SCALABLE AND SECURE LEARNING WITH LIMITED SUPERVISION OVER DATA STREAMS

by

Swarup Rama Chandra

APPROVED BY SUPERVISORY COMMITTEE:

Dr. Latifur Khan, Chair

Dr. Zhiqiang Lin, Co-Chair

Dr. Kevin W. Hamlen

Dr. Murat Kantarcioglu

Dr. Bhavani Thuraisingham

Copyright © 2018

Swarup Rama Chandra

All rights reserved

SCALABLE AND SECURE LEARNING WITH LIMITED SUPERVISION OVER DATA STREAMS

by

SWARUP RAMA CHANDRA, BE, MS

DISSERTATION
Presented to the Faculty of
The University of Texas at Dallas
in Partial Fulfillment
of the Requirements
for the Degree of

DOCTOR OF PHILOSOPHY IN
COMPUTER SCIENCE

THE UNIVERSITY OF TEXAS AT DALLAS
August 2018

ACKNOWLEDGMENTS

I am grateful to a number of people in my personal and professional life who have directly or indirectly contributed to this dissertation. First and foremost, I am indebted to my advisor, Latifur Khan, for his valuable guidance and continuous support throughout the PhD program. His insights and dedication as a mentor helped me nurture and grow the spirit of academic research that I will carry forward to the future. I deeply appreciate his constant efforts to provide a conducive environment for collaborative research among fellow labmates, from which I have benefited immensely.

Next, I would like to thank my co-advisor, Zhiqiang Lin, without whom this dissertation would not be complete. His passion and dedication towards research inspired me to work hard. This resulted in me gaining insights into a broad range of research areas. His constant mentoring has truly helped me grow as a researcher, and I am indebted to him.

I would like to thank my dissertation committee, Dr. Kevin Hamlen, Dr. Murat Kantarcioglu, and Dr. Bhavani Thuraisingham, for their time, encouragement and invaluable advice on my dissertation. I am also immensely thankful to my collaborators, who were also my fellow labmates, for the numerous friendly, inspiring, and intellectually satisfying discussions during my PhD years. Particularly, I thank Ahsanul for his help, without whom many of my research endeavors would have failed. He has been a great friend and an excellent collaborator. I would like to thank Khaled, Vishal, Ahmad, and Solaimani (Rakib) for their enduring friendship and support throughout. I would also like to thank Gbadebo (Debo), Sayeed (Rafee), Gao (Alan), Wang, Tao, Yifan, Bo, and Shihabul for their encouragement and interest to continue the research direction of this dissertation.

I am thankful to my parents for their unwavering support and encouragement, without which this dissertation would not be possible. I am indebted to my wife, Gayathri, for her unconditional love and support. I cannot imagine completing this PhD journey without her.

I am also thankful to my extended family for their support, particularly my grandparents for being a source of inspiration and wisdom.

Finally, I thank the National Science Foundation (NSF), Air Force Office of Scientific Research (AFOSR), National Security Agency (NSA), IBM Research, and Hewlett Packard Enterprise (HPE) for providing financial assistance to conduct my research through various grants during the course of my PhD program.

May 2018

SCALABLE AND SECURE LEARNING WITH LIMITED SUPERVISION OVER DATA STREAMS

Swarup Rama Chandra, PhD
The University of Texas at Dallas, 2018

Supervising Professors: Dr. Latifur Khan, Chair
Dr. Zhiqiang Lin, Co-Chair

Applications that employ machine learning over a stream of data provide the knowledge necessary for its users to make informed decisions at the right time. With the advantages of cloud computing infrastructure, these applications can potentially reach a myriad of users. However, challenges arising from evolving statistical properties of data occurring continuously over time, and concerns about data security has severely limited its adoption in the real world. This dissertation contributes new results to address critical challenges in both these complementary research areas, particularly when deployed over a third-party resource.

The first part of this dissertation introduces a novel framework for data classification over a non-stationary data stream where the goal is to learn from limited labeled data over time. Here, a scenario in which multiple data generating processes, that continuously generate data, is considered, with a constraint of labeled data being generated only by a small set of processes whose data distribution is biased compared to the population. The effect of learning with such sampling bias in a concept-drifting data stream is explored. Changes in data distribution over time with biased labeled data degrades classifier performance. By representing instances along the stream as two independent streams, one containing labeled instances (called the source stream) and the other containing unlabeled instances (called the target stream), methodologies which uniquely combine transfer learning mechanisms with drift detection are presented.

While the above framework may adapt existing batch-wise bias correction techniques, these are computationally expensive and are not scalable over a data stream. The next part of this dissertation explores sampling and ensemble techniques to address this challenge. The theoretical and empirical results show large improvements in computational time while maintaining similar performance compared to the baseline methods.

The final part of this dissertation considers security concerns when deploying applications that use machine learning systems on an untrusted third-party resource. Here, the focus is on protecting the learning system against insider threats. A strong adversary can evade security and privacy of an application aiming to protect its code and data. Using the recent commercially available off-the-shelf (COTS) hardware-based cryptographic platform, called Intel SGX, a black-box system can be achieved to protect against such direct attacks. Unfortunately, side-channels from the platform that leak information during computation exists. A novel defense strategy that leverages the trade-off between computational efficiency and privacy to address this challenge is presented, with results demonstrating a large gain in computational time compared to other competing strategies.

TABLE OF CONTENTS

ACKNOWLEDGMENTS . . . iv

ABSTRACT . . . vi

LIST OF FIGURES . . . xi

LIST OF TABLES . . . xiii

CHAPTER 1 INTRODUCTION . . . 1

- 1.1 Data Stream Classification . . . 2
- 1.2 Secure Machine Learning . . . 4
- 1.3 Contributions . . . 5

CHAPTER 2 BACKGROUND . . . 9

- 2.1 Stream Classification . . . 9
- 2.2 Datashift Adaptation . . . 11
- 2.3 Security for Machine Learning . . . 13

CHAPTER 3 MULTISTREAM CLASSIFICATION . . . 16

- 3.1 Preliminaries . . . 16
  - 3.1.1 Notations . . . 16
  - 3.1.2 Problem Statement . . . 16
  - 3.1.3 Challenges . . . 18
- 3.2 Multistream Classification . . . 19
  - 3.2.1 Initialization and Classifier Training . . . 22
  - 3.2.2 Class Prediction . . . 23
  - 3.2.3 Drift Detection . . . 23
  - 3.2.4 Drift Adaptation . . . 27
  - 3.2.5 Ensemble Update . . . 28
- 3.3 Analysis . . . 29
  - 3.3.1 Complexity Analysis . . . 29
  - 3.3.2 Base Classifier Properties . . . 29
- 3.4 Empirical Evaluation . . . 31
  - 3.4.1 Datasets . . . 32

3.4.2 Experiments 33
3.4.3 Results 35
CHAPTER 4 SCALABLE BIAS CORRECTION OVER DATA STREAMS 38
4.1 Kernel Mean Matching 39
4.1.1 Notations 39
4.1.2 Covariate Shift 39
4.1.3 VFKMM: Sampling-based KMM 41
4.1.4 Extension of VFKMM 45
4.1.5 Analysis of VFKMM 47
4.1.6 Empirical Evaluation of VFKMM 51
4.2 Kullback-Leibler Importance Estimation Procedure 61
4.2.1 FUSION 61
4.2.2 Challenges 64
4.2.3 EnsFusion: Ensemble Multistream Classification 64
4.2.4 Analysis of EnsFusion 70
4.2.5 Empirical Evaluation of EnsFusion 74
CHAPTER 5 DEFENDING AGAINST ADVERSARIAL THREATS ON LEARNING 82
5.1 Securing Data Analytics 82
5.1.1 Threat Model 83
5.1.2 Overview 85
5.1.3 Primitives 86
5.1.4 Learning Algorithms 88
5.2 Implementation 92
5.3 Evaluation 93
5.3.1 Quantification of Privacy Guarantee 93
5.3.2 Datasets 95
5.3.3 Results and Discussion 96
5.3.4 Security Evaluation 98
5.4 Related Works 100

CHAPTER 6 CONCLUSION 102
    6.1 Summary 102
    6.2 Future Research Direction 103
        6.2.1 Classification with Limited Supervision 103
        6.2.2 Adversarial attacks on data classification 104
        6.2.3 Applications 105
REFERENCES 107
BIOGRAPHICAL SKETCH 115

CURRICULUM VITAE

LIST OF FIGURES

3.1 An example illustrating asynchronous data drifts on the source and target streams. 18

3.2 Multistream Classification Overview 20

3.3 Illustration of Drift Adaptation cases. 26

3.4 Average accuracy for Multistream Classification : ···●··· *sKMM*; -+- *mKMM-5k*; —●— *MSC*; —★— *MSC2*; —◆— *srcMSC*; -●- *trgMSC*. 36

4.1 Illustrating the *VFKMM* process. The figure shows s samples generated from the training data. KMM is applied over each training sample with the complete test dataset. 42

4.2 Illustrating the *EVFKMM* process. The figure shows s samples generated from the training data, and k partitions of the test data. KMM is applied over each training sample with each test data partition. 45

4.3 Logarithm of NMSE with different k ($\propto \frac{1}{m}$) on —▲— VFKMM and —◆— EVFKMM, compared to —●— CENKMM, —■— ENSKMM and -●- ENSTRKMM. 55

4.4 Logarithm of NMSE with different s on —▲— VFKMM and —◆— EVFKMM, compared to —●— CENKMM. 56

4.5 Average execution time (in logarithm scale) with different k on —▲— VFKMM and —◆— EVFKMM, compared to —●— CENKMM, —■— ENSKMM and -●- ENSTRKMM. 57

4.6 Logarithm of NMSE with different n_{tr} on —▲— VFKMM and —◆— EVFKMM, compared to —●— CENKMM, —■— ENSKMM and -●- ENSTRKMM. 58

4.7 Average execution time (in logarithm scale) with different n_{tr} on —▲— VFKMM and —◆— EVFKMM, compared to —●— CENKMM, —■— ENSKMM and -●- ENSTRKMM. 59

4.8 Overview of EnsFusion 65

4.9 Updating of ensemble ($\mathcal{E}$), with local and global drift detection, upon arrival of a new target instance at time $t + 1$. 67

4.10 Accuracy of Multistream Classification, comparing EnsFusion-One (🟥), EnsFusion-Max (🟦), and Fusion (🟧). 74

4.11 Execution time overhead of Multistream Classification, comparing EnsFusion-One (🟥) and EnsFusion-Max (🟦). 75

4.12 Cumulative number of drifts detected along T for real-world data sets by competing methods on multistream classification: —+— *Fusion*; —●— *EnsFusion-One*; —×— *EnsFusion-Max*. 76

4.13 Cumulative number of drifts detected along T for synthetic data sets by competing methods on multistream classification: —+— *Fusion*; —●— *EnsFusion-One*; —×— *EnsFusion-Max*. 77

4.14 Parameter sensitivity of EnsFusion-Max on ForestCover data set as an example. (—×— Accuracy; - ● - Time Overhead). 78

5.1 Overview of Data Analytics on SGX using randomization. 85

5.2 Illustration of data-oblivious comparison. 88

5.3 Creating an obfuscated decision tree. Shaded nodes are formed using dummy data while others are formed using user-given data. Labels (denoted by $\{1, 2, 3\}$) of the original tree's leaf node is replicated in its descendant leaf nodes of the obfuscated tree. 89

5.4 Measuring privacy guarantee of SGX defense mechanisms. 94

5.5 Time overhead with increasing L (in proportion of n) on decision tree classifier with Rand, on - ● - D, —■— F and - ▸ - S datasets. 99

5.6 (a) Shows similarity scores between access traces across different sets of instances when evaluated on the same classifier. Here, comparison between different defenses are shown, i.e., ▯▯ Rand, ▯▯ Obliv, and no defense (▯▯). (b) Shows similarity between traces of user-given and dummy data instances within a set of instances evaluated on ▯▯ Rand. 99

LIST OF TABLES

3.1 List of symbols for Multistream Classification 17

3.2 Datasets for Multistream Classification 32

3.3 Baseline methods and variants for Multistream Classification 34

3.4 Parameter sensitivity of MSC by measuring average accuracy. 37

4.1 List of additional symbols for VFKMM. 39

4.2 List of datasets for VFKMM. 51

4.3 List of methods used for evaluation. SR denotes sampling with replacement, and SWR denotes sampling without replacement. The proposed approaches are indicated by *, while others are baseline methods. 52

4.4 List of symbols for EnsFusion 64

4.5 Dataset description for EnsFusion. 74

5.1 List of symbols for studying RAND defense. 84

5.2 List of public and confidential parameters. Here, `Tree` indicates model structure, and P indicates probability function. 84

5.3 Dataset statistics and empirical time overhead with $L = n$. 96

CHAPTER 1

INTRODUCTION

The recent proliferation of Internet technology in everyday 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 successively used in applications, such as automatic categorization, personalized services, human activity recognition, and recommendation systems, which are beneficial to its users. Such applications typically employ data analytics to provide services for its users to make informed decisions. For example, a marketing team of a company can use tweet sentiment classifiers to analyze the market response of its products, or an image classifier can be used to improve recommendation or search results in an image search engine. In both these examples, the data occurs as a stream, i.e., tweets and images are continuously produced by a large number of users over time. This need for large-scale data management and analytics has resulted in numerous tool available for performing analytics. Companies such as Amazon and Google offer computing environment for its users to deploy their applications or utilize available machine learning tools.

Unfortunately, there exist fundamental challenges in adoption of machine learning models used to perform analytics over a stream of data. First, for supervised learning models — a focus in this dissertation — sufficient labeled data is typically necessary for learning and adaptation along the stream so that it can generalize well over future unobserved data instances. However, labeled data instances are scarce or partially available. This indicates that the prediction model should maximize performance by leveraging available labeled data during learning. Second, an intrinsic property of a data stream is the data arrival rate. When performing prediction on data instances produced at high frequency, the model is expected to consume those instances at the same rate, i.e., avoid prediction delays, while maintaining high performance. Moreover, changes in data distribution of incoming instances over time entail an adaptive model. The primary challenge in this scenario is to employ a model that consumes data without delay, quickly adapts to changes, and maintains good prediction performance in time. Third, data and code security are of major concern when using data containing sensitive or secret information. Analytics over an untrusted third-party resource can be vulnerable to adversarial attacks, particularly when the adversary controls code execution. Therefore, mechanisms to protect code and data are needed in such an environment.

In this dissertation, we address the above challenges while performing class label prediction on data instances occurring continuously from a set of independent non-stationary concept-drifting processes. The first part of the dissertation focuses on addressing the challenge of learning from limited labeled data along the stream. Here, a scenario is considered in which labeled data occurs only from a small set of processes whose data distribution is biased compared to the population. We discuss methodologies to address data stream classification challenges within this setting. The second part of the dissertation emphasizes designing scalable solutions to address the challenge of prediction delays due to computational bottlenecks in the above setting. Finally, the third part of the dissertation discusses novel strategies to address security concerns when performing analytics over an untrusted third-party resource.

1.1 Data Stream Classification

In general, 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 data distribution over time. This is known as *data drift* (Ditzler et al., 2015). When a classifier is trained in a supervised manner for predicting class labels of instances in a data stream, its performance typically degrades when a change in data distribution occurs. A plethora of studies (Bifet and Gavalda, 2007; Gama et al., 2004; Haque et al., 2016) in the past decade have proposed techniques to address these challenges. Essentially, data instances occurring within a small time period is assumed to have a stationary distribution. A finite set of sequential data instances occurring within this time period, called a minibatch, is used to train a traditional supervised classifier which is used to predict labels of future instances in the stream. To overcome concept drift, a reactive mechanism is employed where the classifier is re-trained or updated when a change in data distribution within a finite-size first-in-first-out sliding window is detected. The change detection mechanism uses a feedback measure from the classifier on sequential label prediction such as classification error (Chandra et al., 2014) or confidence (Haque et al., 2016). In either case, when a change is detected, a new training data is generated by obtaining the true labels of instances from an oracle. The major challenge is the existence of such an oracle or the difficulty of manually labeling data instances on the stream. Recent studies have mitigated this challenge by requiring fewer instances to be labeled along the stream. However, this requires careful selection of data instances to be labeled. In cases where the selection process is biased towards certain types of data instances or labels of certain instances are not easily available, the resulting labeled dataset may have a biased data distribution compared to the current population. In such cases, the traditional supervised classifier may not perform efficiently in predicting the labels of future instances along the stream, particularly when the model is simple or under-specified.

For example, consider an image classifier used to label a stream of photos. Such a stream can be created by users uploading images to a photo sharing service such as Instagram, or from satellites which continuously takes images of outer space. In these cases, obtaining the true label of each image is challenging due to the sheer amount of images that are produced over time. The performance of the classifier may degrade when the data distribution changes, i.e., characteristics of images such as lighting effects, quality, depth of objects, and the shape of objects from users and satellites alike may change over time. In such cases, a new training dataset is needed to update or re-train the classifier. Manually labeling sufficient amount of images along the stream frequently is expensive and tedious. Instead, it is easier to request the label of these images from a small set of users who upload them or estimate image labels from a small set of satellites whose direction and parameters are well-known. However, this labeling scheme may not be a good representation of the population. The chosen set of users may only upload images of a certain characteristic, or camera directions of inter-galaxy satellites may not be fully known. In such cases, the classifier updated from newly created training sets along the stream may have large prediction errors since its assumption of equivalence in training and test data distribution is violated (Bishop, 2006).

Furthermore, it is desirable for data stream analytics to be computationally efficient to consume incoming data instances at the rate of arrival. In the above example, if images arrive at a faster rate than the average classifier computational time (including training and inference time) per image, the analytical solution would not be practical for such applications.

1.2 Secure Machine Learning

Apart from the machine learning challenges described above, the question of computational integrity and data privacy is most relevant in security-sensitive applications where data stream analytics are employed. Particularly, when computation involving data with sensitive information is outsourced to an untrusted third-party resource, data privacy and security is a matter of grave concern to the data-owner. For example, medical practitioners may use a third-party image classification service to detect malicious cancer in MRI scans of their patients. Clearly, this data contains sensitive information that needs to be protected from external adversaries. At present, such services are implemented by building trust with the providers through various compliance measures. However, adversaries may still do harm by violating such regulatory compliance.

