathcal{W}_S^{(i)}\right\}_{i=1}^{N_m}$, target instances $\mathcal{W}_T = \left\{\mathcal{W}_T^{(i)}\right\}_{i=1}^{N_m}$, new instance $\boldsymbol{x}$, the kernel width $\sigma$, the regularization parameter $\lambda$, and the learning rate $\eta$.
Output: Updated DRM parameters, $\boldsymbol{\alpha} = \{\alpha\}_{i=1}^{N_m}$.
1: if $\boldsymbol{x} \in \mathcal{S}$ then
2:     $\mathcal{W}_S^{(N_m+1)} \leftarrow \boldsymbol{x}$.
3:     $\mathbf{p}^{(j)} = \frac{1}{N_m}\sum_{i=1}^{N_m} K_\sigma(\mathcal{W}_S^{(i+1)}, \mathcal{W}_T^{(j)}), j = 1, \dots, N_m$.
4:     Go to Line 10.
5: end if
6: $\mathcal{W}_T^{(N_m+1)} \leftarrow \boldsymbol{x}$.
7: $\mathbf{p}^{(j)} = \frac{1}{N_m}\sum_{i=1}^{N_m} K_\sigma(\mathcal{W}_S^{(i)}, \mathcal{W}_T^{(j+1)}), j = 1, \dots, N_m$.
8: $\hat{\beta}(\mathcal{W}_T^{(N_m+1)}) = \sum_{i=1}^{N_m} \alpha_i K_\sigma(\mathcal{W}_T^{(N_m+1)}, \mathcal{W}_T^{(i)})$.
9: Update $\boldsymbol{\alpha}$ using Eq. (7).
10: Satisfy constraints:
    $\boldsymbol{\alpha} \leftarrow \boldsymbol{\alpha} + (1 - \mathbf{p}^T\boldsymbol{\alpha})\mathbf{p}/(\mathbf{p}^T\mathbf{p})$,
    $\boldsymbol{\alpha} \leftarrow max(0, \boldsymbol{\alpha})$,
    $\boldsymbol{\alpha} \leftarrow \boldsymbol{\alpha}/(\mathbf{p}^T\boldsymbol{\alpha})$.
11: if $\boldsymbol{x} \in \mathcal{S}$ then
12:     $\mathcal{W}_S^{(i)} \leftarrow \mathcal{W}_S^{(i+1)}, i = 1, \dots, N_m$.
13: else
14:     $\mathcal{W}_T^{(i)} \leftarrow \mathcal{W}_T^{(i+1)}, i = 1, \dots, N_m$.
15: end if
16: Return $\boldsymbol{\alpha} = \{\alpha\}_{i=1}^{N_m}$.
```

Algorithm 3 outlines the online updating of $\boldsymbol{\alpha}$. As discussed before, a new instance in the target stream changes the optimization problem in Eq. (2). Therefore, $\boldsymbol{\alpha}$ needs to be updated along with constraint satisfaction. On the contrary, if the new instance arrives in the source stream, it does not affect the optimization problem directly. However, the constraints may be violated due to the new instance. Therefore, the constraints need to be satisfied again. Subsequently, the corresponding sliding window is updated with the new instance.

3.2 Training and Classification

FUSION uses an ensemble classifier, denoted as $\mathcal{M}$. We start by loading the first N_m instances from $\mathcal{S}$ and $\mathcal{T}$ into $\mathcal{W}_S$ and $\mathcal{W}_T$ respectively, which are referred to as the warm-up period data. FUSION trains the first model in the ensemble using the warm-up period data. However, due to covariate shift between the source stream ($\mathcal{S}$) and the target stream ($\mathcal{T}$), importance weights for labeled source data should be considered in the learning process. These importance weights are estimated by the Density Ratio Estimation (DRM) module using warm-up period data from $\mathcal{W}_S$ and $\mathcal{W}_T$ as follows-

$$\hat{\beta}\left(\mathcal{W}_S^{(i)}\right) = \sum_{j=1}^{N_m} \alpha_j K_\sigma\left(\mathcal{W}_S^{(i)}, \mathcal{W}_T^{(j)}\right), i = 1, \ldots, N_m \qquad (8)$$

Any learning algorithm that incorporates importance weight of training instances can be used in FUSION. As new instances arrive in $\mathcal{S}$ or $\mathcal{T}$, the ensemble classifier $\mathcal{M}$ is updated if there is a drift to ensure that it represents the current concepts. A new base model is trained using data in $\mathcal{W}_S$ and $\mathcal{W}_T$ at that time. Drift detection and updating method used by FUSION will be discussed later in this section. FUSION predicts the majority voted class in the ensemble as the class of an incoming test instance from the target stream.