An adversary in this environment may be a curious man-in-the-middle or the third-party resource owner who controls the operating system. Typically, data is protected from such external adversaries using cryptographically secure encryption schemes. However, computation performed over unencrypted data have minimal execution overhead, which is ideal for applications over data streams. Techniques that directly perform computation on encrypted data, such as fully-homomorphic encryption schemes (Gentry et al., 2009), are shown to be inefficient for many practical purposes (Liu et al., 2015), and therefore is not suitable for analytics over data streams.

1.3 Contributions

In this dissertation, we introduce a new problem setting to perform data stream classification under the constraint of biased labels. Here, we represent the data stream containing two types of instances over the same domain to address the above learning challenges. A stream of labeled data instances is generated by a small set of non-stationary processes from a particular domain. This is referred to as the *Source Stream*. Meanwhile, another stream of data instances is assumed to be generated by an independent set of non-stationary processes, referred to as the *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., a labeled source stream and an unlabeled target stream, we call it as *Multistream Classification*. Here, the source stream can be viewed as the stream of data with biased data distribution compared to the unlabeled instances in the target stream, at a given time.

In the above examples, labeled images generated from a small set of under-represented users from the source stream, while images from the rest (of the population) form the target stream. Naively, one can combine the two data streams to form a single stream over which existing data stream classification methods can be utilized. However, unavailability of target data instance labels and occurrence of asynchronous concept drifts (i.e., drifts may occur at different times in the independent source and target stream) may attenuate the distinguishing statistical characteristics of data from different classes and the classifier may not be able to maximize the use of labeled data.

The classification problem in the multistream setting partly resonates with that of domain adaptation (Ben-David et al., 2010) and transfer learning (Pan and Yang, 2010) when considering a set of observations in each stream within a particular time window. Numerous applications involving such problems include text classification (Dai et al., 2007), sensor-based location estimation (Pan et al., 2008) and collaborative filtering (Li et al., 2009). Under different types of assumptions on the training and test data, these solutions transform an existing classifier to leverage the available biased label data for predicting labels of unlabeled data instances. Clearly, such applications can be easily augmented to data streams, i.e., in the multistream setting, with concept drift detection. However, these techniques assume a stationary environment with known dataset size, disregarding the computational time involved. Ideally, data stream analytics require fast computation time due to the high velocity and volume of data generated over time. Typically, the computational time limits the speed at which data can be processed. Therefore, there is a need to develop scalable and efficient techniques for multistream analytics.

The main contributions of this dissertation are to first introduce the multistream setting (Chandra et al., 2016a) and address its challenges by appealing to an instance weighting scheme useful for classification over a stream of data having a limited amount of labeled instances. In the process, we develop two scalable bias correction mechanisms for training a supervised classifier using labeled data from the source stream to predict labels of instances over the target stream (Chandra et al., 2016b, 2018), whilst addressing concept drift over time. We evaluate the proposed approaches to study its theoretical properties and empirical performance on real-world and synthetic datasets. Next, we also address the data security and privacy concerns when employing classification in security-sensitive applications over a third-party untrusted resource (Chandra et al., 2017). Here, we appeal to a hardware-based cryptographically secure mechanism and develop data privacy techniques to resist attacks from a strong adversary. We evaluate the execution time performance that is affected due to the security measure employed. Overall, the results in this dissertation indicate that our proposed approach provides a scalable and secure data stream analytics system that can operate under constrained environment. This provides a black-box view of the learning system to an adversary. Yet, recent studies have show that an adversary can attack a machine learning system even when under this black-box setting. In the future, we want to evaluate the multistream setting for its vulnerability to a black-box adversarial attack.

These contributions are summarized as follows.

1. We introduce a new data stream classification setting, called Multistream classification, where data instances are represented by two independent non-stationary streams. Class labels are predicted over one stream (target) whose data instances are unlabeled, using the biased labeled data instances from another stream (source), in an adaptive framework with ensemble classifiers. We empirically evaluate this framework over numerous datasets and compare the results with competing methods.

2. We present a scalable bias correction approach that uses a technique called Kernel Mean Matching, which minimizes the Euclidean distance between the source and target data distribution to compute source instance weights. Particularly, we demonstrate a sampling-based technique over source stream minibatch to form multiple models that can be computed in parallel. We evaluate the scalability of this approach theoretically and empirically to showcase its properties and show its parallel execution on Apache Spark.

3. Similarly, we present another scalable bias correction approach using a technique called Kullback-Liebler Importance Estimation Procedure, which minimizes KL distance instead of Euclidean distance between the two distributions. Here, we utilize an ensemble approach over the target stream to construct smaller models, thereby providing scalability and parallelism. We evaluate the approach to showcase its theoretical properties and empirical performance.

4. Finally, we present a defense strategy for data security and privacy when performing data analytics on an untrusted third-party host. Particularly, we demonstrate a data poisoning strategy to defend against side-channel adversarial attacks on trusted execution environments within Intel SGX. We perform an empirical evaluation to measure execution overhead and security guarantees provided by the solution.

This dissertation is organized as follows. Chapter 2 discusses the relevant background and related works on data stream mining, domain adaptation, and security concerns in machine learning applications. Here, notations that will be followed throughout the dissertation are formally introduced as well. Next, a framework to address challenges of multistream classification is introduced in Chapter 3, including fundamentals of the problem and an empirical evaluation. Chapter 4 extends the multistream classification framework by discussing two important techniques that make the framework scalable on fast data streams. In Chapter 5, the strategy to build a secure black-box application employing machine learning algorithm is presented, with the empirical evaluation that compares with current state-of-the-art defense techniques against strong adversarial attacks. Finally, the dissertation concludes with remarks on future work in Chapter 6.

CHAPTER 2

BACKGROUND

In this chapter, we will discuss relevant background and related works on data stream classification, as well as associated security concerns when used by an application deployed on an untrusted third-party resource.

2.1 Stream Classification

Class label prediction in machine learning has been well studied in the past few decades (Widmer and Kubat, 1996; Domingos and Hulten, 2000; Datar et al., 2002; Bifet and Gavalda, 2007; Gama et al., 2014) under various constraints on the learning process. These include batch and online learning, availability of labels (supervised, semi-supervised or unsupervised), a difference between training and test domains (transfer learning or domain adaptation), etc. The primary question that these studies address is on *how to* predict labels of unknown instances. Particularly in supervised learning, the question of *how to* learn discriminatory patterns from the training set that can be generalized towards the test set is important. Unlike these approaches, studies on data stream classification aim to address an additional question, i.e., *when to* learn newer discriminatory pattern from data.

A *stream of data* can be viewed as a sequence of data instances that continuously occur from a set of data generating processes in an independent and identically distributed (i.i.d) fashion. Unlike time-series data, instances in a data stream are typically assumed to be exchangeable within a short period of time. The task of classification is to predict labels of instances along the stream. More importantly, the data generating processes are assumed to be non-stationary and may cause changes in data distribution over time. These changes may directly affect the labeling or discriminatory function of a classifier learned previous to the change, and consequently affect their classification performance in the future. Furthermore, the classifier training is also affected by the limitation of the observed data. Since a data stream is assumed to be theoretically infinite, it is impractical to save every data instance in memory for training. In applications where data instances occur rapidly, the classifier operation should not ideally cause computational bottlenecks that affect data consumption. Therefore, scalability challenges in learning and inference need to be addressed.

The infinite length problem of a data stream is typically addressed by dividing the whole stream into fixed-size minibatches, e.g., (Masud et al., 2008) or using a gradual forgetting mechanism, e.g., (Klinkenberg, 2004). Recent approaches (Bifet and Gavalda, 2007; Gama et al., 2004; Haque et al., 2016) 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 change in class boundaries, is detected. Then, the classifier is updated while minibatch instances representing the old concept are removed. This aids in using an updated classifier that generalizes well on a data distribution currently represented by instances along the stream.

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 (Harel et al., 2014). 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 over time, approaches proposed in (Bifet and Gavalda, 2007; Gama et al., 2004) adopt the principle by Vapnik et al. (Vapnik, 1998) to detect this change indirectly by tracking drift in the error rate of the 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., (Masud et al., 2008) have focused on partly addressing this issue by assuming delayed labeling or active learning settings, yet requiring some test data label. Approach proposed in (Haque et al., 2016) addresses this issue by estimating classifier confidence in classification, and tracking any significant change in confidence scores.

2.2 Datashift Adaptation

A fundamental assumption in machine learning is that both the training and test data represent the same data distribution (Bishop, 2006). This is known as the stationary distribution assumption. 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 such a scenario. In the literature, this problem is studies under the name of domain adaptation or transfer learning (Zhang et al., 2013). Here, the goal is to learn a transfer function between the training and test datasets such that it aids in learning an improved predictive function over the test dataset. In general, the training and test data may not occur from the same domain. However, recent studies show that a relation between the two domains is necessary to achieve a substantial improvement in learning a discriminatory function for the test dataset (Ben-David et al., 2010). In this dissertation, we will focus on data streams formed by a set of data generating processes that belong to the same domain. We leave the exploration of problems related to different domains as future work.

In the Bayesian perspective, a change in data distribution between training and test datasets is viewed as a difference in their joint probability distribution. In other words, if (x, y) denote a data instance with covariates x and corresponding label y, and the joint probability distribution of the training and test is denoted by $P_{tr}(x, y)$ and $P_{te}(x, y)$ respectively, then $P_{tr}(x, y) \neq P_{te}(x, y)$. There exist various settings depending on the availability of labels in the training and test domains. Here, we focus on one such setting where labels of training data instances are fully available while no labels are available on the test dataset.

In this setting of domain adaptation, a type of difference in distribution can be viewed as $P_{tr}(x) \neq P_{te}(x)$ with $P_{tr}(y|x) = P_{te}(y|x)$. This is popularly known as *covariate shift* (Huang et al., 2006), which typically occur due to the use of under-specified model that underfit the data. Nevertheless, such simple models (e.g. SVM) are typically considered ideal for performing analytics over a data stream due to its low training and inference complexity compared to a more complex model such as neural networks. Recent studies have focused on developing correction mechanisms for covariate shift by estimating probability density ratio given by $\beta(x) = \frac{P_{te}(x)}{P_{tr}(x)}$. Here, the aim is to directly complete $\beta(x)$ rather than estimating the density ratio by computing $P_{te}(x)$ and $P_{tr}(x)$ separately. These batch-algorithms include Kernel Mean Matching (Huang et al., 2006) (KMM) and Kullback Leibler Importance Estimation Procedure (Sugiyama et al., 2008) (KLIEP), and unconstrained Least Square Importance Fitting (Kanamori et al., 2009) (uLSIF).

Concretely, 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

$$\left\| E_{x\sim P_{tr}(x)}[\beta(x)\phi(x)] - E_{x\sim P_{te}(x)}[\phi(x)] \right\| \tag{2.1}$$

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 2.1 converges to $P_{te}(x) = \beta(x)P_{tr}(x)$ (Yu and Szepesvári, 2012).

Instead of minimizing the Euclidean distance using MMD, Sugiyama et al. (Sugiyama et al., 2008) minimize the Kullback-Leibler distance between the training and test data distribution in KLIEP. Particularly, they argue that KMM suffers from a model selection problem where the chosen model parameters can be biased under ordinary cross-validation. Therefore, they model the density ratio using a linear kernel. This is given as follows.

$$\hat{\beta}(x) = \sum_{j=1}^{N} \alpha_j K_\sigma(x, x_{te}^{(j)}) \tag{2.2}$$

where $\boldsymbol{\alpha} = \{\alpha_j\}_{j=1}^{N}$ are the set of N parameters needed to be learned, $K_\sigma(\cdot,\cdot)$ is the Gaussian kernel, i.e., $K_\sigma(x^{(i)}, x^{(j)}) = \exp\left\{-\frac{\left\|x^{(i)}-x^{(j)}\right\|^2}{2\sigma^2}\right\}$, and σ is the kernel width. Using this model, the empirical minimization of KL distance provides the following optimization function.

$$\begin{aligned}
&\underset{\{\alpha_j\}_{j=1}^{N}}{\text{maximize}} \left[\sum_{i=1}^{N} \log\left(\sum_{j=1}^{N} \alpha_j K_\sigma(x_{te}^{(i)}, x_{te}^{(j)})\right)\right] \\
&\text{subject to } \frac{1}{N}\sum_{i=1}^{N}\sum_{j=1}^{N} \alpha_j K_\sigma(x_{tr}^{(i)}, x_{te}^{(j)}) = 1, \\
&\textit{and } \alpha_1, \alpha_2, ..., \alpha_N \geq 0.
\end{aligned} \tag{2.3}$$

where N is the size of training and test datasets. We adopt this batch-wise density ratio estimation methods over a data stream to address the challenges of multistream classification, i.e., label prediction of streaming data instances under the constraint of limited and biased availability of labeled data.

2.3 Security for Machine Learning

The security concerns when deploying a machine learning model burgeons from the perils of a strong adversary. In the perspective of an attack, the adversary may be interested in the following.

- Extracting sensitive information from a model trained on a secret dataset.
- Obtain classified information about the data used for prediction.
- Disrupt model performance by contaminating data during training or evaluation.
- Generate adversarial examples to fool the model (black-box attack).

To aid in protecting an application using security-sensitive or secret data from strong adversaries who may control computation, hardware-assisted cryptographic platforms have been recently developed. One such platform that is available on commercial devices is Intel's SGX.

Intel Software Guard Extensions (SGX) (Anati et al., 2013) is a set of additional processor instructions to the x86 family, with hardware support to create secure memory regions within existing address space. Such an isolated container is called a *Enclave*, while the rest of the address space is untrusted. Data within these memory regions can only be accessed by code running within the enclave. This access control is enforced by the hardware, using attestation and cryptographically secure keys (Costan and Devadas, 2016) with a trusted processor. The new SGX instructions are used to load and initialize an enclave, as well as enter and exit the protected region. From a developer's perspective, an enclave is entered by calling trusted `ecalls` (enclave calls) from the untrusted application space. The enclave can invoke untrusted code in its host application by calling `ocalls` (outside calls) to exit the enclave. Data from the enclave is always encrypted when it is in memory, but there are cases in which the content should be securely saved outside the enclave. The process of exporting the secrets from an enclave is known as *Sealing*. The encrypted sealed data can only be decrypted by the enclave. Every SGX-enabled processor contains a secret hardware key from which other platform keys are derived. A remote party can verify that a specific enclave is running on SGX hardware by having the enclave perform remote attestation. Therefore, execution within the enclave can protect against most of the attacks. However, their naive execution fails to create a secure black-box.

Attacks

While performing computations within the enclave, an adversary controlling the host OS may infer sensitive and confidential information from side-channels (Rane et al., 2015). Assuming the application executed within an enclave is benign, i.e., it does not actively leak information, the attacker may observe input-dependent patterns in data access and execution timing for inferring sensitive information. This is called as *cache-timing attack* (Götzfried et al., 2017). Since OS is allowed to have full control over the page table of an SGX enclave execution, the attacker controlling the OS may know page access patterns. This eliminates noise in side-channels and is called as *Controlled-channel attack* (Xu et al., 2015).

Defenses

The burden of ensuring efficiency, data privacy and confidentiality lies with the application developer who verifies platform authenticity and performs guarded memory and I/O access. Therefore, studies have proposed various mechanisms including balanced execution (Shinde et al., 2016) and data-oblivious computations (Ohrimenko et al., 2016). In balanced execution, each branch of a conditional statement is forcefully executed by creating dummy operations of data and resource access (Rane et al., 2015). Whereas a data-oblivious solution has the control-flow independent of its input data. In particular to deploying machine learning models in such an adversarial environment, recent studies (Ohrimenko et al., 2016) have shown that efficient ORAM techniques (Stefanov et al., 2013) cannot be employed for data analytics since it does not hide input-dependent access paths, and is not ideal for applications making a large number of memory accesses. However, data-independent access techniques can be used to defend against page-level and cache-level attacks.

CHAPTER 3

MULTISTREAM CLASSIFICATION[1]

In this chapter, we present the problem of multistream classification, discuss its unique challenges, and detail a framework to address them. We also analyze the theoretical properties of the mechanisms used specific to the framework and evaluate it with real-world and synthetic datasets.

3.1 Preliminaries

We now formalize the multistream classification problem and present challenges of performing label prediction over drifting data streams in this context.

3.1.1 Notations

Table 3.1 lists frequently used symbols. A bold symbol or letter is used to denote a set of elements, and a superscript to indicate the index of an element in the set. A subscript is used to indicate the association of an element or a set to a type such as a source or a 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 $\mathcal{S}$.

3.1.2 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 independent covariates, and $y \in \mathbf{Y} = \{1 \ldots K\}$, is its corresponding class label. Here, K is the number of classes known apriori. In this setting,

---

[1] Chandra, Swarup, et al. "An adaptive framework for multistream classification." Proceedings of the 25th ACM International on Conference on Information and Knowledge Management. ©2016 ACM, Inc. http://doi.acm.org/10.1145/2983323.2983842

Table 3.1: List of symbols for Multistream Classification

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
K	Number of class labels
$\mathcal{M}$	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

assume without loss of generality that $\mathbf{P}$ consists of two processes, one called a *source* (denoted as $\mathcal{S}$) and the other called a *target* (denoted by $\mathcal{T}$). Throughout the dissertation, 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.*

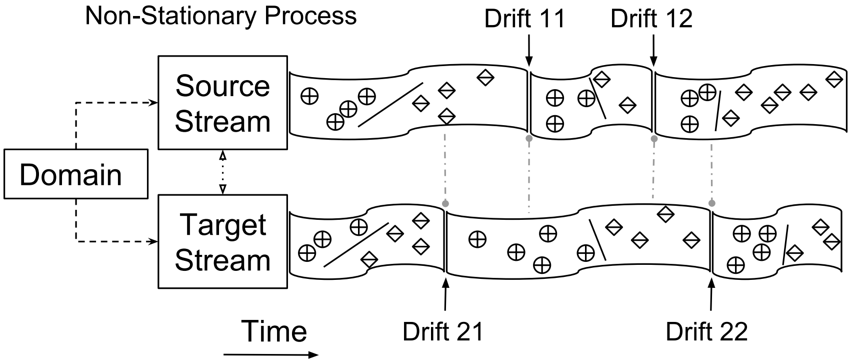


Figure 3.1: An example illustrating asynchronous data drifts on the source and target streams.

3.1.3 Challenges

As mentioned in Chapter 1, we consider a case where data instances from $\mathcal{S}$ have a biased data distribution compared to those from $\mathcal{T}$. This leads to a biased training distribution among observed instances within any given time period. This biased 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.

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 the source and target streams is illustrated in Figure 3.1 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 3.1, 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, the 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.

3.2 Multistream Classification

In this section, we describe our proposed framework for multistream classification, referred to as MSC ($\underline{\text{M}}$ulti$\underline{\text{S}}$tream $\underline{\text{C}}$lassifier). In our multistream setting, 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.

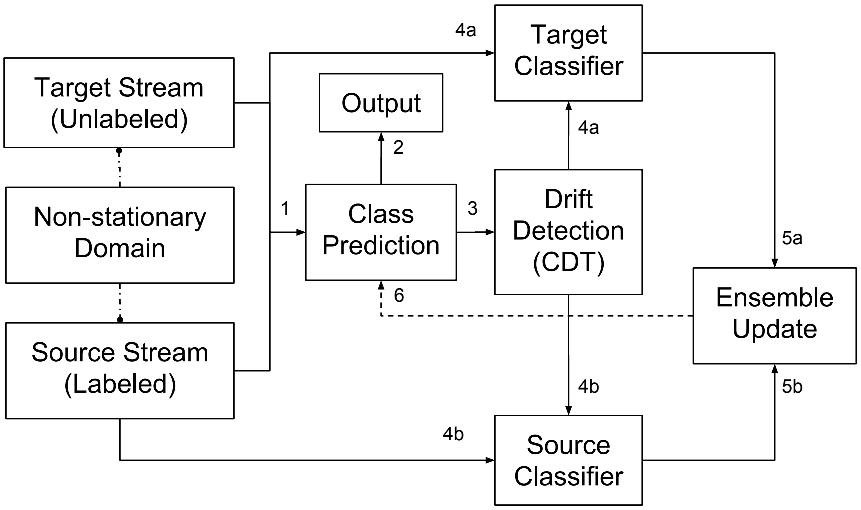


Figure 3.2: Multistream Classification Overview

Figure 3.2 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 of classifiers is initialized with a source-classifier and a target-classifier, each trained on an initial minibatch of data instances from appropriate streams, i.e., source and target stream respectively. Using this initialized 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 classifier-ensemble, illustrated by `Class Prediction` in Figure 3.2 (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 the corresponding class label is observed on $\mathcal{S}$. This recorded classifier feedback is 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 the distribution of recorded classifier error is detected on $\mathcal{S}$. We call the resulting classifier as

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($\hat{Y}_T, B_T$)
```

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 the 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, the 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.

3.2.1 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 $\mathcal{M}_S$ 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 2.1. An optimal β is obtained by minimizing the corresponding empirically equivalent quadratic optimization problem (Huang et al., 2006), given by

$$\begin{aligned} &\boldsymbol{\beta}^{(t)*} \approx \underset{\beta^{(t)}}{\text{minimize}} \, \frac{1}{2}\boldsymbol{\beta}^T \mathbf{K} \boldsymbol{\beta} - \boldsymbol{\kappa}^T \boldsymbol{\beta} \\ &\text{subject to } \beta_i \in [0, B_{kmm}] \;\&\; \left| \frac{1}{n_{tr}} \sum_{i=1}^{n_{tr}} \beta_i - 1 \right| \leq \epsilon_{kmm} \end{aligned} \tag{3.1}$$

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. We use this training procedure throughout the classification process, 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 §3.3. The weighted classifier is then added to E.

3.2.2 Class Prediction

Every classifier in the ensemble predicts a class label for a given data instance. The classifier-ensemble's output is a 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.

3.2.3 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.

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 (Gama et al., 2004). Moreover, we show in §3.3.2 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 §3.2.2. 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 sequential data instances from $\mathcal{T}$, as indicated in §3.2.2. 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.

Change Detection

We invoke a change point detection method to check for significant changes in W_S and W_T over time. Particularly, we propose a CUSUM-type change detection technique (Bifet and Gavaldà, 2009) (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

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 3.2).
        $\omega_n = \max_{\gamma \leq q \leq n-\gamma} s(q,n)$
        if $\omega_n \geq$ Threshold then
            Return $q_m$, where $s(q,n) = \omega_n$.
        else
            Return $-1$.
    else
        Return 0.
```

each split is estimated. At least γ values are needed for a 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. At any point in time, if the 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

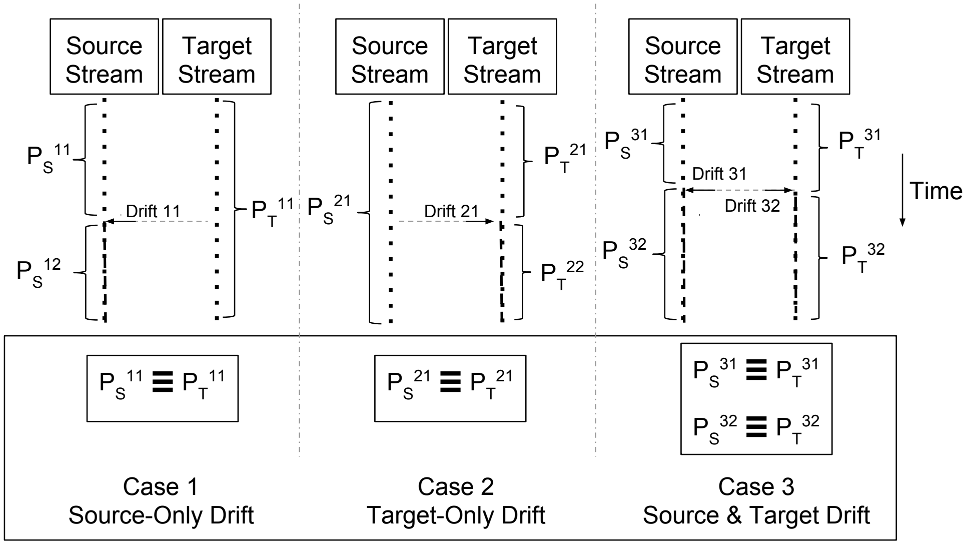


Figure 3.3: Illustration of Drift Adaptation cases.

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 the 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:

$$s(q,n) = \sum_{i=q+1}^{n} log\left(\frac{P\left(W_h[i] \mid \boldsymbol{\theta}_a\right)}{P\left(W_h[i] \mid \boldsymbol{\theta}_b\right)}\right) \tag{3.2}$$

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 (denoted by `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.

3.2.4 Drift Adaptation

Once a drift is detected on a stream, new classifiers are trained appropriately using a mini-batch 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 3.3 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 drift detected along $\mathcal{S}$ and $\mathcal{T}$ can be viewed as an application of these three cases in series with the 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 target-classifier may be required to address the issues discussed above in case 1 and 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 dynamically sized minibatches. This is performed using `updateMinibatch` in Algorithm 1. Note that the data and feedback buffers of a stream are of equal size at all times. Therefore, a change point associated with the feedback buffer has an equivalent index on the corresponding data buffer.

3.2.5 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 $\mathcal{T}$.

Accordingly, 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 indices 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 replaced.

3.3 Analysis

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

3.3.1 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, i.e., instance weights from 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 $\mathcal{O}(n_{tr}^2 + n_{tr} n_{te})$ (Miao et al., 2015). 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 (Haque et al., 2016), 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 $\mathcal{O}(n^2)$.

3.3.2 Base Classifier Properties

The choice of a base classifier for *MSC* should meet the following two criteria.

- It should be able to perform kernel computations in RKHS since instance weights from KMM in target-classifier can only be used in RKHS (Huang et al., 2006).

- 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 the 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 un-thresholded 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 et al. (Platt, 1999), 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 on the positive side of the hyperplane, and that of class b is on the negative side. Therefore, $P(f(x) \mid y = a) = r_a e^{-r_a(1-f)}, f \leq 1$, and $P(f(x)|y = b) = r_b e^{-r_b(f-1)}, f \geq -1$. 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-

$$P(y = a \mid f(x)) = \frac{1}{1 + \exp(Af(x) + B)} \tag{3.3}$$

where $A = -(r_a + r_b)$ and $B = r_a - r_b + \ln \frac{P(y=b)}{P(y=a)}$. Since $r_a, r_b \geq 0$, A ¡ 0. Therefore, monotonicity of (Equation 3.3) in $f(x)$ can be assured.

Lemma 1. *A 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.

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

$$\begin{aligned} f(x_a^{(t)}) = f(x_a^{(t+1)}) + \delta_a^{(t+1)} &\Rightarrow f(x_a^{(t)}) > f(x_a^{(t+1)}) \\ &\Rightarrow 1 + \exp\big(Af(x_a^{(t)}) + B\big) < 1 + \exp\big(Af(x_a^{(t+1)}) + B\big) \\ &\Rightarrow P(y = a|f(x_a^{(t)})) > P(y = a|f(x_a^{(t+1)})) \text{ [using (}\textit{Equation}\text{ 3.3)]} \end{aligned} \tag{3.4}$$

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

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

$$\begin{aligned} f(x_a^{(t)}) > 0 \;\&\; f(x_a^{(t+1)}) < 0 &\Rightarrow f(x_a^{(t)}) > f(x_a^{(t+1)}) \\ &\Rightarrow P(y = a|f(x_a^{(t)})) > P(y = a|f(x_a^{(t+1)})) \text{ [Similar to (}\textit{Equation}\text{ 3.4)]} \end{aligned}$$

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 (Wu et al., 2004)

3.4 Empirical Evaluation

We now describe the experimental setup and present empirical results on various datasets using *MSC*, while comparing its performance to other competing baseline methods.

Table 3.2: Datasets for Multistream Classification

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

3.4.1 Datasets

We use 3 real-world and 3 synthetic datasets for evaluation of the proposed approach. Table 3.2 lists these datasets with corresponding properties. The publicly available real-world datasets are ForestCover [2], Sensor [3] and SEA [4]. Particularly, the ForestCover dataset consists of geospatial records describing forest-cover types (class label). The Sensor dataset contains information regarding temperature, humidity, light, and sensor voltage. Based on the reading, the classification task is to identify the sensorID. Finally, SEA dataset consists of concept-drifting data instances generated from 3 independent attributes, typically used as a benchmark for evaluating data stream algorithms. The other three synthetic data with concept drift were created using the well-known MOA framework (Bifet et al., 2010).

We generate a biased source stream in each dataset, using a method similar to previous studies (Huang et al., 2006), 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 (Bifet and Gavalda, 2007). A minibatch is constructed from data instances between the points at which *ADWIN* detects a significant change in performance. Following (Huang et al., 2006), we first compute the sample mean $\bar{x}$ of a minibatch, and divide it into two equal

---

[2] `https://archive.ics.uci.edu/ml/datasets/Covertype`

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

[4] `http://www.liaad.up.pt/kdus/products/datasets-for-concept-drift`

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 is concatenated to form the source stream, and target minibatches are concatenated to form the target stream.

3.4.2 Experiments

Baseline

Since there exist 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*.

Variants

We now describe 3 variants of the proposed *MSC* method to evaluate our design decisions.

1. Recall that *MSC* originally 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

Table 3.3: Baseline methods and variants for Multistream Classification

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.

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 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.

2. Similarly, we construct *target-MSC* or *trgMSC* by replacing each `buildSourceModel` in Algorithm 1 with `buildTargetModel`. In this case, the ensemble contains only target-classifiers.

3. Finally, 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 independent ensembles and update each of them accordingly, we call this method as *MSC2*.

Setup

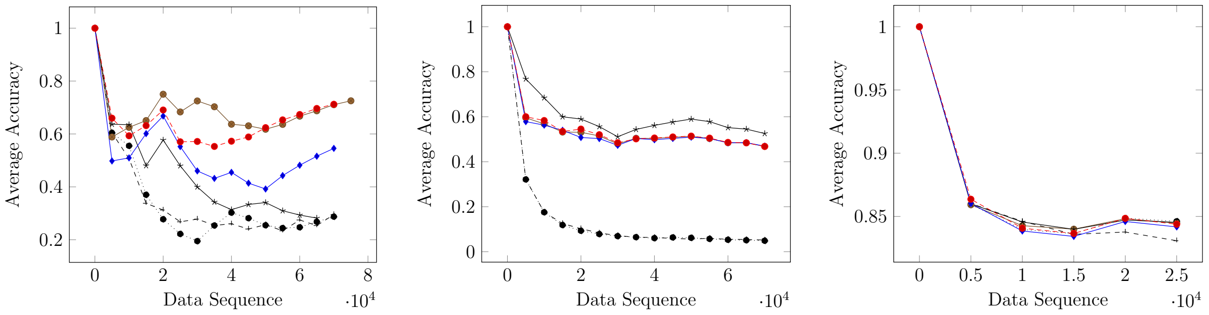
Table 3.3 summarizes each competing methods. We use the weighted LibSVM library (Chang and Lin, 2011) with $C_{svm} = 1.3 \times 10^5$, $\gamma_{svm} = 1 \times 10^{-4}$ and an RBF Kernel as the base classifier. We use KMM for covariate shift correction. The quadratic program of Equation 3.1 is evaluated using the CVXOPT python library (Dahl and Vandenberghe, 2006). 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 (Huang et al., 2006). 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.

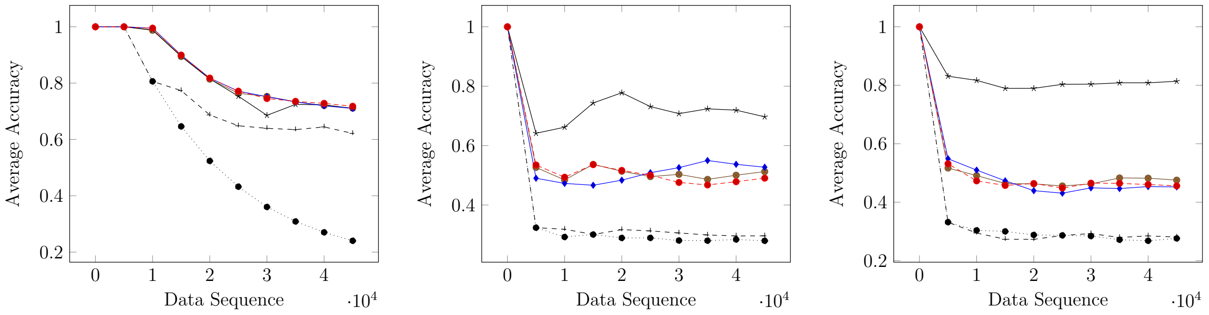
3.4.3 Results

Figure 3.4 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 (Wang et al., 2003) due to reduced variance.

Among variants of the proposed framework, *MSC* and *MSC2* performs better than *srcMSC* and *trgMSC* on most datasets. Particularly, *MSC* performs best in the ForestCover



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



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

Figure 3.4: Average accuracy for Multistream Classification : ···●··· *sKMM*; -+- *mKMM-5k*; —●— *MSC*; —✶— *MSC2*; —◆— *srcMSC*; -●- *trgMSC*.

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 due to a bias 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 since changes due to drifts are only measured based on classifier feedback such as error and confidence value.

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

Table 3.4: Parameter sensitivity of MSC by measuring average accuracy.

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

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 3.4. From the table, it can be observed that *MSC* and *MSC2* are marginally sensitive to different values of α_d (within a given range) 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, and can be chosen by cross-validation. Moreover, greedy replacement of non-performing classifiers in the fixed-size ensemble may also affect overall accuracy, especially in *MSC* where a single type of classifier may become dominant in the ensemble over time. Especially when drifts occur more frequently on one stream ($\mathcal{S}$ or $\mathcal{T}$) more than the other. This behavior is reflected in Figure 3.4 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 (Haque et al., 2016). We leave this for future work as well.

CHAPTER 4

SCALABLE BIAS CORRECTION OVER DATA STREAMS[1][2]

The previous chapter introduced a framework to perform classification with limited labeled data over a data stream where the labeled data distribution is biased compared to the unlabeled data distribution. While the empirical results in §3.4 show that MSC performs best among competing methods, it still suffers from large computational overhead. Particularly, the time complexity of MSC for bias correction and drift detection is $\mathcal{O}(n^3)$, as shown in §3.3. When instances in the data stream occur at a rate greater than it being evaluated by MSC, prediction delays accumulate.

In this chapter, we explore various techniques to address this challenge. Specifically, the first part describes a sampling-based method to address the computational bottleneck in MSC when using the Kernel Mean Matching method. Here, we focus on the batch process of KMM since that can be directly applicable over the multistream setting. We call this VFKMM or Very Fast KMM. The next part introduces a modified framework where the density ratio computation can be adapted for online learning over the multistream setting. We call this FUSION as it leverages KLIEP to perform both bias correction and drift detection. Here, we explore ensemble methods to address FUSION's scalability problem when used over large datasets. For both these parts, we thoroughly analyze the theoretical properties and evaluate it empirically on real-world and synthetic datasets.

---

[1] ©2016 IEEE. Reprinted, with permission, from S. Chandra, A. Haque, L. Khan and C. Aggarwal, "Efficient Sampling-Based Kernel Mean Matching," 2016 IEEE 16th International Conference on Data Mining (ICDM), December 2016

[2] ©2018 IEEE. Reprinted, with permission, from S. Chandra, A. Haque, H. Tao, J. Liu, L. Khan and C. Aggarwal, "Ensemble Direct Density Ratio Estimation for Multistream Classification," 2018 IEEE 34th International Conference on Data Engineering (ICDE)

Table 4.1: List of additional symbols for VFKMM.

Symbols	Description
n_{tr}, n_{te}	Total number of train/test instances
ϕ	RKHS Map
h	RKHS Kernel
ϵ, B, σ	KMM parameters
K, κ	KMM kernel functions
R	Bound of feature space
δ	Confidence interval
η	Sampling error tolerance
m	Sample size
s	Number of samples
k	Number of data components

4.1 Kernel Mean Matching

4.1.1 Notations

We use a similar notation as mentioned in §3.1. In addition, Table 4.1 lists symbols specifically used in this chapter.