3.3 Drift Detection Module (DDM)

As mentioned before, $P_T(\boldsymbol{x})$ is estimated by $\hat{P_T}(\boldsymbol{x}) = \hat{\beta}(\boldsymbol{x})P_S(\boldsymbol{x})$. The classifier is updated following a drift, i.e., a significant difference between $P_T(\boldsymbol{x})$ and $\hat{\beta}(\boldsymbol{x})P_S(\boldsymbol{x})$. Let $\boldsymbol{\alpha}^0$ be the set of initial parameters. These parameters are updated online as new instances arrive in $\mathcal{S}$ or $\mathcal{T}$. Let $\boldsymbol{\alpha}^t$ be the set of parameters at time t. Let $\hat{\beta}_0$ and $\hat{\beta}_t$ are density ratios defined by $\boldsymbol{\alpha}^0$ and $\boldsymbol{\alpha}^t$ respectively. The following likelihood ratio measures the deviation of the weighted training distribution from the test distribution at time t.

$$S = \sum_{i=1}^{N_m} \ln \frac{P_T\left(\mathcal{W}_T^{(i)}\right)}{\hat{\beta}_0 P_S\left(\mathcal{W}_T^{(i)}\right)} = \sum_{i=1}^{N_m} \ln \frac{\hat{\beta}_t\left(\mathcal{W}_T^{(i)}\right)}{\hat{\beta}_0\left(\mathcal{W}_T^{(i)}\right)}$$

A drift is detected if $S > -\ln(\tau)$, where τ is a user defined parameter. It can be proved that the false alarm rate of the drift detection algorithm is bounded by τ. The efficiency of FUSION stems from the fact that in addition to estimating importance weights, it uses the same Gaussian kernel model for drift detection. Therefore, FUSION detects drift without adding any extra overhead.

Algorithm 4 DetectDrift: Drift Detection

```
Input: Target instances $\mathcal{W}_T = \left\{\mathcal{W}_T^{(i)}\right\}_{i=1}^{N_m}$, Set of initial parameters $\left\{\alpha_i^0\right\}_{i=1}^{N_m}$, Set of current parameter $\left\{\alpha_i^t\right\}_{i=1}^{N_m}$, The kernel width $\sigma$, and The parameter $\tau$.
Output: *True* if drift is detected, else *False*.
1: $\hat{\beta}_0(\mathcal{W}_T^{(i)}) = \sum_{j=1}^{N_m} \alpha_j^0 K_\sigma(\mathcal{W}_T^{(i)}, \mathcal{W}_T^{(j)})$ for $i = 1, \ldots, N_m$.
2: $\hat{\beta}_t(\mathcal{W}_T^{(i)}) = \sum_{j=1}^{N_m} \alpha_j^t K_\sigma(\mathcal{W}_T^{(i)}, \mathcal{W}_T^{(j)})$ for $i = 1, \ldots, N_m$.
3: $S = \sum_{i=1}^{N_m} \ln \frac{\hat{\beta}_t(\mathcal{W}_T^{(i)})}{\hat{\beta}_0(\mathcal{W}_T^{(i)})}$.
4: Return $S > -\ln(\tau)$.
```

Algorithm 4 sketches drift detection of FUSION. A drift is detected if the drift score S, i.e., the sum of log-likelihood ratios is greater than a pre-fixed threshold. As $\boldsymbol{\alpha}$ is updated online with any new instance in $\mathcal{S}$ or $\mathcal{T}$, both importance weight estimation and drift detection of FUSION are efficient.

3.4 Classifier Update

Algorithm 5 UpdateClassifier: Update the Classifier

```
Input: Target instances $\mathcal{W}_T = \left\{\mathcal{W}_T^{(i)}\right\}_{i=1}^{N_m}$, DRM parameters $\{\alpha_i\}_{i=1}^{N_m}$, the kernel width $\sigma$, and threshold $\tau$.
Output: The updated ensemble.
1: Get $\boldsymbol{\alpha}$ from $\mathcal{W}_S$ and $\mathcal{W}_T$ using Algorithm 2 and 3.
2: Calculate $\left\{\hat{\beta}(\mathcal{W}_S^{(i)})\right\}_{i=1}^{N_m}$ using Eq. (8).
3: Train a new classifier $M_n$ using weighted $\mathcal{W}_S$.
4: Find the least desired model, $M'$, in the current ensemble.
5: Update the ensemble by replacing $M'$ with $M_n$.
6: Return the updated ensemble classifier.
```

If a significant difference, i.e., a drift between the distributions represented by weighted source and target data is detected, the Gaussian kernel model in (1) needs to be updated by re-evaluating $\boldsymbol{\alpha}$. Therefore, if a drift is detected, $\boldsymbol{\alpha}$ is recalculated from $\mathcal{W}_S$ and $\mathcal{W}_T$ using Algorithm 2. Then, importance weight of each instance in $\mathcal{W}_S$ is evaluated following Eq. (8) using the re-evaluated $\boldsymbol{\alpha}$. Next, a new model is trained based on instances from $\mathcal{W}_S$ along with importance weights. Finally, the ensemble classifier $\mathcal{M}$ is updated using the newly trained model along with re-initializing $\mathcal{W}_S$, and $\mathcal{W}_T$. The maximum number of models $\mathcal{M}$ can contain is L. If $\mathcal{M}$ contains less than L models currently, the new model is simply added to $\mathcal{M}$. Otherwise, the least desired model in the ensemble is replaced by the new model.

As instances in $\mathcal{T}$ are unlabeled, it is not practical to find the least desired model by calculating accuracy. Rather, we calculate the confidence of a classifier on each instance in $\mathcal{W}_T$, and replace the model having the least average confidence. We use SVM as the base model in our experiments. A method to produce probabilistic output from an SVM model has been proposed in [17]. We use the probability associated with each predicted class as its confidence in classification. Confidence for most classifiers can be calculated from classification metadata. For examples, the confidence of Bayesian classifier and clustering based classifiers can be estimated using associated probabilities and techniques proposed in [8] respectively.

4 THEORETICAL ANALYSIS

In this section, first we analyze the convergence rate of density ratio generated by the Density Ratio Estimation (DRM) module. Then, we derive the time and space complexity of FUSION.

4.1 Convergence Rate

In order to get the convergence rate, we first prove that the error function $\tilde{E}_i(\hat{\beta})$ is a *strictly convex function*, and gradient of $\tilde{E}_i(\hat{\beta})$ is *Lipschitz* continuous and bounded. Next, we find the convergence rate of *UpdateAlpha* (Algorithm 3). Finally, we determine the convergence rate of the *Density Ratio Estimation*(DRM) module.

LEMMA 4.1. *$\tilde{E}_i(\hat{\beta})$ is a strictly convex function, i.e., $\tilde{E}_i(t\hat{\beta}' + (1-t)\hat{\beta}) < t\tilde{E}_i(\hat{\beta}') + (1-t)\tilde{E}_i(\hat{\beta})$.*

PROOF. From the definition, $\tilde{E}_i(\hat{\beta}) = -\log \hat{\beta}(\mathcal{W}_T^{(i)}) + \frac{\lambda}{2}\left\|\hat{\beta}\right\|_{\mathcal{H}}^2$.

Since *logarithm* is a concave function, we have-

$$\log(t\hat{\beta}' + (1-t)\hat{\beta}) > t\log\hat{\beta}' + (1-t)\log\hat{\beta} \tag{9}$$

$$\frac{\lambda}{2}\|t\hat{\beta}' + (1-t)\hat{\beta}\|^2 < \frac{\lambda t}{2}\|\hat{\beta}'\|^2 + \frac{\lambda(1-t)}{2}\|\hat{\beta}\|^2 \tag{10}$$

Then from Eq. (9) and Eq. (10), we get

$$\tilde{E}_i(t\hat{\beta}' + (1-t)\hat{\beta}) < t\tilde{E}_i(\hat{\beta}') + (1-t)\tilde{E}_i(\hat{\beta}) \tag{11}$$

Here, we assume that $\hat{\beta}'$ is not equal to $\hat{\beta}$. □

Lemma 4.2. *Gradient of* $\tilde{E}_i(\hat{\beta})$ *is* Lipschitz *continuous and bounded, i.e.,* $\left\|\nabla\tilde{E}_i(\hat{\beta}') - \nabla\tilde{E}_i(\hat{\beta})\right\| \le L\left\|\hat{\beta}' - \hat{\beta}\right\|$, *where* $L > 0$.

Proof. The gradient of $\tilde{E}_i$, $\nabla\tilde{E}_i(\hat{\beta}) = -\frac{K_\tau(\cdot, \mathcal{W}_T^{(i)})}{\hat{\beta}} + \lambda\hat{\beta}$

Therefore-

$$\left\|\nabla\tilde{E}_i(\hat{\beta}') - \nabla\tilde{E}_i(\hat{\beta})\right\| = \left\|-\frac{K_\tau(\cdot, \mathcal{W}_T^{(i)})}{\hat{\beta}'} + \lambda\hat{\beta}' + \frac{K_\tau(\cdot, \mathcal{W}_T^{(i)})}{\hat{\beta}} - \lambda\hat{\beta}\right\|$$
$$\le |\lambda|\left\|\hat{\beta}' - \hat{\beta}\right\| + \left|\frac{K_\tau(\cdot, \mathcal{W}_T^{(i)})}{\hat{\beta}\hat{\beta}'}\right|\left\|\hat{\beta}' - \hat{\beta}\right\|$$
$$\le L\left\|\hat{\beta}' - \hat{\beta}\right\|$$

Here, $|\lambda| + \left|\frac{K_\tau(\cdot, \mathcal{W}_T^{(i)})}{\hat{\beta}\hat{\beta}'}\right| \le L$, assuming that $\hat{\beta}, \hat{\beta}' \ne 0$. □

Theorem 4.3. *Assume there are positive numbers M, D, such that* $\left\|\tilde{E}_i(\hat{\beta}')\right\| \le M$ *and* $\left\|\hat{\beta}' - \hat{\beta}\right\|^2 \le D$. *Then, for step size* $\eta = \frac{1}{\gamma N}$, $\mathbb{E}[\tilde{E}_i(\hat{\beta}') - \tilde{E}_i(\hat{\beta})] \le \frac{LQ}{2N}$, *where* $Q = max\{\frac{\eta^2 M^2}{2\eta c - 1}, \left\|\hat{\beta}' - \hat{\beta}\right\|^2\}$.