4.1.2 Covariate Shift

According to Ben-David et al. (Ben-David et al., 2010), if the two distributions are arbitrarily different, then learning from the datasets representing two dissimilar distributions is not possible with bounded error. However, under a few assumptions, the difference in distribution can be addressed by techniques that transfer knowledge (patterns) from training data to test data using instances or feature representation (Pan and Yang, 2010). In §2.2, we have described the motivation for covariate shift. In general, the relation between $P_{tr}(x)$ and $P_{te}(x)$ is accounted by computing an importance weight $\beta(x) = \frac{P_{te}(x)}{P_{tr}(x)}$ for each training instance x. Recent studies have focused on computing $\beta(x)$ without explicitly estimating $P_{te}(x)$ and $P_{tr}(x)$. Though they provide elegant solutions to estimate importance weights, these methods do not scale well on large datasets.

Specifically, the idea in Kernel Mean Matching is to minimize the mean Euclidean distance between weighted training data distribution $\beta(\mathbf{x})P_{tr}(x)$ and corresponding test data distribution $P_{te}(x)$ in a Reproducing Kernel Hilbert Space (RKHS) $\mathcal{F}$ with feature map $\phi : \mathcal{D} \rightarrow \mathcal{F}$. Mean distance is measured by computing the *Maximum Mean Discrepancy* (MMD), given in Equation 2.1. Here, it is assumed that 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 h is assumed to be universal in $\mathcal{D}$. It has been shown that under these conditions, minimizing MMD converges to $P_{te}(x) = \beta(x)P_{tr}(x)$ (Yu and Szepesvári, 2012).

Centralized KMM (cenKMM)

In particular, minimizing MMD to obtain optimal importance weights is equivalent to minimizing the corresponding quadratic program that approximates the population expectation with an empirical expectation. The empirical approximation of MMD to obtain the desired $\hat{\beta}(\mathbf{x})$ is given by

$$\hat{\boldsymbol{\beta}} \approx \arg\min_{\beta} \left\| \frac{1}{n_{tr}} \sum_{i=1}^{n_{tr}} \beta(x_{tr}^{(i)})\phi(x_{tr}^{(i)}) - \frac{1}{n_{te}} \sum_{j=1}^{n_{te}} \phi(x_{te}^{(j)}) \right\|^2 \tag{4.1}$$

where n_{tr} and n_{te} are sizes of the training and test datasets respectively, and $\hat{\beta}(x) \in \hat{\boldsymbol{\beta}}$. The equivalent quadratic program is as follows.

$$\hat{\boldsymbol{\beta}} \approx \underset{\beta}{\text{minimize}} \frac{1}{2}\boldsymbol{\beta}^T \mathbf{K} \boldsymbol{\beta} - \boldsymbol{\kappa}^T \boldsymbol{\beta} \tag{4.2}$$

$$\text{subject to } \beta(x^{(i)}) \in [0, B], \forall i \in \{1 \dots n_{tr}\}$$

$$\& \left| \sum_{i=1}^{n_{tr}} \beta(x^{(i)}) - n_{tr} \right| \leq n_{tr}\epsilon$$

where $\mathbf{K}$ and $\boldsymbol{\kappa}$ are matrices of a RKHS kernel $h(\cdot)$ with $K^{(ij)} = h(x_{tr}^{(i)}, x_{tr}^{(j)}) \in \mathbf{K}$, and $\kappa^{(i)} = \frac{n_{tr}}{n_{te}} \sum_{j=1}^{n_{te}} h(x_{tr}^{(i)}, x_{te}^{(j)}) \in \boldsymbol{\kappa}$. $B > 0$ is an upper bound on the solution search space, and ϵ is the normalization error. Note that we have used this method in the MSC framework (in Chapter 3).

Ensemble KMM (ensKMM)

The *cenKMM* approach requires the complete set of training and test data to be in memory while solving the quadratic program. When considering a large test dataset, which is a typical sampling bias scenario, this constraint cannot be satisfied if all test data instances do not fit into the memory. Recently, (Miao et al., 2015) proposed a technique called ensemble KMM (denoted by ENSKMM) to address this challenge. Since $\beta(x) \propto P_{te}(x)$, an ensemble of estimators is obtained by dividing only the test data instances into k components. They assume that the training dataset is small, and can completely fit in memory. For each test component $\mathbf{C}$, the weight estimates $\hat{\boldsymbol{\beta}}_c$ is computed by solving Equation 4.2 using the complete set of n_{tr} training instances. These component-wise instance weights are combined to form $\hat{\boldsymbol{\beta}} = \frac{1}{k}\sum_{c=1}^{k}\hat{\boldsymbol{\beta}}_c$. While the study demonstrates improvement in accuracy and execution time, computational efficiently is still largely limited by the size of training dataset used. As mentioned in §3.3, the time complexity of KMM is $\mathcal{O}(n_{tr}^3 + n_{tr}^2 v + n_{tr} n_{te} v)$. Clearly, this method becomes computationally expensive with an increase in n_{tr}, even when it is reasonably smaller than n_{te}.

4.1.3 VFKMM: Sampling-based KMM

The challenge of scalability while estimating density ratio can be addressed naively by splitting the training data into smaller subsets and follow an ensemble approach (Chawla et al., 2004) by applying KMM over each subset independently. We refer to this naive method by Ensemble of TRaining data KMM or ENSTRKMM. Here, sampling over training data is performed without replacement, where each sample represents an independent subset (or partition). KMM can be applied over each sample independently of the other, using the complete test dataset. The union of density ratio estimates from all the samples provides instance weight for each training data instance. However, such a method may not perform well since a small subset of training data instances (chosen uniformly at random) may exhibit

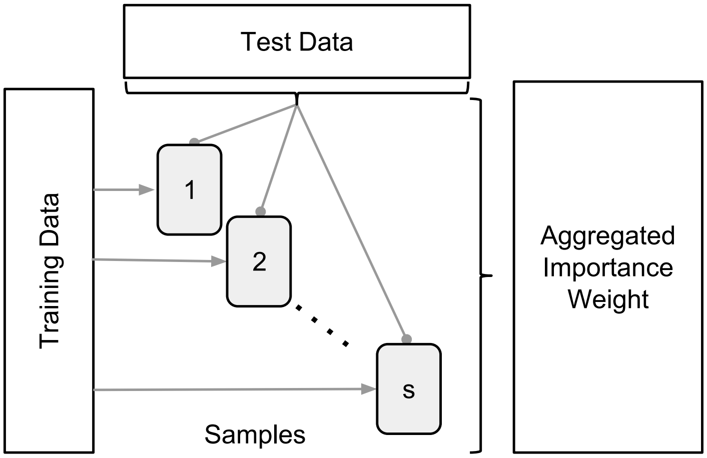


Figure 4.1: Illustrating the *VFKMM* process. The figure shows s samples generated from the training data. KMM is applied over each training sample with the complete test dataset.

glaringly different distribution compared to the original training data distribution. This can adversely affect KMM output (Yu and Szepesvári, 2012).

In this paper, we propose an approach to address the above challenge employing bootstrap training samples whose data instances are chosen with replacement. Using the well-established theoretical support on *m/n bootstrap* methods (Bickel and Sakov, 2008), we show its applicability to KMM. In the following sections, we present and analyze our proposed approach.

Very Fast KMM

Given an i.i.d. set of training covariates $\mathbf{X}_{tr}$ and an i.i.d. set of test covariates $\mathbf{X}_{te}$, such that $\mathbf{X}_{tr}$ is sufficiently large, the problem is to efficiently estimate instance weight $\hat{\beta}(x)$ for each $x \in \mathbf{X}_{tr}$ using the Kernel Mean Matching method. Estimation of $\beta(x) \in \boldsymbol{\beta}$ is sensitive to the training data distribution, i.e., the estimates may vary depending on the size and choice of instances used as training data. Bootstrap methods (Efron, 1992) have been shown to be extremely useful when estimators are unstable. In this scenario, one can employ a bootstrap sampling process by generating samples with replacement from the given training

```
Algorithm 3: VFKMM : KMM by Bootstrap Aggregation
  Data: Covariates $\mathbf{X}_{tr}$ and $\mathbf{X}_{te}$
  Input:  Sample Size: $m$, Tolerance: $\eta$, Parameters: $\theta$
  Result: $\hat{\boldsymbol{\beta}}$
  begin
      $s \leftarrow \frac{\ln \eta}{m \ln\left(1-\frac{1}{n_{tr}}\right)}$ /* Compute # samples */
      $\hat{\boldsymbol{\beta}} \leftarrow zeros$
      for $i \in \{1, s\}$ do
          $\mathbf{X}_{tr}^{*(i)} \leftarrow$ generateSample($\mathbf{X}_{tr}, m$)
          $\hat{\boldsymbol{\beta}}^{*(i)} \leftarrow KMM(\mathbf{X}_{tr}^{*(i)}, \mathbf{X}_{te}, \theta)$ /*Equation 4.2*/
          $\hat{\boldsymbol{\beta}} \leftarrow$ aggregate($\hat{\boldsymbol{\beta}}^{*(i)}$)
      return normalize($\hat{\boldsymbol{\beta}}$)
```

data. However, a naive bootstrap sample consists of n_{tr} instances. This does not aid in improving the computational time efficiency of KMM as desired. Therefore, the *m-out-of-n* bootstrap sampling (or m/n bootstrap) method is more appropriate since $m < n_{tr}$ can be fixed, thereby reducing KMM time complexity to be linear in the size of test dataset. Here, m is the sample size and $n = n_{tr}$. We utilize this notion to achieve scalability during covariate shift correction, which is similar to the generic bag-of-little-bootstrap method (Kleiner et al., 2014).

An overview of our proposed approach to achieve scalability in Kernel Mean Matching is illustrated in Figure 4.1. We refer to it by V̲ery F̲ast K̲M̲M̲ or VFKMM. A number of fixed-size bootstrap samples are generated from the training dataset $\mathbf{X}_{tr}$. Data instances in each sample are chosen with replacement from $\mathbf{X}_{tr}$, where each sample is denoted by $\mathbf{X}_{tr}^{*}$. By considering the complete set of test data instances, $\hat{\beta}^{*}(x)$ for each instance $x \in \mathbf{X}_{tr}^{*}$ is computed using Equation 4.2. Since each instance can be associated with multiple samples, it would have an equivalent number of estimated instance weights. Final weight for each instance is calculated by taking the average of all weights calculated for this instance.

Algorithm 3 details the process of using m/n bootstrap method over training data to compute instance weights using KMM. First, we compute the minimum number of samples (denoted as s) to be generated. Ideally, one would desire each training data instance to be selected at least once overall, i.e., each data instance is associated with at least one sample. Due to randomness in the selection process, this desired property cannot be guaranteed. With a large number of samples, however, one can be highly confident that each data instance belongs to at least one sample. Therefore, we define η as the sampling error tolerance level such that each instance in the training dataset is associated with at least one of the s samples with probability $1 - \eta$. Since m data instances are to be selected per sample, the minimum number of samples needed, given η, is

$$\frac{\ln \eta}{m \ln \left(1 - \frac{1}{n_{tr}}\right)} \tag{4.3}$$

where $\frac{1}{n_{tr}}$ is the probability of selecting a training instance from $\mathbf{X}_{tr}$ uniformly at random with replacement. We present more details of this in §4.1.5.

Next, `generateSample` creates a sample $\mathbf{X}_{tr}^{*(i)}$, where $i \in \{1 \dots s\}$, by uniformly selecting data instances with replacement from $\mathbf{X}_{tr}$. We use this along with the complete test dataset $\mathbf{X}_{te}$ to solve Equation 4.2 in KMM. We repeat this process s times, thereby generating s independent samples. At each iteration, we aggregate $\hat{\beta}^{*(i)}(x)$ into $\hat{\beta}(x)$ for each $x \in \mathbf{X}_{tr}^{*(i)}$. For $x \in \mathbf{X}_{tr}$, the final $\hat{\beta}(x) \in \hat{\boldsymbol{\beta}}$ contains the sum of all estimated value in each iteration. For example, let $n_{tr} = 10$, and $m = 5$, and let the i^{th} sample $\mathbf{X}_{tr}^{*(i)} = (x^{(1)}, x^{(3)}, x^{(4)}, x^{(5)}, x^{(9)})$ be selected at random by `generateSample`. Here, $x^{(j)}$ is the j^{th} instance of $\mathbf{X}_{tr}$. If $\hat{\boldsymbol{\beta}}^{*(i)} = (\hat{\beta}^{*(i)}(x^{(1)}), \hat{\beta}^{*(i)}(x^{(3)}), \hat{\beta}^{*(i)}(x^{(4)}), \hat{\beta}^{*(i)}(x^{(5)}), \hat{\beta}^{*(i)}(x^{(9)}))$ are the resulting estimated instance weights for $x \in \mathbf{X}_{tr}^{*(i)}$, then `aggregate` performs summation of new estimates for each instance in $\mathbf{X}_{tr}^{*(i)}$, i.e., $\hat{\beta}(x^{(1)}) = \hat{\beta}(x^{(1)}) + \hat{\beta}^{*(i)}(x^{(1)})$, $\hat{\beta}(x^{(3)}) = \hat{\beta}(x^{(3)}) + \hat{\beta}^{*(i)}(x^{(3)})$, and so on. Moreover, we initialize $\hat{\beta}(x)$ to 0 for each $x \in \mathbf{X}_{tr}$. We finally output $\hat{\beta}(x) \in \hat{\boldsymbol{\beta}}$, $\forall x \in \mathbf{X}_{tr}$, by dividing it with the number of times x is selected in the sampling process, using `normalize`.

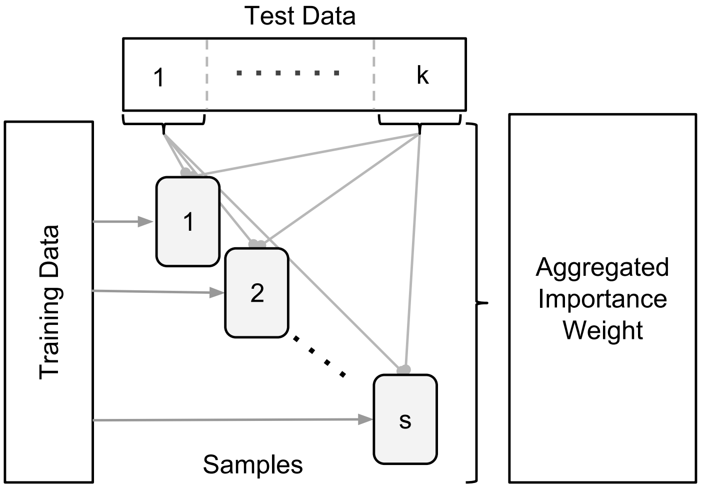


Figure 4.2: Illustrating the *EVFKMM* process. The figure shows s samples generated from the training data, and k partitions of the test data. KMM is applied over each training sample with each test data partition.

Algorithm 4: EVFKMM : Ensemble-based VFKMM

```
Data: Covariates $\mathbf{X}_{tr}$ and $\mathbf{X}_{te}$
Input: # of components: $k$, Tolerance: $\eta$, Parameters: $\theta$
Result: $\hat{\boldsymbol{\beta}}$
begin
    /* Sample without replacement */
    $\tilde{\mathbf{X}}_{te} \leftarrow splitData(\mathbf{X}_{te}, k)$
    $m \leftarrow \frac{n_{tr}}{k}$
    for $\mathbf{C} \in \tilde{\mathbf{X}}_{te}$ do
        $\hat{\boldsymbol{\beta}}_c \leftarrow VFKMM(\mathbf{X}_{tr}, \mathbf{C}, m, \eta, \theta)$
        $\hat{\boldsymbol{\beta}} \leftarrow aggregate(\hat{\boldsymbol{\beta}}_c)$
    return $normalize(\hat{\boldsymbol{\beta}})$
```

4.1.4 Extension of VFKMM

We now propose a variant of VFKMM to address the challenge of scalability when both training and test datasets are large. In particular, we divide the test dataset into multiple non-overlapping partitions, and utilize VFKMM method over each partition. We refer to this as Ensemble-based Very Fast KMM (or EVFKMM).

ENSKMM requires the complete training data to be in memory. Likewise, VFKMM requires the complete test data to be in memory, as described in §4.1.3. When both training and test datasets are large, in-memory sequential computation of KMM becomes a bottleneck in the application employing it. One can address this challenge by applying the sampling process of VFKMM over the test dataset, along with the training dataset. However, sampling over the test dataset only approximates its data distribution. Moreover, a method that partitions the test data and applies KMM on each partition independently, has been demonstrated to achieve better performance (Miao et al., 2015).

Instead of sampling from test dataset, we propose a method that augments VFKMM with ENSKMM. Here, we first split the test data into multiple components, where each component is a subset with a fixed number of data instances, called *test-component*. Next, for each of these, we employ VFKMM where fixed-size samples are obtained from the training data. Finally, we aggregate the density ratio resulting from KMM, similar to VFKMM.

Figure 4.2 illustrates this many-to-many computation scheme between the training samples and test-components, as compared to the many to one computation scheme between training samples and the complete test dataset of VFKMM in Figure 4.1. Algorithm 4 details this method. We split the given test data $\mathbf{X}_{te}$ into k components, to form a set $\tilde{\mathbf{X}}_{te}$. Since VFKMM requires sample size m, we compute it as $\frac{n_{tr}}{k}$. We then run VFKMM using each test-component (denoted as $\mathbf{C}$) and the training data, to obtain aggregated density ratio $\hat{\boldsymbol{\beta}}_c$ for each $\mathbf{C}$. We further aggregate all $\hat{\beta}_c(\mathbf{x}) \in \hat{\boldsymbol{\beta}}_c$ for every $\mathbf{x} \in \mathbf{X}_{tr}$, and normalize all density ratio estimates, similar to Algorithm 3. Since each combination of training and test components can be computed independently, it can be computed in a distributed environment. We implemented this approach over Apache Spark (Haque et al., 2016). However, in this dissertation, we will examine the primary scalability of the approach rather than discussing the distributed implementation.