Proof. From Lemma 4.2, for any $\mathcal{W}_T^{(i)}$, we know that $\left\|\nabla\tilde{E}_i(\hat{\beta}') - \nabla\tilde{E}_i(\hat{\beta})\right\| \le L\left\|\hat{\beta}' - \hat{\beta}\right\|$

Therefore, we have-

$$\tilde{E}_i(\hat{\beta}') \le \tilde{E}_i(\hat{\beta}) + \frac{1}{2}L\left\|\hat{\beta}' - \hat{\beta}\right\| \tag{12}$$

$$\mathbb{E}[\tilde{E}_i(\hat{\beta}') - \tilde{E}_i(\hat{\beta})] \le \frac{1}{2}L * \mathbb{E}\left[\left\|\hat{\beta}' - \hat{\beta}\right\|^2\right] \tag{13}$$

Then from Eq. (12) and Eq. (13), we can get

$$\mathbb{E}[\tilde{E}_i(\hat{\beta}') - \tilde{E}_i(\hat{\beta})] \le \frac{LQ}{2N} \tag{14}$$

□

Therefore, the convergence rate for *UpdateAlpha* (Algorithm 3) is $O\left(\frac{1}{N}\right)$, where N is the sample size.

The convergence rate of *LearnAlpha* (Algorithm 2) is $O\left(n^{-\frac{1}{2+\gamma}}\right)$ for arbitrary small $\gamma > 0$, where n is the number of instances [19]. Assuming that the parameters of the DRM module are estimated initially by *LearnAlpha* algorithm using N_1 number of instances, and thereafter updated online by *UpdateAlpha* algorithm using N_2 instances ($N_2 >> N_1$), the convergence rate of DRM is $O\left(\frac{1+N_1^{\frac{1+\gamma}{2+\gamma}}}{N_1+N_2}\right)$.

4.2 Time and Space Complexity

FUSION has four modules, i.e., *Density Ratio Estimation (DRM)*, *Drift Detection (DDM)*, *Classification*, and *Update*. DRM has two operations, one is to learn $\boldsymbol{\alpha}$ (Algorithm 2), and the other one is to update $\boldsymbol{\alpha}$ online (Algorithm 3). Time complexity to learn $\boldsymbol{\alpha}$ is $O(N_m^2)$, where N_m is the size of the sliding windows. Time complexity to update $\boldsymbol{\alpha}$ is $O(N_m)$. As DRM learns $\boldsymbol{\alpha}$ only once at the beginning, and updates it onward, the amortized time complexity of DRM is less than $O(N_m^2)$. Time complexity of DDM is $O(N_m)$. Time complexity of classification and update depends on the learning algorithm used as the base model. Therefore, FUSION has total time complexity of $O(N_m^2) + f(N_m)$, where $f(N_m)$ is the time complexity for training a new model. However, amortized time complexity of FUSION is much less as $\boldsymbol{\alpha}$ is learned from data occasionally only if a data drift is detected at Line 15, or initially at Line 2 of Algorithm 1.

Space complexity of DRM is $O(N_m^2)$, which dominates space complexities of other modules. Moreover, most learning algorithms have space complexity less than that. Therefore, overall space complexity of FUSION is $O(N_m^2)$. Both time and space complexity of FUSION are functions of N_m. In real world applications, N_m can be tuned to execute FUSION within available resource.

5 EVALUATION

In this section, we describe the experiment setup, and evaluate the proposed approach using synthetic and benchmark real-world data sets. We compare performance of the proposed approach with a number of baseline methods.