4.1.5 Analysis of VFKMM

As mentioned earlier in §4.1, importance weights of training data are bounded within $[0, B]$, where $0 < B < \infty$. Furthermore, kernel h is continuous in domain $\mathcal{D}$. According to Gretton et al. (Gretton et al., 2009), the convergence error of $\beta(x) \in \boldsymbol{\beta}$ computed from cenKMM is given by the following lemma.

Lemma 2. *Let $\mathbf{X}_{tr}$ be a set of training data covariates and $\mathbf{X}_{te}$ be a set of test data covariates, with size n_{tr} and n_{te} respectively. Let $\beta(x) < B$ be a fixed function of x where $x \in \mathbf{X}_{tr}$. Assume that instances in $\mathbf{X}_{tr}$ and $\mathbf{X}_{te}$ are drawn i.i.d. from $\mathcal{D}$ using $P_{te}(x) = \beta(x)P_{tr}(x)$, and $\|\phi\| \leq R < \infty$. Then with probability at least $1 - \delta$,*

$$\left\| \frac{1}{n_{tr}} \sum_{i=1}^{n_{tr}} \beta(x_{tr}^{(i)})\phi(x_{tr}^{(i)}) - \frac{1}{n_{te}} \sum_{j=1}^{n_{te}} \phi(x_{te}^{(j)}) \right\| \leq$$

$$\left(1 + \sqrt{2 \ln \frac{2}{\delta}}\right) R \sqrt{\frac{B^2}{n_{tr}} + \frac{1}{n_{te}}}$$

Algorithm 3 indicates that $m < n_{tr}$ training data instances are used to form a sample $\mathbf{X}_{tr}^*$. Here, we ignore the sample index i for brevity. The KMM output $\hat{\boldsymbol{\beta}}^*$ is obtained from $\mathbf{X}_{tr}^*$ and $\mathbf{X}_{te}$. Following Lemma 2, the bound of change in density ratio for each sample is $4R^2\left(\frac{B^2}{m} + \frac{1}{n_{te}}\right)$. The overall density ratio is considered by aggregating $\hat{\boldsymbol{\beta}}^*$ resulting from KMM applied over each sample. Note that not all data instances from $\mathbf{X}_{tr}$ are always selected due to randomness. Therefore, each data instance has a certain probability of being selected at least once depending on the sample size m and number of samples s, which affects the overall bound of change calculation in this situation.

Intuitively, the probability of a training data instance being associated with at least one sample in the overall process increases with increase in number of samples. We denote this probability measure as $(1 - \eta)$, where η is the sampling error tolerance level (probability) that a data instance is not selected in any sample of Algorithm 3. Minimum number of samples s to be generated under this condition is given by the following Lemma.

Lemma 3. *Let s be the number of training samples generated from $\mathbf{X}_{tr}$ in* VFKMM*, where each sample $\mathbf{X}_{tr}^{*(i)}$, $i \in \{1 \dots s\}$ is of size $m < n_{tr}$. Union of all samples is given by $\hat{\mathbf{X}}_{tr} = \bigcup_{i=1}^{s} \mathbf{X}_{tr}^{*(i)}$. The minimum number of samples required to be generated such that an instance $\mathbf{x} \in \mathbf{X}_{tr}$ belongs to the set $\hat{\mathbf{X}}_{tr}$ with probability at least $(1-\eta)$ is given by $\left\lceil \frac{\ln \eta}{m \ln\left(1-\frac{1}{n_{tr}}\right)} \right\rceil$*

Proof. Probability that a data instance $\mathbf{x} \in \mathbf{X}_{tr}$ is not selected in any of s independent samples, each having m independent trials, is $\left(1 - \frac{1}{n_{tr}}\right)^{ms}$. Using the definition, $\eta \leq \left(1 - \frac{1}{n_{tr}}\right)^{ms}$. Therefore, $s \geq \frac{\ln \eta}{m \ln\left(1-\frac{1}{n_{tr}}\right)}$. □

Theorem 1. *Following the assumptions of Lemma 2 and Lemma 3, and considering only the instances that have been associated with at least one of the samples, the error of* VFKMM*, where s independent training samples (each of size m) are generated, is given by*

$$\left\| \frac{1}{n_{tr}} \sum_{i=1}^{n_{tr}} \hat{\beta}(\mathbf{x}_{tr}^{(i)}) \phi(\mathbf{x}_{tr}^{(i)}) - \frac{1}{n_{te}} \sum_{j=1}^{n_{te}} \phi(\mathbf{x}_{te}^{(j)}) \right\| \leq$$

$$R \left(\sqrt{2 \ln\left(\frac{2}{\delta}\right) \left(\frac{(1-\eta)^2 B^2}{n_{tr}} + \frac{1}{n_{te}} \right)} + \sqrt{\frac{B^2}{n_{tr}} + \frac{1}{n_{te}}} \right)$$

Proof. Let $\Gamma(\hat{\mathbf{X}}_{tr}, \mathbf{X}_{te})$ be defined by

$$\Gamma(\hat{\mathbf{X}}_{tr}, \mathbf{X}_{te}) = \left\| \frac{1}{n_{tr}} \sum_{i=1}^{n_{tr}} \hat{\beta}(\mathbf{x}_{tr}^{(i)}) \phi(\mathbf{x}_{tr}^{(i)}) - \frac{1}{n_{te}} \sum_{j=1}^{n_{te}} \phi(\mathbf{x}_{te}^{(j)}) \right\| \tag{4.4}$$

Lemma 3 states that each instance in the training dataset is associated with at least one of the samples, with probability $(1-\eta)$. Let $\mathbf{x}_{tr}^{(i)} \in \mathbf{X}_{tr}$ be an instance selected in the sampling process. If it is replaced by an arbitrary $\mathbf{x} \in \mathcal{D}$, then bound on the change in $\Gamma(\hat{\mathbf{X}}_{tr}, \mathbf{X}_{te})$ is

$$\frac{1}{n_{tr}} \left| (1-\eta) \left(\hat{\beta}(\mathbf{x}_{tr}^{(i)}) \phi(\mathbf{x}_{tr}^{(i)}) - \hat{\beta}(\mathbf{x}) \phi(\mathbf{x}) \right) \right| \leq \frac{2(1-\eta)BR}{n_{tr}}$$

since $\hat{\beta}(\cdot) \leq B$. Similarly, $\frac{1}{n_{te}} |\phi(\mathbf{x}_{te}^{(j)}) - \phi(\mathbf{x})| \leq \frac{2R}{n_{te}}$. Using McDiarmid's tail bound (McDiarmid, 1989) on $\Gamma(\hat{\mathbf{X}}_{tr}, \mathbf{X}_{te})$, for an arbitrary small ϵ, we get

$$p(|\Gamma(\hat{\mathbf{X}}_{tr}, \mathbf{X}_{te}) - E_{\hat{\mathbf{X}}_{tr}, \mathbf{X}_{te}}[\Gamma(\hat{\mathbf{X}}_{tr}, \mathbf{X}_{te})]| > \epsilon) \leq$$

$$2 \exp \left(\frac{-\epsilon^2}{2R^2 \left(\frac{(1-\eta)^2 B^2}{n_{tr}} + \frac{1}{n_{te}} \right)} \right)$$

Therefore, the two-tail bound with probability $1-\delta$ is

$$|\Gamma(\hat{\mathbf{X}}_{tr}, \mathbf{X}_{te}) - E_{\hat{\mathbf{X}}_{tr}, \mathbf{X}_{te}}[\Gamma(\hat{\mathbf{X}}_{tr}, \mathbf{X}_{te})]| \leq R\sqrt{2\ln\left(\frac{2}{\delta}\right)\left(\frac{(1-\eta)^2 B^2}{n_{tr}} + \frac{1}{n_{te}}\right)} \tag{4.5}$$

Finally, the bound on expected value of $\Gamma(\hat{\mathbf{X}}_{tr}, \mathbf{X}_{te})$ is derived using Jensen's inequality as $E_{\hat{\mathbf{X}}_{tr}, \mathbf{X}_{te}}[\Gamma(\hat{\mathbf{X}}_{tr}, \mathbf{X}_{te})] \leq \sqrt{E_{\hat{\mathbf{X}}_{tr}, \mathbf{X}_{te}}[\Gamma(\hat{\mathbf{X}}_{tr}, \mathbf{X}_{te})^2]}$. Similar to (Gretton et al., 2009),

$$E_{\hat{\mathbf{X}}_{tr}, \mathbf{X}_{te}}[\Gamma(\hat{\mathbf{X}}_{tr}, \mathbf{X}_{te})^2] \leq R^2\left[\frac{B^2}{n_{tr}} + \frac{1}{n_{te}}\right] \tag{4.6}$$

Combining tail bound (Equation 4.5) and expectation bound (Equation 4.6) completes the proof. □

Due to randomness during sampling, a few data instances may not be selected in any of the samples, even when considering a sufficiently large number of samples. Therefore, we add a penalty for each data instance $\mathbf{x} \in \mathbf{X}_{tr}$ not present in the union of samples $\hat{\mathbf{X}}_{tr}$, for comparison with cenKMM. This penalty term is defined as follows.

Definition 2. *Let* $\mathbf{X}_0$ *be the set of instances from training dataset* $\mathbf{X}_{tr}$ *that are not associated with any of the bootstrap samples, i.e.,* $\mathbf{X}_0 = \mathbf{X}_{tr} \setminus \hat{\mathbf{X}}_{tr}$. *Then, replacing each data instance* $\mathbf{x} \in \mathbf{X}_0$ *by an arbitrary instance* $\mathbf{x} \in \mathcal{D}$ *changes* $\Gamma(\hat{\mathbf{X}}_{tr}, \mathbf{X}_{te})$ *by at most* τ.

Corollary 1. *Following similar assumptions as Theorem 1, error of utilizing the estimated* $\hat{\boldsymbol{\beta}}$ *from VFKMM over the entire training dataset* $\mathbf{X}_{tr}$ *is given by*

$$\left\|\frac{1}{n_{tr}}\sum_{i=1}^{n_{tr}}\hat{\beta}(\mathbf{x}_{tr}^{(i)})\phi(\mathbf{x}_{tr}^{(i)}) - \frac{1}{n_{te}}\sum_{j=1}^{n_{te}}\phi(\mathbf{x}_{te}^{(j)})\right\| \leq R\left(\sqrt{\frac{1}{2}\ln\left(\frac{2}{\delta}\right)\left(\frac{(2B(1-\eta)+\eta\tau)^2}{n_{tr}} + \frac{4}{n_{te}}\right)} + \sqrt{\frac{B^2}{n_{tr}} + \frac{1}{n_{te}}}\right)$$

Proof. Following Lemma 3 and Definition 2, an instance belongs to $\mathbf{X}_0$ with probability η. Therefore, if any instance $\mathbf{x}_{tr}^{(i)}$ in the training dataset is replaced by an arbitrary $\mathbf{x} \in \mathcal{D}$, the expected bound on change in $\Gamma(\hat{\mathbf{X}}_{tr}, \mathbf{X}_{te})$ is $\frac{(1-\eta)2BR+\eta\tau}{n_{tr}}$. Likewise, replacing $\mathbf{x}_{te}^{(i)}$ by an arbitrary $\mathbf{x} \in \mathcal{D}$ changes $\Gamma(\hat{\mathbf{X}}_{tr}, \mathbf{X}_{te})$ by at most $\frac{2R}{n_{te}}$. Using McDiarmid's tail bound (McDiarmid, 1989), we get the following.

$$p(|\Gamma(\hat{\mathbf{X}}_{tr}, \mathbf{X}_{te}) - E_{\hat{\mathbf{X}}_{tr}, \mathbf{X}_{te}}[\Gamma(\hat{\mathbf{X}}_{tr}, \mathbf{X}_{te})]| > \epsilon) \leq 2\exp\left(\frac{-2\epsilon^2}{R^2\left(\frac{(2B(1-\eta)+\eta\tau)^2}{n_{tr}} + \frac{4}{n_{te}}\right)}\right)$$

Therefore, the two-tail bound with probability $1-\delta$ is

$$|\Gamma(\hat{\mathbf{X}}_{tr}, \mathbf{X}_{te}) - E_{\hat{\mathbf{X}}_{tr}, \mathbf{X}_{te}}[\Gamma(\hat{\mathbf{X}}_{tr}, \mathbf{X}_{te})]| \leq R\sqrt{\frac{1}{2}\ln\left(\frac{2}{\delta}\right)\left(\frac{(2B(1-\eta)+\eta\tau)^2}{n_{tr}} + \frac{4}{n_{te}}\right)} \tag{4.7}$$

Finally, the bound on expected value of $\Gamma(\hat{\mathbf{X}}_{tr}, \mathbf{X}_{te})$ follows Equation 4.6 due to similar conditions. Combining tail bound (Equation 4.7) and expectation bound (Equation 4.6) completes the proof. □

Corollary 2. *If sufficiently large number of samples are taken from the training dataset, error bound of* VFKMM *converges to that of* CENKMM*.*

Proof. From Lemma 3, it follows that η decreases with increasing s, i.e., number of samples. With sufficiently large number of samples, $\lim_{s\to\infty} \eta = 0$. Therefore, the error bound of VFKMM (from Corollary 1) converges to the error bound of CENKMM (shown in Lemma 2). □

Complexity Analysis

We now compute the time and space complexity of VFKMM. As mentioned in (Miao et al., 2015), the original KMM approach (CENKMM) has a time complexity of $\mathcal{O}(n_{tr}^3 + n_{tr}^2 d + n_{tr} n_{te} d)$. However, in the sampling process of VFKMM, each sample contains m training

Table 4.2: List of datasets for VFKMM.

Dataset	**# Features**	**Total Size**
ForestCover	54	50,000
KDD	34	50,000
PAMAP	53	50,000
PowerSupply	2	29,928
SEA	3	50,000
Syn002	70	50,000
Syn003	70	50,000
MNIST	780	50,000

data instances instead of n_{tr}. Hence, the time complexity of estimating $\hat{\boldsymbol{\beta}}^*$ in each sample is $\mathcal{O}(m^3 + m^2 d + mn_{te}d)$. The aggregation requires $\mathcal{O}(m)$. Together, the time complexity per sample is $\mathcal{O}(m^3 + m^2 d + mn_{te}d + m)$. Since computation on samples can be done in parallel, the overall time complexity remains the same. Clearly, scale-up in execution time is achieved when $m \ll n_{tr}$. Similarly, the space complexity of cenKMM is $\mathcal{O}(n_{tr}^2 + n_{tr}n_{te})$, whereas that of VFKMM is $\mathcal{O}(m^2 + mn_{te} + n_{tr})$.

4.1.6 Empirical Evaluation of VFKMM

Dataset

Table 4.2 lists the datasets used in our empirical evaluation. All datasets are publicly available (Repository, 1998; Fan et al., 2008). We use a similar set of datasets as used in the previous chapter. The additional set of datasets are as follows. KDD is an intrusion detection dataset with network features collected from various attacks. The classification task is to identify the type of attack. PAMAP is a human activity monitoring dataset where data from different sensors are collected. The classification task is to identify the activity associated with a set of sensor readings. The PowerSupply is a dataset consisting of power supply records of an electric company where the classification task is to predict whether the future demand for power increases or decreases based on current usage. Finally, the MNIST

Table 4.3: List of methods used for evaluation. SR denotes sampling with replacement, and SWR denotes sampling without replacement. The proposed approaches are indicated by *, while others are baseline methods.

		Training Data		**Test Data**	
		SR	SWR	SR	SWR
Method	cenKMM	✗	✗	✗	✗
	ensKMM	✗	✗	✗	✔
	ensTrKMM	✗	✔	✗	✗
	VFKMM*	✔	✗	✗	✗
	EVFKMM*	✔	✗	✗	✔

dataset is a set of images with handwritten digits. The task is to identify the digit using pixel features. Similar to the previous chapter, Syn002 and Syn003 are synthetically generated using MOA (Bifet et al., 2010).

In order to simulate sampling bias between the training and test data, we follow the procedure similar to a previous study (Huang et al., 2006). For each dataset, we first compute the covariate mean $\bar{\mathbf{x}}$ of all data instances, and select n_{tr} data instances with probability of $p(\xi = 1|\mathbf{x}^{(i)}) = \exp\frac{-\left\|\mathbf{x}^{(i)}-\bar{\mathbf{x}}\right\|}{\sigma}$, where ξ is a binary indicator variable with 1 indicating training data, and σ is the standard deviation of $\left\|\mathbf{x}^{(i)} - \bar{\mathbf{x}}\right\|$, $\forall\mathbf{x}^{(i)}$. Remaining part of the dataset is considered for testing. The above method may be biased with the Gaussian form of the importance weighting function. Therefore, we also use another method for creating bias in the training data w.r.t the population, following (Sugiyama et al., 2008). We randomly choose one sample from the data pool between two consecutive change points and accept this as source sample with probability $min(1, 4 * (x^v))$, where x^v is the v^{th} feature of and instance x. Then we remove x from the pool regardless of its rejection or acceptance. Once 10% of data instances are selected as the source, we choose the rest as target. Finally, we concatenate the source and target data to simulate the respective streams.

Methods

Table 4.3 lists the competing methods considered for evaluation. It also shows the type of sampling methods used in each approach, where SR indicates sampling with replacement and SWR indicates sampling without replacement. As mentioned in §4.1, we use the original KMM (cenKMM) and the ensemble KMM (ensKMM) approaches as baseline methods for comparison. Here, ensKMM divides the test data into multiple components via sampling without replacement. We also consider the naive ensTrKMM method, mentioned in §4.1.3, as a baseline approach. It partitions the training data by sampling without replacement. Note that in Table 4.3, there are no methods that perform sampling with replacement (SR) on the test data, due to reasons given in §4.1.4.

Experiments

We first evaluate sensitivity of each competing method to input parameters, i.e., m and η (or s). Density ratio estimates depend on the training data sample size m for VFKMM and EVFKMM, and the number of test data partitions k for ensKMM and EVFKMM. Note that k is inversely proportional to the test data partition size. For the sake of comparison, we unify the notations m and k with the relation $m = \frac{n_{tr}}{k}$ (as given in Algorithm 4) for each training dataset, and vary the number of partitions with $k = \{5, 10, 15, 20\}$. In the case of VFKMM and EVFKMM, the effect of larger m on training data samples can be observed with smaller k value. In this set of experiments, we choose $\eta = 0.001$.

In the next set of experiments, we vary η to empirically demonstrate Corollary 2, with $k = 10$. Particularly, we vary η such that the resulting number of samples belong to $s = \{50, 100, 150, 200\}$. Here, instead of referring the results of these experiments with respect to η, we refer it with respect to s for clarity of understanding.

Importantly, the primary purpose of the method discussed in this section is to demonstrate the scalability of proposed approaches. Therefore, we vary the size of training dataset, i.e., we set $n_{tr} = \{100, 500, 1000, 1500, 2000\}$, and estimate density ratio using each competing method. Here, we use $k = 10$, $\eta = 0.001$.

Note that each of these sets of experiments were performed over all datasets listed in Table 4.2. Here, we only consider the first 50,000 instances in each dataset for simplicity. Moreover, a large amount of data merely increases the test dataset size.

We measure the goodness of estimated importance weights (denoted as $\hat{\beta}(\mathbf{x}) \in \hat{\boldsymbol{\beta}}$) by computing the Normalized Mean Square Error (NMSE). This is given by

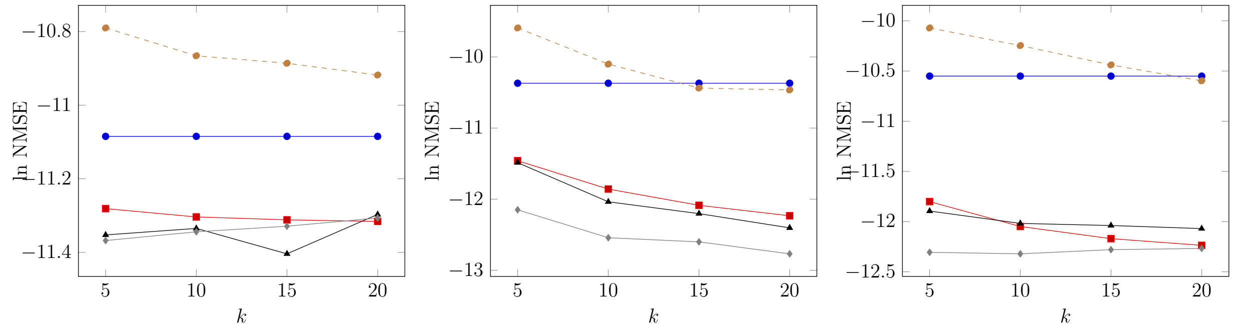
$$\frac{1}{n}\sum_{i=1}^{n}\left(\frac{\hat{\beta}(\mathbf{x}^{(i)})}{\sum_{j=1}^{n}\hat{\beta}(\mathbf{x}^{(j)})} - \frac{\beta(\mathbf{x}^{(i)})}{\sum_{j=1}^{n}\beta(\mathbf{x}^{(j)})}\right) \tag{4.8}$$

where $\beta(\mathbf{x}^{(i)}) = \frac{1}{p(\xi=1|\mathbf{x}^{(i)})}$, following (Miao et al., 2015). n indicates dataset size, which depends on the method used. For example, in cenKMM, $n = n_{tr}$ since it estimates density ratio for all training data, whereas in VFKMM and EVFKMM, $n \leq n_{tr}$ since density ratio is estimated for only those data instances which are selected during random sampling. We use the well-known QP solver in CVXOPT python library (Dahl and Vandenberghe, 2006) to execute the KMM quadratic program, with $B = 1000$ and $\epsilon = \frac{\sqrt{n_{tr}}-1}{\sqrt{n_{tr}}}$. Following (Huang et al., 2006), we use a Gaussian kernel with width γ equal to the median of pairwise distances.

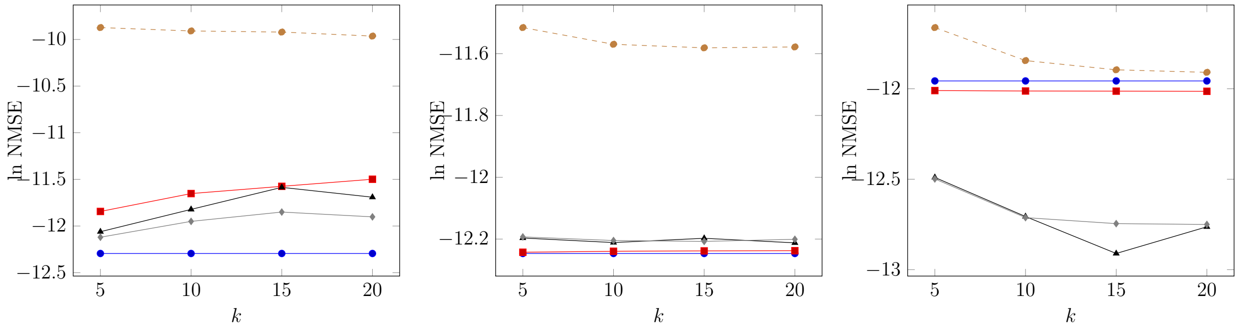
Results

We performed each experiment over 5 different sets of training and test data, generated via a random training data selection process, with 5 iterations in each to account for randomness in the sampling process. Moreover, we present our results on the natural logarithmic scale for clarity in comparison with competing methods.

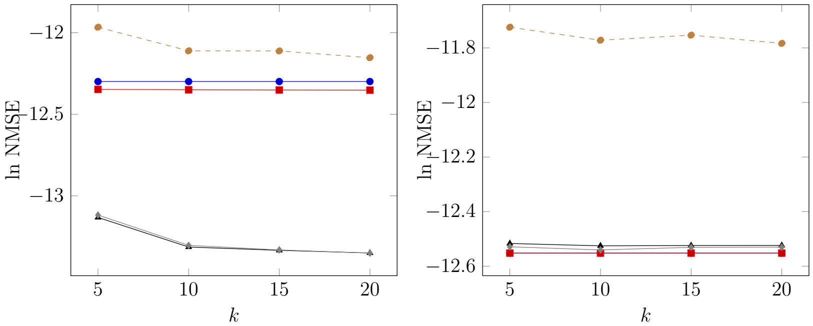
We first present results illustrating the behavior of VFKMM and EVFKMM when k (i.e., $m = \frac{n_{tr}}{k}$) is varied, and compare it with that of cenKMM, ensKMM and ensTrKMM. In all these experiments, we choose $n_{tr} = 500$. Figure 4.3 shows the ln NMSE obtained on various datasets. Here, lower value of k indicates larger m, resulting in lower value of s in VFKMM and EVFKMM. Also, a lower value of ln NMSE is desired. As seen



(a) ForestCover (b) KDD (c) PAMAP



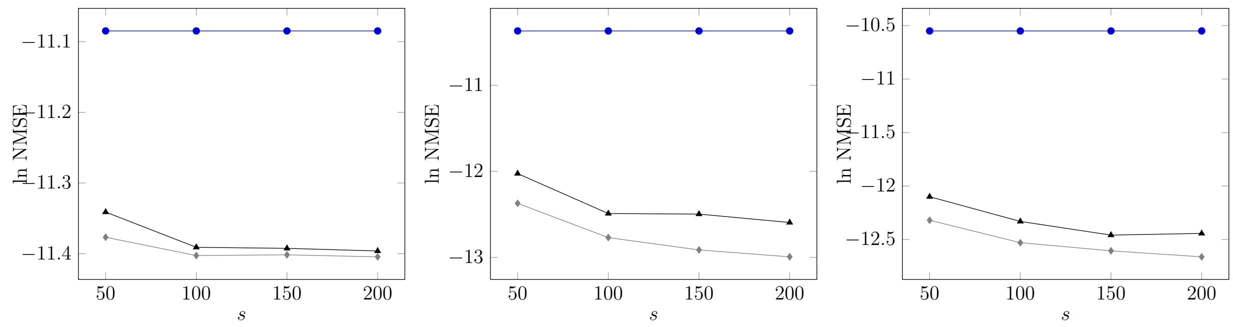
(d) PowerSupply (e) SEA (f) Syn002



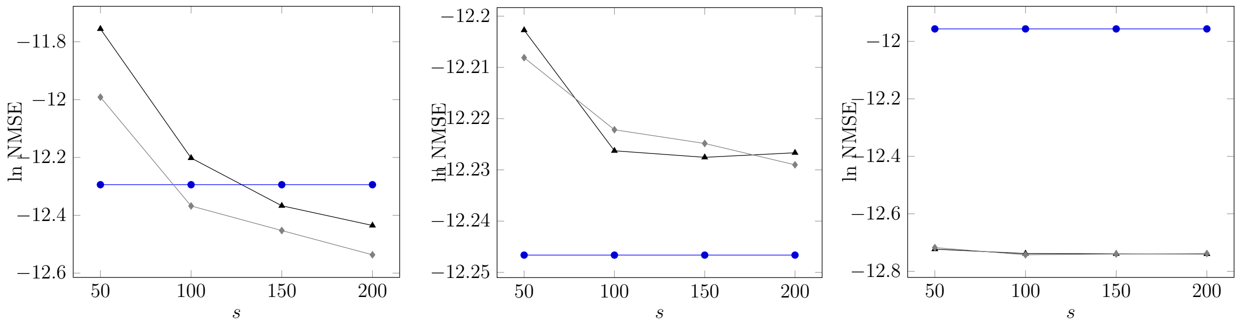
(g) Syn003 (h) MNIST

Figure 4.3: Logarithm of NMSE with different k ($\propto \frac{1}{m}$) on —▲— VFKMM and —◆— EVFKMM, compared to —●— cenKMM, —■— ensKMM and -●- ensTrKMM.

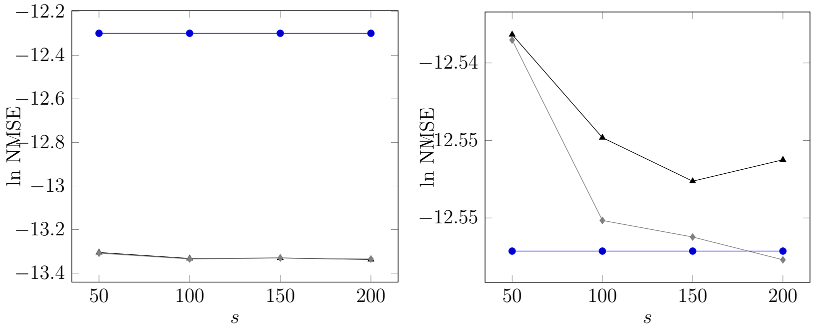
in these plots, VFKMM and EVFKMM resulted in better (smaller) ln NMSE than baseline methods, on most datasets. However, ln NMSE of baseline methods in a few datasets, including PowerSupply, SEA, and MNIST, is marginally better than the proposed methods. This mixed result is expected since the sampling-based methods on the training data approximates cenKMM, according to Corollary 1. Additionally, VFKMM and EVFKMM performs equivalently, with EVFKMM resulting in a better NMSE than VFKMM on multi-



(a) ForestCover (b) KDD (c) PAMAP



(d) PowerSupply (e) SEA (f) Syn002

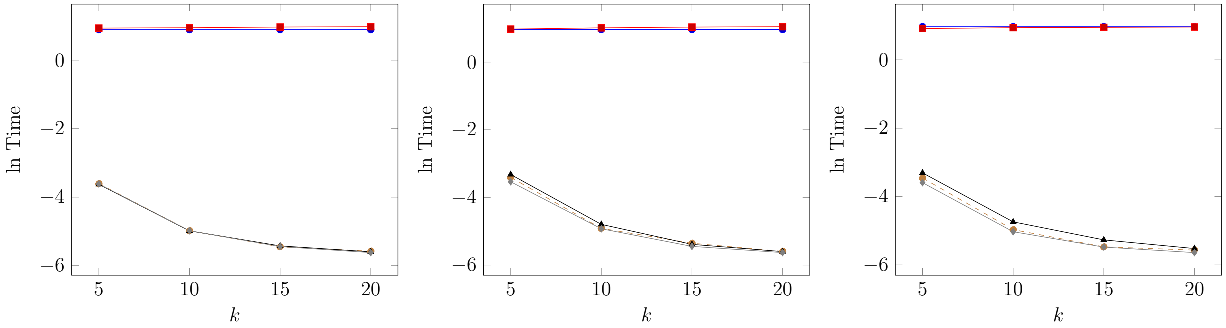


(g) Syn003 (h) MNIST

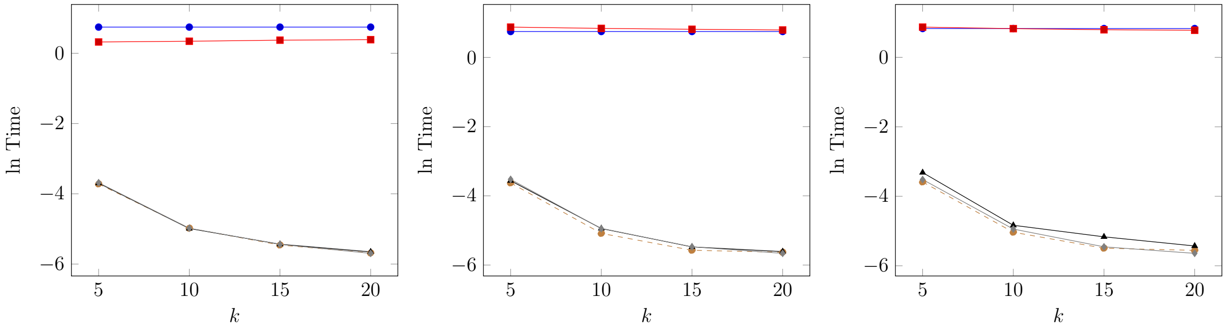
Figure 4.4: Logarithm of NMSE with different s on —▲— VFKMM and —◆— EVFKMM, compared to —●— cenKMM.

ple datasets including KDD, PAMAP, PowerSupply and MNIST. Importantly, ensTrKMM performs worst on all datasets. This supports our hypothesis that splitting (sampling without replacement) training data produces largely different data distributions compared to the original training data distribution.

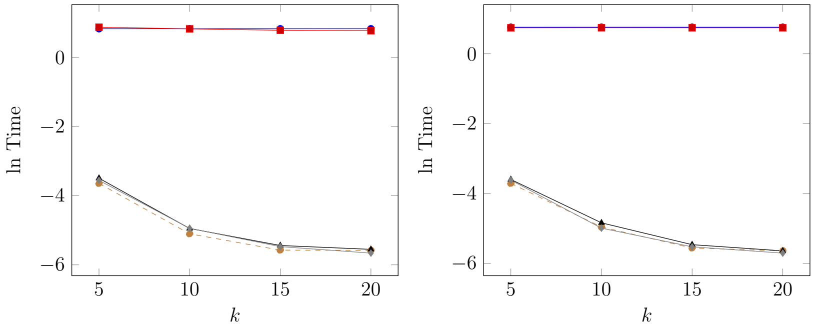
Figure 4.4 shows the effect of varying η (equivalently varying s) over all datasets using VFKMM and EVFKMM. Clearly, a larger number of samples results in a lower error for



(a) ForestCover (b) KDD (c) PAMAP



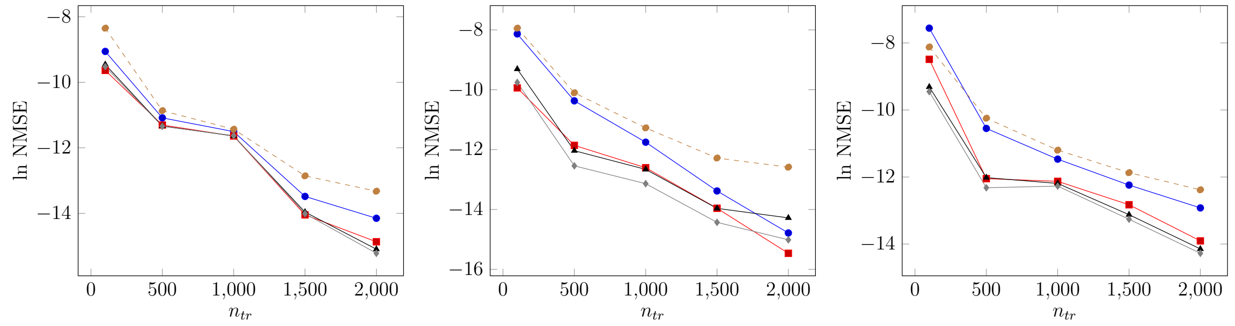
(d) PowerSupply (e) SEA (f) Syn002



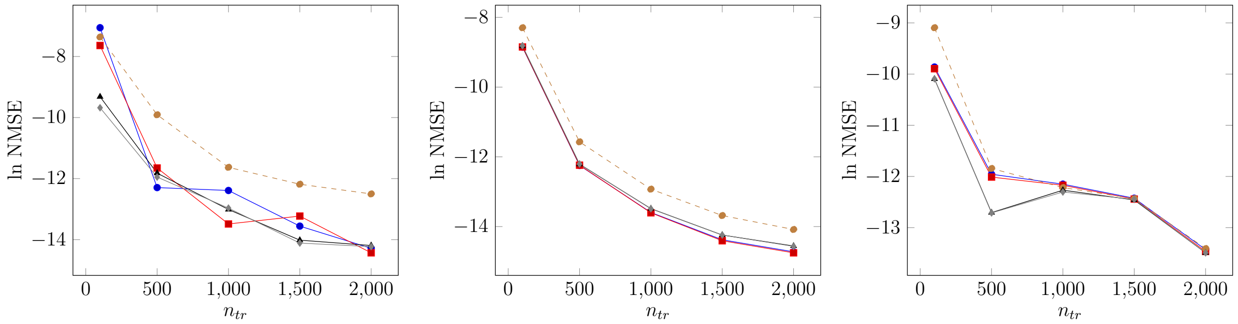
(g) Syn003 (h) MNIST

Figure 4.5: Average execution time (in logarithm scale) with different k on ▲ VFKMM and ◆ EVFKMM, compared to ● cenKMM, ■ ensKMM and ● ensTrKMM.