Table 2: Characteristics of data sets

Dataset	# features	# classes	# instances
ForestCover	54	7	150,000
KDD	42	23	200,000
PAMAP	53	19	150,000
Electricity	8	2	45,311
SynRBF@002-1	50	5	100,000
SynRBF@002-2	70	7	100,000
SynRBF@003	70	7	100,000

5.1 Data sets

Table 2 lists the data sets used in the experiments. The first four data sets are from real-world, all of them are publicly available. The *ForestCover* data set is obtained from the UCI repository as explained in [15]. It contains geospatial descriptions of different types of forests. The labeling task is to find the actual forest cover type for a given observation from US Forest Service (USFS) Region-2 Resource Information System (RIS) data. The *KDD* [14] data set contains TCP connection records extracted from LAN network traffic over a period of two weeks. Each record refers either to a normal connection or an attack. In *Physical Activity Monitoring (PAMAP)* [18] data set, nine individuals were equipped with sensors that gathered a total of 53 streaming features whilst they performed activities. Nineteen total activities were identified as class labels - including one category for miscellaneous or transient activities. The last real-world data set used in this paper is Electricity [16], which contains data collected from the Australian New South Wales

Table 3: Comparison of performance

Data Set	FUSION		MSC		AHT		SVM	
	Accuracy	Time (Seconds)	Accuracy	Time (Seconds)	Accuracy	Time (Seconds)	Accuracy	Time (Seconds)
ForestCover	**85.10**	469.89	84.4	270.57	61.52	0.05	69.29	8.78
KDD	**97.30**	417.85	96.80	451.54	97.2	0.05	96.29	10.0
PAMAP	**99.80**	471.99	97.40	564.56	94.95	0.08	88.04	7.54
Electricity	**76.50**	238.33	74.60	601.08	75.02	0.02	73.37	0.09
SynRBF@002-1	**98.10**	415.22	93.60	533.33	85.58	0.07	86.29	8.51
SynRBF@002-2	**96.20**	561.86	69.80	232.34	83	0.13	44.13	7.72
SynRBF@003	**93.10**	591.18	58.30	194.49	80.11	0.12	41.28	8.75

Electricity Market. In this market, the price is affected by demand and supply. The class label identifies the change of the price relative to a moving average of the last 24 hours.

SynRBF@X are synthetic data sets generated using *RandomRBFGeneratorDrift* of MOA [2] framework, where X is the Speed of change of centroids in the model. We generate two such data sets using $X = \{0.002, 0.003\}$ to evaluate the approaches on concept drifts having various intensities and frequencies. We generate two versions of *SynRBF@002*, using a different number of cluster centroids and classes. We normalize all the data sets used, and reshuffle the instances from different classes randomly to remove novel classes from them.

We generate a biased source stream from each data set mentioned above using a method similar to previous studies [3, 9] as follows. First, we detect concept drifts in the data set by employing a Naïve Bayes classifier to predict class labels, and monitoring its performance using *ADWIN*, similar to [1]. A minibatch is constructed from data instances between the points at which *ADWIN* detects a significant change in the performance, i.e., a concept drift. Following [9], we first compute the sample mean $\bar{\boldsymbol{x}}$ of a minibatch. Next, we divide the minibatch to form the source and target minibatches. Each instance $\boldsymbol{x}$ is selected to be included in the biased source minibatch according to the probability $P(\xi = 1|\boldsymbol{x}) = \exp\left(-\frac{\|\boldsymbol{x}-\bar{\boldsymbol{x}}\|^2}{2\sigma^2}\right)$, where σ is the standard deviation of $\|\boldsymbol{x} - \bar{\boldsymbol{x}}\|$, for all $\boldsymbol{x}$ in the minibatch. Finally, we select n% of the instances in the minibatch to be included in the source minibatch, and the rest of the instances are included in the target minibatch. The source and target minibatches are concatenated together to form the source and the target stream respectively. We vary n in our experiments to introduce different level of sampling bias between the source and target streams.

5.2 Baseline Methods

The first baseline method we use in this paper is *Multistream Classifier (MSC)* [3], which is the only available method in the literature for multistream classification. MSC uses *Support Vector Machine (SVM)* as the base classifier. To implement the base classifier, we use weighted LibSVM library [4] with RBF kernel. Moreover in MSC, Kernel Mean Matching (KMM) [9] has been used for data shift adaptation. We evaluate the quadratic program in KMM using the CVXOPT python library [5]. To select the parameters of KMM, we use $B_{kmm} = 1000$, $\epsilon_{kmm} = \frac{\sqrt{N_S}-1}{\sqrt{N_S}}$, and γ_{kmm} as the median of pairwise distances in the training set, as suggested in [3].

Although MSC is the only available method for multistream classification, there are a number of methods available for traditional data stream classification. We use SVM and Adaptive Hoeffding Tree (AHT) [2] on a single stream formed by combining the source and the target stream to examine if these approaches really suffer in presence of covariate shift and asynchronous drift in streaming data.