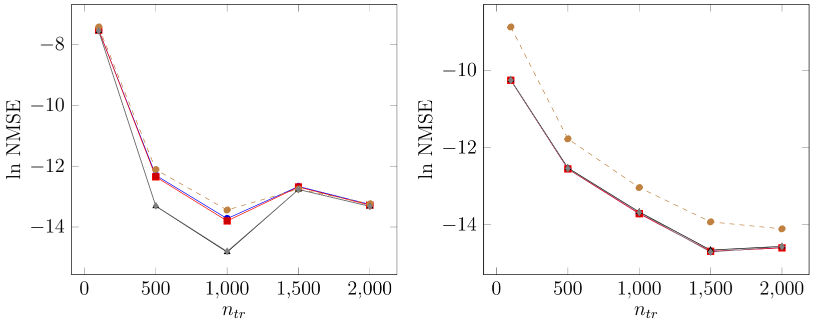
the two methods. Particularly, the proposed approach resulted in larger error than baseline on datasets such as Powersupply, SEA, and MNIST in Figure 4.3. With more number of samples, the decrease in error can be observed more distinctly in these datasets than others in Figure 4.4, following Corollary 2. Note that performance of VFKMM and EVFKMM are very similar in Figure 4.4f and Figure 4.4g.



(a) ForestCover (b) KDD (c) PAMAP



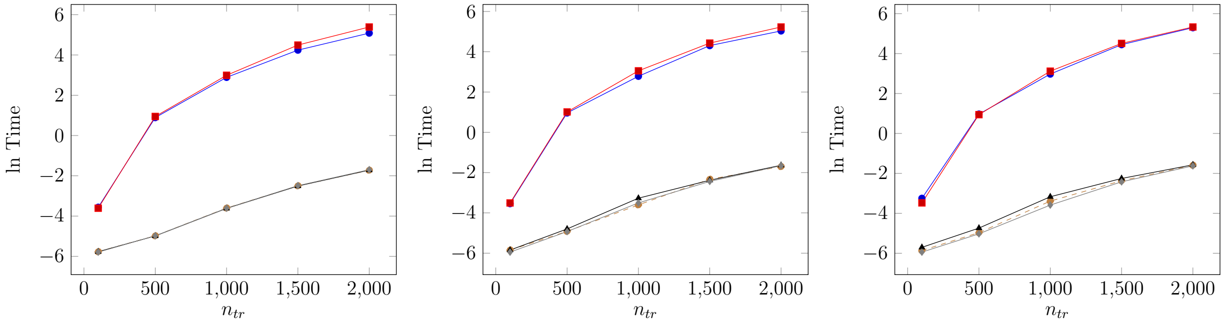
(d) PowerSupply (e) SEA (f) Syn002



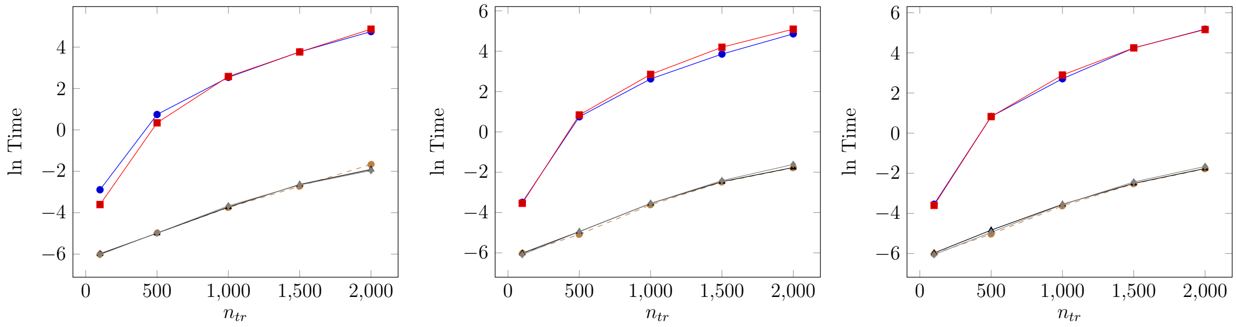
(g) Syn003 (h) MNIST

Figure 4.6: Logarithm of NMSE with different n_{tr} on —▲— VFKMM and —◆— EVFKMM, compared to —●— cenKMM, —■— ensKMM and - -●- - ensTrKMM.

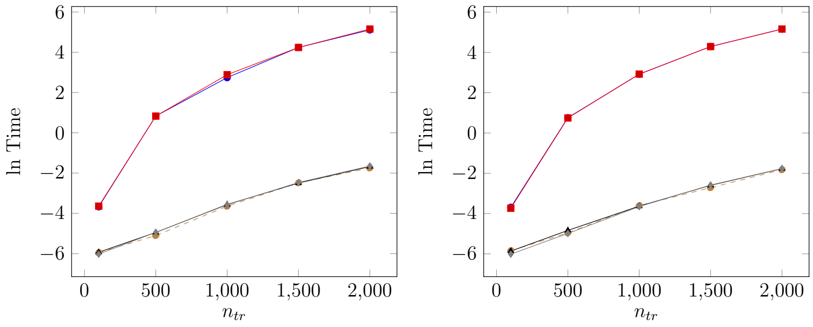
Nevertheless, the major advantage of performing sampling with replacement over training data can be observed in the lower execution time obtained on all datasets compared to baseline methods. Figure 4.5 illustrates this with the average run-time (in the natural logarithm of seconds) on each sample (or component) while computing density ratio. For example, the execution time with $k = 20$ on the ForestCover dataset for cenKMM and ensKMM was 2.43s and 2.65s respectively, while that of VFKMM and EVFKMM was



(a) ForestCover (b) KDD (c) PAMAP



(d) PowerSupply (e) SEA (f) Syn002



(g) Syn003 (h) MNIST

Figure 4.7: Average execution time (in logarithm scale) with different n_{tr} on ▲ VFKMM and ◆ EVFKMM, compared to ● cenKMM, ■ ensKMM and ● ensTrKMM.

only 0.0037s and 0.0036s respectively. On the logarithmic scale, this translates to 0.88 and 0.97 for cenKMM and ensKMM respectively, and −5.59 and −5.62 for VFKMM and EVFKMM respectively. The figure shows that with smaller m (equivalently larger k), execution time per sample drastically decreases. Here, time used is the average execution for QP, where it represents the time per component in ensKMM and the time per sample in VFKMM and EVFKMM. We assume that each component or sample can be executed in parallel.

The next set of experiments showcases the effect of NMSE and execution time on different training dataset size n_{tr}. Here, we set $k = 10$ (fixing m). Figure 4.6 shows the ln NMSE for all datasets considered on each competing method. From the plots, it can be observed that marginally equivalent results were obtained for each value of n_{tr} on all methods, following Figure 4.3. Moreover, increase in n_{tr} reduces error due to larger training sample size. As before, the benefit of VFKMM and EVFKMM is better observed in the execution time as illustrated in Figure 4.7. With increasing training set size, the execution time of all methods increases as expected due to the $\mathcal{O}(n_{tr}^3)$ time complexity of KMM. However, the execution time of VFKMM is very minimal as compared to other methods due to the smaller size of training data in each sample. For example, the execution time of cenKMM, ensKMM, VFKMM and EVFKMM on the MNIST dataset with $n_{tr} = 500$ was $2.13s$, $2.1s$, $0.008s$ and $0.0068s$ respectively. On the other hand, with $n_{tr} = 2000$, the execution time was $173.7s$, $174.4s$, $0.17s$ and $0.17s$ respectively. This example result can be observed on the corresponding logarithmic scale in Figure 4.7h. The dramatic gain in performance of VFKMM and EVFKMM is traded with more number of samples. Nevertheless, each of these samples can be utilized independently to compute density estimates in a parallel and distributed manner, thereby taking advantage of a distributed system. We leave this for future work. Clearly, VFKMM can be used in MSC (introduced in Chapter 3) instead of KMM to train a target classifier for scaling up to fast data streams. Yet, a major concern about KMM is its model selection process. In the next section, we will address this challenge over the multistream setting.

4.2 Kullback-Leibler Importance Estimation Procedure

In the MultiStream classification (or MSC) framework, the challenges of drift detection with bias correction is handled with the use of ensemble classifiers and two types of classifier training methods. This combination suffers from two main drawbacks.

- Management of two different classifier types within the ensemble is complex and may lead to degenerate cases, as illustrated by the difference between MSC and MSC2 in §3.4.
- The bias correction method (KMM) suffers from model selection problem. KMM parameters in Chapter 3 were chosen based on a heuristic. A systematic cross-validation approach may result in the parameters being biased towards the training distribution, as shown in (Sugiyama et al., 2008). Therefore, if the model parameters are not chosen well, it may negatively affect the overall classification accuracy.

In this section, we will first describe an alternative bias correction approach where the above two drawbacks are addressed. Unfortunately, this also suffers from scalability issues with regards to fast data streams. We will then address this challenge by appealing to ensemble methods.

4.2.1 FUSION

Similar to MSC, we utilized an alternative method for estimating importance weight in the multistream setting in a recent study (Haque et al., 2017). We call this technique as FUSION. The main idea in FUSION is to model the density ratio function as a linear combination of Gaussian kernels and adopt the batch-processing scheme of estimating the ratio called Kullback-Liebler Importance Estimation Procedure (KLIEP). Apart from its inherent model selection property, the density ratio model can be updated online. This update can be used as a measure of change in data distribution over time, and therefore can be used to detect concept drifts that directly affect the density ratio estimates.

To achieve this, FUSION maintains two fixed-size sliding windows, denoted by $\mathbf{W}_S$ and $\mathbf{W}_T$, for storing recent data instances in the source and the target streams respectively. It has four main modules, i.e., *Density Ratio Estimation* Module (DRM), *Drift Detection* Module (DDM), *Classification*, and the *Update* module. DRM is used to mitigate the covariate shift between the source and the target distributions by estimating the density ratio, i.e., importance weight (similar to MSC), directly for each data instance in the source sliding window. The objective of associating the importance weight with each source instance is to estimate the target distribution by the weighted source distribution. FUSION builds a classifier model based on the weighted source data instances. The classification module predicts a class label for any incoming test data instances in the target stream using the trained model. The DDM detects a concept drift if there is a significant change between the weighted source and the unweighted target distributions. Once a concept drift is detected, the Update module uses the recent instances from source sliding windows, along with the weights estimated by the DRM for training a new classification model.

FUSION uses a Gaussian kernel model for direct density ratio estimation (Sugiyama et al., 2007), as follows:

$$\hat{\beta}(x) = \sum_{j=1}^{N} \alpha_j K_\sigma(x, x_{te}^{(j)}) \tag{4.9}$$

where $\boldsymbol{\alpha} = \{\alpha_j\}_{j=1}^{N}$ are the set of N parameters needed to be learned, $K_\sigma(\cdot,\cdot)$ is the Gaussian kernel, i.e., $K_\sigma(x^{(i)}, x^{(j)}) = \exp\left\{-\frac{\left\|x^{(i)}-x^{(j)}\right\|^2}{2\sigma^2}\right\}$, and σ is the kernel width. Recent data instances in the target stream stored in $\mathbf{W}_T$ are used as the Gaussian kernel centers. The following objective function is solved to estimate the parameters of the model:

$$\begin{aligned}
&\underset{\{\alpha_j\}_{j=1}^{N}}{\text{maximize}} \left[\sum_{i=1}^{N} \log\left(\sum_{j=1}^{N} \alpha_j K_\sigma(x_T^{(i)}, x_T^{(j)})\right)\right] \\
&\text{subject to } \frac{1}{N}\sum_{i=1}^{N}\sum_{j=1}^{N} \alpha_j K_\sigma(x_S^{(i)}, x_T^{(j)}) = 1, \\
&and\ \alpha_1, \alpha_2, ..., \alpha_N \geq 0.
\end{aligned} \tag{4.10}$$

where N is the size of $\mathbf{W}_S$ and $\mathbf{W}_T$. Gradient ascent is used to learn the parameters. Following a batch learning scheme, FUSION also provides an online updating method for the parameters with each new source or target instance in the stream as follows.

$$\begin{cases} \hat{\alpha}_j^t \leftarrow (1-\eta\lambda)\hat{\alpha}_{j+1}^{t-1},\ j = 1, ..., N-1 \\ \hat{\alpha}_j^t \leftarrow \frac{\eta}{\beta(\hat{x}_T^{(N+1)})},\ j = N \end{cases} \tag{4.11}$$

Here, the j^{th} parameter is updated using the previous value that is held at the $j+1^{th}$ index after adding a new data instance into the window.

The DDM module detects a change between the weighted source and the target distribution as follows:

$$S = \sum_{j=1}^{N} \ln \frac{p_T(x_T^{(j)})}{\hat{\beta}_0 p_S(x_T^{(j)})} = \sum_{j=1}^{N} \ln \frac{\hat{\beta}_t(x_T^{(j)})}{\hat{\beta}_0(x_T^{(j)})} > \mu \tag{4.12}$$

where $\hat{\beta}_0$ and $\hat{\beta}_t$ are density ratios defined by $\boldsymbol{\alpha}^0$ and $\boldsymbol{\alpha}^t$ respectively, and μ is a user-defined threshold. Here, the subscripts 0 refers to an initial value, and t refers to the updated value at time $t > 0$. A change is detected if the likelihood ratio is greater than μ. Following detection of a concept drift, the parameters of the Gaussian kernel model are learned again using instances stored in $\mathbf{W}_S$ and $\mathbf{W}_T$. Subsequently, a new classifier model is trained on the weighted source instances as mentioned before. Apart from the task of classification, we recently extended this framework to regression (prediction over a continuous domain) as well (Haque et al., 2018).

Table 4.4: List of symbols for EnsFusion

Symbol	Description
α	Scalar parameters of Gaussian kernel model
$\mathbf{W}$	Stream window of size N
$\mathcal{V}$	Ensemble of density ratio estimation models
V	A density ratio estimation model in $\mathcal{V}$
$\mathcal{M}$	Ensemble of classifiers
k	Size of $\mathcal{V}$
m	Size of each t-minibatch
D	Output of local drift detection
μ	Local drift detection threshold

4.2.2 Challenges

The bottleneck of FUSION is the batch learning algorithm used to estimate the Gaussian kernel parameters. It has a time complexity of $\mathcal{O}(N^2)$. Although these parameters are updated online following the batch learning scheme, for each data instance encountered in S and T, until the next concept drift is detected, it puts an overhead on overall framework execution time, especially in case of frequent concept drifts. In this paper, we aim to address this computational bottleneck for multistream classification.

4.2.3 EnsFusion: Ensemble Multistream Classification

In FUSION, the computational overhead in solving the convex optimization problem (Equation 4.10) is quadratic in the number of data instances in the target window. Therefore, a significantly small target window size may have a major influence on the overall execution time but may affect prediction accuracy. Using this intuition, we now present the main idea of this section. For clarity, Table 4.4 lists frequently used symbols in this section of the dissertation, in addition to the symbols listed earlier (Table 3.1 and Table 4.1).

We divide data instances in the target window $\mathbf{W}_T$, at time t, into k non-overlapping sets containing sequential data instances. We call each set as a *t-minibatch*. We then perform parameter learning, online update, and change detection independently on each t-minibatch

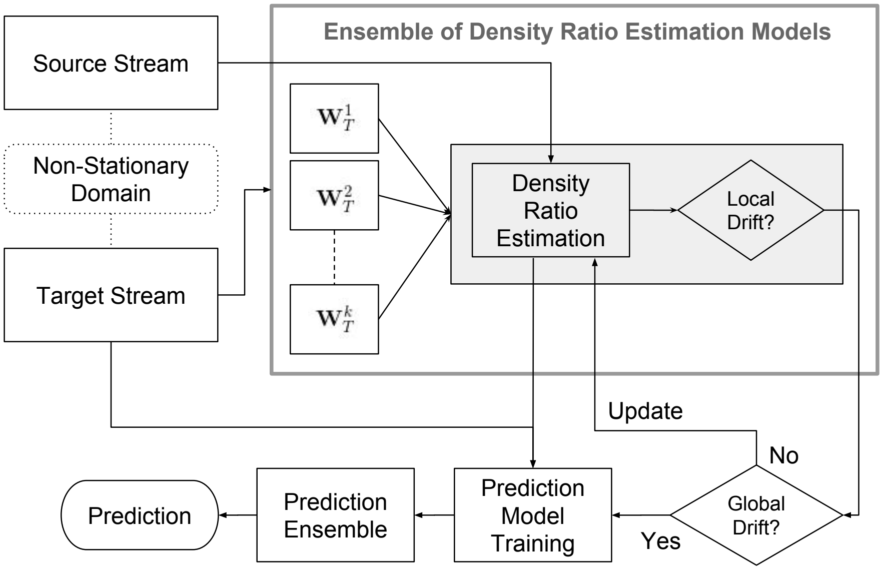


Figure 4.8: Overview of EnsFusion

of size $m < N$. These form the ensemble of k *density estimator* models, and is denoted by $\mathcal{V}$. We estimate importance weight for all source data instances from each of these models, i.e., we use all instances in $\mathbf{W}_S$ with each t-minibatch instances, and later combine the results by normalizing the weights on each source instance in $\mathbf{W}_S$. Similar to (Haque et al., 2017), we use an ensemble of classifier models, which is denoted by $\mathcal{M}$, to perform label prediction. Figure 4.8 illustrates this process, which we call *EnsFusion*.