5.3 Setup

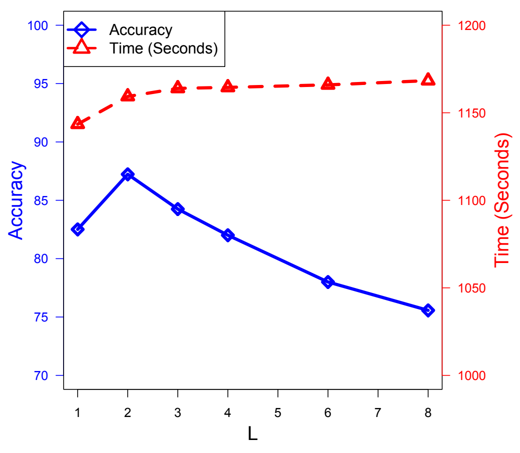
We implemented the proposed approach FUSION and one of the baseline methods MSC using *Python* version 2.7.6. To implement SVM and AHT, we used Weka [6] and MOA [2] respectively. All the methods have been evaluated using a *Linux* machine with *2.40 GHz* core and *16 GB* of main memory. For a fair comparison, we have used SVM with the RBF kernel as the base classifier in the proposed approach (FUSION). As mentioned in Section 3.1, the kernel width (σ) for the Gaussian kernel model in FUSION is selected by likelihood cross-validation. In the experiments, we have used $N_m = 500$, and $L = 2$, and $\tau = 0.0001$. Moreover, we used regularization parameter $\lambda = 0.01$ and learning rate $\eta = 1$ following [11].

5.4 Classification Performance

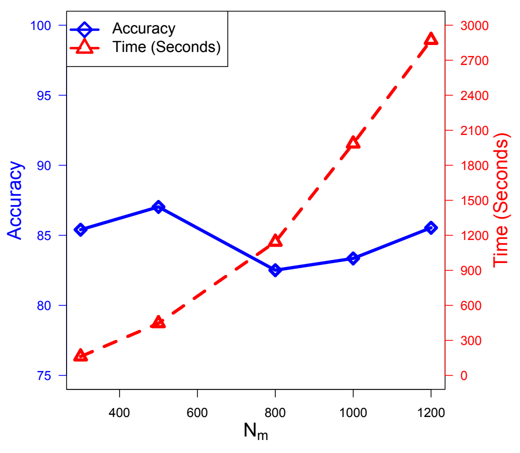
The first set of experiments are designed for comparing classification accuracy and execution time of the approaches considered in this paper on all the data sets mentioned in Table 2.

5.4.1 *Classification Accuracy.*

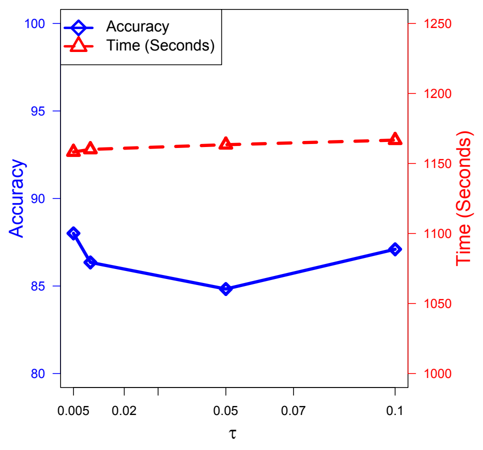
Classification accuracy on different data sets have been shown in Table 3. The proposed approach (FUSION) clearly outperforms all the other baseline approaches considered in this paper. As stated before, we apply SVM and AHT on the combined stream for examining if simply combining the source and the target streams is useful. For a fair comparison, we consider that true labels of only the source stream instances are available. Since both SVM and AHT are fully supervised models, we update the model on labeled source stream data instances once a concept drift is detected. We use ADWIN [1] for detecting concept drifts. Performance of these baseline methods are evaluated on unlabeled target stream data. We observe that SVM and AHT perform poorly compared to the proposed approach on both real-world and synthetic data sets. It indicates that SVM and AHT suffer on the combined stream due to not handling data shift and asynchronous data drifts as stated in Section 2.2.2. The proposed approach also outperforms MSC by a big margin especially on synthetic data sets,



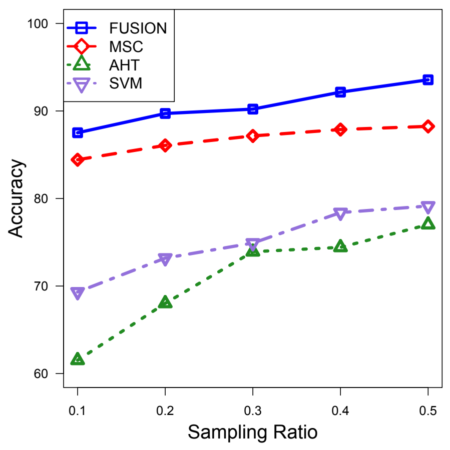
(a) Sensitivity to the ensemble size



(b) Sensitivity to the size of the sliding window



(c) Sensitivity to the drift detection parameter



(d) Sensitivity to Sampling Bias

Figure 3: Parameter sensitivity of FUSION on ForestCover data set

where we introduce frequent concept drifts. This indicates that FUSION adapts to data shift and data drifts more efficiently than MSC.

5.4.2 Execution Time. As discussed in Section 4.2, amortized time complexity of FUSION is less than $O(N_m^2) + f(N_m)$, where N_m is the size of the sliding windows, and $f(N_m)$ is the time complexity for learning a new model. On the contrary, time complexity of MSC is $O(N_m^3)$, where N_m is the maximum size of the sliding windows. Therefore, worst case time complexity of FUSION is better than MSC. Table 3 shows average time to process 1000 instances (in seconds) by the approaches on different data sets. We observe that FUSION achieves competitive execution time compared to MSC if not better. Online density ratio estimation, and inherent drift detection contribute to the improved performance of FUSION in terms of execution time. The other two baselines SVM and AHT shows better execution time understandably as they do not counter for data shift and asynchronous data drift adaptation.

5.5 Parameter Sensitivity

The next set of experiments are designed to examine parameter sensitivity of FUSION. In these experiments, we have used $N_m = 800$, $L = 1$, and $\tau = 0.0001$ as the default setting if not mentioned otherwise.

5.5.1 Ensemble Size. First, we vary the ensemble size (L), and observe how it affects FUSION on the ForestCover data set from Figure 3a. We observe that initially with increasing ensemble size, accuracy also increases. However, further increasing the ensemble size decreases the accuracy slightly, possibly due to a correlation among individual model errors [20]. The execution time of FUSION increases slightly with increasing ensemble size. As mentioned before, unlike MSC, FUSION uses an ensemble classifier containing only models trained for predicting target stream data. As a consequence, ensemble management is much lightweight in FUSION compared to MSC. Therefore, changing ensemble size does not affect the execution time of FUSION significantly.

5.5.2 Maximum Window Size. Next, we examine affect of window size on FUSON using the ForestCover data set in Figure 3b. We observe that the accuracy remains similar with little fluctuations as the size of the sliding window (N_m) increases. However, the execution time increases with the size of the sliding window. The time complexity of FUSION is quadratic with respect to N_m as analyzed in Section 4.2, which is reflected in the experiment result. In real-world applications, N_m can be tuned to execute FUSION within the resource limit.

5.5.3 Drift Detection Parameter. Figure 3c shows affect of the drift detection parameter (τ) on FUSION. We mentioned in Section 3.3 that the false alarm rate of the drift detection algorithm is bounded by τ. We observe from the figure that as τ increases, the number of false alarms produced by the drift detection increases, and the classifier is updated using wrong data instances. Therefore, the accuracy decreases and the execution time increases with increasing τ as expected.

5.5.4 Sampling Bias. Finally, we observe performance of FUSION with different sampling bias introduced in ForestCover data set from Figure 3d. As discussed in Section 5.1, we vary sampling bias in the data set by varying sampling ratio between the source and the target stream. As an example, sampling ratio 0.1 means that we sample only 10% data to be labeled, i.e., included in the source stream data. The rest 90% data are considered unlabeled and included in the target stream data. Therefore, increasing sampling ratio results into decreasing sampling bias and vice versa. We observe that all the methods considered in this paper have better accuracy as the sampling ratio increases. However, the proposed approach FUSION exhibits the best performance. We also observe that performance of AHT and SVM improve rapidly with increasing sampling ratio, as the penalty for not handling sampling bias reduces.

To summarize, the experiments indicate that FUSION is not much sensitive to its parameters. However, it seems that choosing good values for L and N_m is vital for getting better performance. These parameters can be set by doing cross-validation on initial warm-up period data. Furthermore, as the time and space complexity of FUSION depends on N_m, it can be tuned for executing FUSION within the resource limit.

6 CONCLUSION

We have proposed a framework called FUSION in this paper for efficient multistream classification, where unlabeled test data from a target stream needs to be classified using labeled training data from a source stream. The main challenges of multistream classification are data shift, and asynchronous concept drifts between source and target stream data. To address these challenges, FUSION uses an ensemble classifier, where each model is trained using weighted instances from the source stream. The weights are estimated using a Gaussian kernel model by estimating density ratios. The same model is also used for addressing asynchronous concept drifts. Experiment results show the effectiveness of the proposed approach.

ACKNOWLEDGMENTS

This material is based upon work supported by NSF award number 1737978 and DMS-1322353, AFOSR award number FA9550-14-1-0173, and IBM faculty award (Research).

REFERENCES

[1] Albert Bifet. 2009. Adaptive Learning and Mining for Data Streams and Frequent Patterns. *SIGKDD Explor. Newsl.* 11, 1 (Nov. 2009), 55–56.
[2] Albert Bifet, Geoff Holmes, Richard Kirkby, and Bernhard Pfahringer. 2010. MOA: Massive Online Analysis. *J. Mach. Learn. Res.* 11 (Aug. 2010), 1601–1604.
[3] Swarup Chandra, Ahsanul Haque, Latifur Khan, and Charu Aggarwal. 2016. An Adaptive Framework for Multistream Classification. In *Proceedings of the 25th ACM International on Conference on Information and Knowledge Management (CIKM '16)*. ACM, New York, NY, USA, 1181–1190.
[4] Chih-Chung Chang and Chih-Jen Lin. 2011. LIBSVM: a library for support vector machines. *ACM Transactions on Intelligent Systems and Technology (TIST)* 2, 3 (2011), 27.
[5] Joachin Dahl and Lieven Vandenberghe. 2006. Cvxopt: A python package for convex optimization. In *Proc. eur. conf. op. res.*
[6] Mark Hall, Eibe Frank, Geoffrey Holmes, Bernhard Pfahringer, Peter Reutemann, and Ian H. Witten. 2009. The WEKA Data Mining Software: An Update. *SIGKDD Explor. Newsl.* 11, 1 (Nov. 2009), 10–18. https://doi.org/10.1145/1656274.1656278
[7] Ahsanul Haque, Latifur Khan, and Michael Baron. 2016. SAND: Semi-Supervised Adaptive Novel Class Detection and Classification over Data Stream. In *Thirteenth AAAI Conference on Artificial Intelligence*. 1652–1658.
[8] A. Haque, L. Khan, M. Baron, B. Thuraisingham, and C. Aggarwal. 2016. Efficient handling of concept drift and concept evolution over Stream Data. In *2016 IEEE 32nd International Conference on Data Engineering (ICDE)*. 481–492.
[9] Jiayuan Huang, Arthur Gretton, Karsten M Borgwardt, Bernhard Schölkopf, and Alex J Smola. 2006. Correcting sample selection bias by unlabeled data. In *Advances in neural information processing systems*. 601–608.
[10] Takafumi Kanamori, Shohei Hido, and Masashi Sugiyama. 2009. A least-squares approach to direct importance estimation. *The Journal of Machine Learning Research* 10 (2009), 1391–1445.
[11] Yoshinobu Kawahara and Masashi Sugiyama. 2012. Sequential Change-point Detection Based on Direct Density-ratio Estimation. *Stat. Anal. Data Min.* 5, 2 (April 2012), 114–127.
[12] J. Kivinen, A. J. Smola, and R. C. Williamson. 2004. Online Learning with Kernels. *IEEE Transactions on Signal Processing* 52 (August 2004), 2165–2176.
[13] Efthymios Kouloumpis, Theresa Wilson, and Johanna D Moore. 2011. Twitter sentiment analysis: The good the bad and the omg! *Icwsm* 11 (2011), 538–541.
[14] M. Lichman. 2013. UCI Machine Learning Repository. (2013). http://archive.ics.uci.edu/ml
[15] Mohammad M. Masud, Jing Gao, Latifur Khan, Jiawei Han, and Bhavani M. Thuraisingham. 2011. Classification and Novel Class Detection in Concept-Drifting Data Streams under Time Constraints. *IEEE Trans. Knowl. Data Eng.* 23, 6 (2011), 859–874.
[16] MOA. 2015. MOA Massive Online Analysis-Real Time Analytics for Data Streams repository Data Sets. http://moa.cms.waikato.ac.nz/datasets/. (2015).
[17] John C. Platt. 1999. Probabilistic Outputs for Support Vector Machines and Comparisons to Regularized Likelihood Methods. In *Advances in Large Margin Classifiers*. MIT Press, 61–74.
[18] Attila Reiss and Didier Stricker. 2012. Introducing a New Benchmarked Dataset for Activity Monitoring.. In *ISWC*. IEEE, 108–109.
[19] Masashi Sugiyama, Taiji Suzuki, Shinichi Nakajima, Hisashi Kashima, Paul von Bünau, and Motoaki Kawanabe. 2008. Direct importance estimation for covariate shift adaptation. *Annals of the Institute of Statistical Mathematics* 60, 4 (2008), 699–746.
[20] K. Tumer and J. Ghosh. 1996. Error correlation and error reduction in ensemble classifiers. *Connection Science* 8, 3-4 (1996), 385–403.
[21] Bianca Zadrozny Zadrozny. 2004. Learning and Evaluating Classifiers under Sample Selection Bias. In *International Conference on Machine Learning (ICML)*. 903–910.

View publication stats