Initialization

Initially, we generate a predictor using an ensemble $\mathcal{V}$ of Gaussian kernel models as follow. We split target window $\mathbf{W}_T$ into k non-overlapping and sequential t-minibatches, each denoted by $\mathbf{W}_T^i, \forall i \in [1, k]$, and $k < N$. For data instances in each $\mathbf{W}_T^i$, we create a Gaussian kernel model, denoted by $V_i \in \mathcal{V}$, using the formulation similar to Equation 4.10. Particularly, the convex optimization problem we solve in each V_i is as follows.

$$\begin{aligned}
&\underset{\{\alpha_j\}_{j=1}^{m}}{\text{maximize}} \left[\sum_{g=1}^{m} \log\left(\sum_{j=1}^{m} \alpha_i^j K_{\sigma_i}(x_{T_i}^{(g)}, x_{T_i}^{(j)})\right)\right] \\
&\text{subject to } \frac{1}{N}\sum_{g=1}^{N}\sum_{j=1}^{m} \alpha_i^j K_{\sigma_i}(x_S^{(g)}, x_{T_i}^{(j)}) = 1, \qquad (4.13)\\
&\text{and } \alpha_i^1, \alpha_i^2, ..., \alpha_i^m \geq 0.
\end{aligned}$$

where $\alpha_i^j \in \boldsymbol{\alpha}_i$ is the parameter associated with the j^{th} Gaussian kernel, m is the size of $\mathbf{W}_T^i$, N is the size of $\mathbf{W}_S$, and $x_{T_i}^{(j)}$ is the j^{th} instance in $\mathbf{W}_T^i$. Here, we estimate the kernel width σ_i through likelihood cross validation. Note that we use all data instances in $\mathbf{W}_S$ for each $\mathbf{W}_T^i$ to solve the optimization problem. This estimates the density ratio $\hat{\beta}_i(x)$ from $V_i \in \mathcal{V}$ for each source data instance x. We combine the density ratio estimates from each Gaussian kernel model in $\mathcal{V}$ through weighted average to form source instance weights. Concretely, the estimated density ratio is as follows.

$$\hat{\beta}(x) = \sum_{i=1}^{k} w_i(x)\hat{\beta}_i(x) \qquad (4.14)$$

where $w_i = p(x|x \in \mathbf{W}_T^i)$, and $\hat{\beta}_i(x) = \frac{p_T(x \in \mathbf{W}_T^i)}{p_S(x \in \mathbf{W}_S)}$. If m_i is the size of the i^{th} t-minibatch $\mathbf{W}_T^i$, then $w_i = \frac{m_i}{N}$. We then train a classifier using weighted training data from $\mathbf{W}_S$ and initialize the prediction ensemble $\mathcal{M}$ with this classifier.

Streaming Update

Once all detection of local drifts is complete, we then employ a strategy to combine the binary output of each local drift detection to address overall or global concept drift. Specifically, we follow the change approach of FUSION. If a change is detected globally, we first re-initialize each Gaussian kernel model using current data instances in $\mathbf{W}_T$ and $\mathbf{W}_S$ by updating its parameters and then use its output estimates to train a new classifier that updates the prediction ensemble $\mathcal{M}$. Finally, we use this updated ensemble of classifiers to predict the

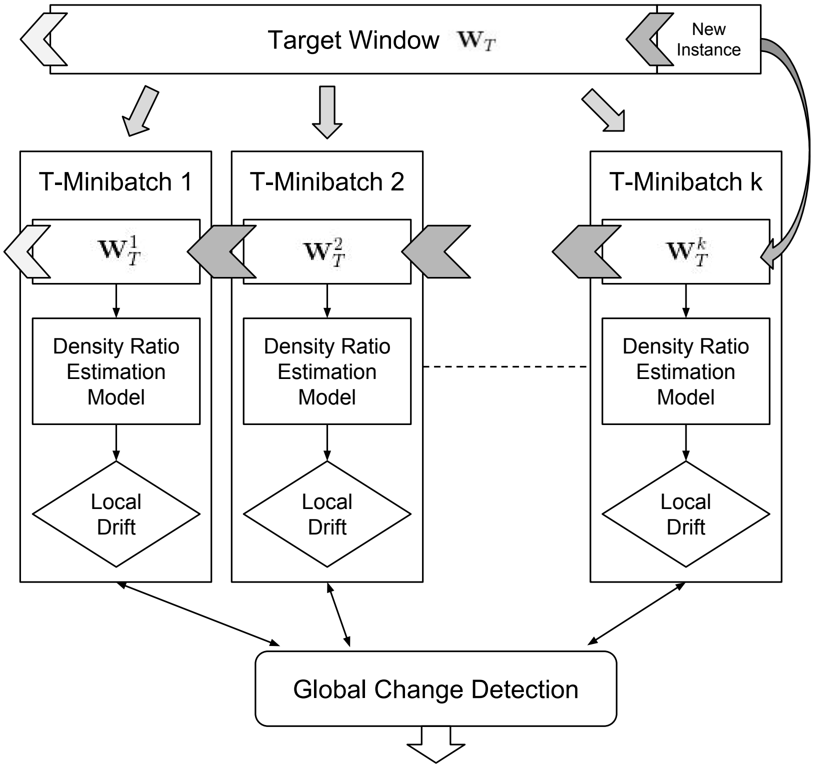


Figure 4.9: Updating of ensemble ($\mathcal{E}$), with local and global drift detection, upon arrival of a new target instance at time $t+1$.

class label of future instances in T. We now detail the update procedure of density ratio estimation ensemble and various strategies for global change detection.

Ensemble Update A new data instance can occur in either S or T. If the data instance belong to S, we update $\boldsymbol{\alpha}_i$ for satisfying constraint in Equation 4.13 for each model in the ensemble of density ratio estimators $\mathcal{V}$. Whereas, if the new data instance is associated with T, we first update each t-minibatch $\mathbf{W}_T^i$ and then update the corresponding parameter set $\boldsymbol{\alpha}_i$ for each $i \in [1, k]$. Concretely, we update each t-minibatch $\mathbf{W}_T^i$ for a new target instance as follows. Each $\mathbf{W}_T^i$ is formed by dividing the sequential data in $\mathbf{W}_T$ into k non-overlapping set. Here, we preserve the order of their appearances in $\mathbf{W}_T$ within each $\mathbf{W}_T^i$. Therefore, $\mathbf{W}_T$ and $\mathbf{W}_T^i$ can be viewed as a queue. Since the size of $\mathbf{W}_T$ is fixed, any new data instance arriving in T is appended to it at the end. This will remove the oldest data instance at

Algorithm 5: Multistream Prediction using Ensemble of Density Ratio Estimators.

```
Data: $S$: Source stream, $T$: Target stream
Input: $N$: Window size
Result: $\{y\} \forall x \in T$
begin
    Create $\mathbf{W}_S$ and $\mathbf{W}_T$
    Split $\mathbf{W}_T$ into $\mathbf{W}_T^i \forall i \in [1, k]$
    Initialize density ratio ensemble $\mathcal{V}$
    while $i = k$ do
        Generate $V_i \in \mathcal{V}$ using Eq. 4.10 from the combination of $\mathbf{W}_S$ and $\mathbf{W}_T^i$
        $i = i + 1$
    Estimate $\hat{\boldsymbol{\beta}}(\mathbf{W}_S)$ using Eq. 4.14
    Initialize predictor ensemble $\mathcal{M}(\boldsymbol{\beta}, \mathbf{W}_S, \mathbf{W}_T)$
    while $T$ exists do
        Get new data instance $x$
        if $x \in S$ then
            Update parameters $\boldsymbol{\alpha}_i$ for each $V_i \in \mathcal{V}$
        if $x \in T$ then
            Predict label of $x$ using $\mathcal{M}$
            Update each $V_i \in \mathcal{V}$ by sliding window
            Update parameters $\boldsymbol{\alpha}_i \in V_i$
        Check local drift $D_i$ using $\boldsymbol{\beta}^i$ from $V_i \in \mathcal{V}$
        change: Check $D$ using Eq. 4.15 or Eq. 4.16
        if change then
            Reinitialize each $V_i \in \mathcal{V}$ by recomputing $\boldsymbol{\alpha}_i$
            Compute $\hat{\boldsymbol{\beta}}$ using Eq. 4.14
            Update predictor ensemble $\mathcal{M}$
```

the start of the queue if the size of $\mathbf{W}_T$ is N. Indices of remaining instances in the queue are adjusted to accommodate the new data instance. We follow this queue-based update operation within and across each t-minibatch $\mathbf{W}_T^i$. Figure 4.9 illustrates the update process for each $\mathbf{W}_T^i$, where $i \in [1, k-1]$. Each $\mathbf{W}_T^i$ receives a new instance from the $i + 1^{th}$ t-minibatch. The new instance is added to the k^{th} t-minibatch, and the oldest instance in the 1^{st} t-minibatch is discarded. Since the associated Gaussian kernel model for $\mathbf{W}_T^i$ uses each instance in the i^{th} t-minibatch as Gaussian centers, every new instance in T affects the parameter set $\boldsymbol{\alpha}_i$ in each model $V_i \in \mathcal{V}$. We update these models using Equation 4.11. The resulting density estimates are used for local drift detection using Equation 4.12.

Global Change Detection The result of the drift detection module is binary, where 1 indicates that a concept drift is detected, and 0 indicates otherwise. Since we have k such decisions from the ensemble $\mathcal{V}$, we can employ various strategies to combine the results. Here, we show 2 different strategies. First, we say that a concept drift is detected if the output of any $V_i \in \mathcal{V}$ is 1. We call this a *one-vote* strategy. Formally, if D is the k dimensional decision vector where each element corresponds to a drift detection output for each $V_i \in \mathcal{V}$, then a change is detected if,

$$\sum_{i=1}^{k} D_i > 0 \tag{4.15}$$

Alternatively, the second strategy is *majority-vote*, where a change is detected if the following is true.

$$\sum_{i=1}^{k} D_i > \frac{k}{2} \tag{4.16}$$

Clearly, we can expect the majority-vote strategy to detect less number of drifts than the one-vote strategy for $k > 2$. Moreover, the single-vote strategy for such values of k can overestimate the number of concept drifts, and trigger model re-training. This is disadvantageous if the classifier re-training is computationally expensive.

Algorithm 5 lists the steps involved in our proposed method for efficiently predicting over a multistream setting. The ensemble containing density ratio estimators (denoted $\mathcal{V}$) is of size k, where $V_i \in \mathcal{V}$ is the i^{th} density estimator trained using $\mathbf{W}_S$ and $\mathbf{W}_T^i$ as before. Whereas, the ensemble of classifiers (denoted $\mathcal{M}$) contain a maximum of L base predictors trained on weighted source data instances at different times along the stream. We maintain the top L best performing classifier based on classifier confidence (Chandra et al., 2016a). Once k density ratio estimators are initialized, new incoming data instances update or re-initialize the parameters of $V_i \in \mathcal{V} \forall i \in [1, k]$. Particularly, $\mathbf{W}_T^i$ is updated with a new

Gaussian center if the data instance is from T. Whereas, only existing parameters $\boldsymbol{\alpha}_i$ are updated if the instance is from S. Once the output of all k local drift detection mechanism is obtained, the decision to update models in $\mathcal{V}$ and $\mathcal{M}$ is considered.

4.2.4 Analysis of EnsFusion

We now study some theoretical properties of the approach, along with its computational complexity.

Convergence Rate

In EnsFusion, we assume that data instances in T are generated in an i.i.d fashion from a non-stationary process. Therefore, each of the k t-minibatches $\mathbf{W}_T^i$ created by dividing these instances in a window $\mathbf{W}_T$ can also be assumed i.i.d. This indicates that each $\hat{\boldsymbol{\beta}}_i$ from $V_i \in \mathcal{V}$ at time t are independent estimates. We now show that the weighted average combination of $\hat{\boldsymbol{\beta}}_i \forall i \in [1, k]$ according to Equation 4.14 has a similar rate of convergence as the original algorithm in FUSION.

We assume a Reproducing Kernel Hilbert Space (RKHS) for estimating $\boldsymbol{\beta}_i$ in each $V_i \in \mathcal{V}$. We denote this RKHS by $\mathcal{H}$. Let $E_i^j(\beta)$ be the empirical error from $\mathbf{W}_T^i$ in V_i. From Equation 4.13, we estimate $\hat{\beta}_i$ by minimizing $\sum_{j=1}^{m} E_i^j(\beta)$. According to (Kivinen et al., 2010), this may lead to overfitting. Therefore, we estimate $\hat{\beta}$ by minimizing the regularized empirical error given by,

$$\tilde{E}_i^j(\beta) = -\log \beta_i(x_{T_i}^j) + \frac{\lambda}{2}\|\beta_i\|_{\mathcal{H}}^2 \tag{4.17}$$

where $\lambda > 0$ is a regularization parameter, and the norm in $\mathcal{H}$ is denoted by $\|\cdot\|_{\mathcal{H}}^2$. This empirical estimate is used to update V_i using Equation 4.11. Since we average over k independent models, the overall empirical error is given by,

$$\tilde{E}^j(\hat{\beta}) = \frac{1}{k}\sum_{i=1}^{k} \tilde{E}_i^j(\hat{\beta}_i) \tag{4.18}$$

Here, we show that the overall empirical error $\tilde{E}^j(\hat{\beta})$ is strongly convex, and derive a bound, similar to (Haque et al., 2017). We follow the justification that strong convexity enables improvement in regret bound (Shalev-Shwartz et al., 2009).

For simplicity, we assume that the size of each t-minibatch is m, i.e., $m_i = m \forall i \in [1, k]$ or $m = \frac{N}{k}$.

Lemma 4. *$\tilde{E}^j(\hat{\beta})$ is strictly convex, i.e., if $\hat{\beta}' \neq \hat{\beta}$, then,*

$$\tilde{E}^j(t\hat{\beta}' + (1-t)\hat{\beta}) < t\tilde{E}^j(\hat{\beta}') + (1-t)\tilde{E}^j(\hat{\beta})$$

Proof. In Equation 4.17, both negative logarithm and regularization terms are strictly convex functions. Therefore, the local regularization error functions $\tilde{E}_i^j(\beta)$ for $i \in [1, k]$ are strictly convex. Following the property of convex functions (Boyd and Vandenberghe, 2004), sum of convex functions are convex. Since each $\tilde{E}_i^j(\beta)$ in Equation 4.18 is strictly convex, $\tilde{E}_i(\beta)$ is strictly convex. □

For strong convexity of $\tilde{E}^j(\hat{\beta})$, we show that $\exists \theta > 0$ such that, $\tilde{E}^j(\hat{\beta}) - \theta \|\tilde{E}^j(\hat{\beta})\|^2$ is convex. Following the proof of Lemma 4, it can be seen that both the logarithmic term and the quadratic term of Equation 4.17 is strongly convex. Using the additive property of strong convexity and triangular inequality, $\tilde{E}^j(\hat{\beta})$ is strongly convex.

Let β^* be the optimum value. We assume that $\tilde{E}_i^j(\beta)$ is μ-smooth in the neighborhood of β^*. We desire to find an upper bound for $\|\tilde{E}^j(\hat{\beta}) - \tilde{E}^j(\beta^*)\|$. Within each $V_i, \forall i \in [1, k]$, the regularized empirical error $\tilde{E}_i^j(\hat{\beta}_i)$ is $Q-$Lipschitz continuous. Following (Shalev-Shwartz et al., 2009),

$$\mathbb{E}[\|\hat{\beta}_i - \beta^*\|^2] \leq \frac{4Q^2}{m\lambda} \tag{4.19}$$

where L is the Lipschitz constant.

Lemma 5. *Let $\tilde{E}_i^j$ be a strongly convex function and Q-Lipschitz continuous for $i \in [1, k]$ where $m = \frac{N}{k}$, and $\mathbb{E}[\|\bar{\beta} - \beta^*\|^2] \leq \frac{4Q^2}{m\lambda}$. Then, with learning rate $\eta = \frac{1}{m\lambda}$ in each $V_i \in \mathcal{V}$, the following inequality holds,*

$$\mathbb{E}[\|\hat{\beta} - \beta^*\|^2] \leq \frac{8Q^2}{N\lambda} \tag{4.20}$$

Proof. From Equation 4.14,

$$\begin{aligned} \mathbb{E}[\|\hat{\beta} - \beta^*\|^2] &= \mathbb{E}[\|\frac{1}{k}\sum_{i=1}^{k} \hat{\beta}_i - \beta^*\|^2] \\ &= \frac{1}{k^2}\mathbb{E}[\sum_{i=1}^{k} \|\hat{\beta}_i - \beta^*\|^2] \end{aligned} \tag{4.21}$$

Since we assume that $\tilde{E}_i^j$ is independent $\forall i \in [1, k]$,

$$\begin{aligned} \mathbb{E}[\sum_{i=1}^{k} \|\hat{\beta}_i - \beta^*\|^2] =& \sum_{i=1}^{k} \mathbb{E}[\|\hat{\beta}_i - \beta^*\|^2] \\ &+ \sum_{i \neq j} \mathbb{E}[\|(\hat{\beta}_i - \beta^*)(\hat{\beta}_j - \beta^*)\|] \end{aligned} \tag{4.22}$$

Therefore,

$$\begin{aligned} \mathbb{E}[\|\hat{\beta} - \beta^*\|^2] &\leq \frac{2}{k^2}\sum_{i=1}^{k} \mathbb{E}[\|\hat{\beta}_i - \beta^*\|^2] \\ &\leq \frac{2}{k}\mathbb{E}[\|\bar{\beta} - \beta^*\|^2] \end{aligned} \tag{4.23}$$

This is combined with inequality 4.19, and $m = \frac{N}{k}$. □

Finally, using Lemma 4 and 5 with (Shalev-Shwartz et al., 2009), we can see that the convergence rate is similar to FUSION, i.e.,

$$\mathbb{E}[\tilde{E}^j(\hat{\beta}) - \tilde{E}^j(\beta^*)] \leq \mathcal{O}(\frac{1}{N}). \tag{4.24}$$

Complexity

We assume that the size of the source window $\mathbf{W}_S$ at time t is N. The density ratio estimation ensemble $\mathcal{V}$ contains k models. We estimate density ratio for each source instance in $\mathbf{W}_S$ at time t from each of these models using the smaller target window $\mathbf{W}_T^i$, where $i \in [1, k]$. For simplicity of analysis, we assume that each $\mathbf{W}_T^i$ consists of equal number of data instances. Therefore, $w_i = \frac{1}{k}$ in Equation 4.14. The size of each t-minibatch is $m = \frac{N}{k}$.

Within each density estimation module, the four main operations are learning model parameters $\boldsymbol{\alpha}$, updating them, computing $\boldsymbol{\beta}$, and detecting drift. Learning $\boldsymbol{\alpha}$ involves solving Equation 4.13. With source window of size N and target window of size m, its time complexity is $\mathcal{O}(Nm + m^2)$. The update operation has a time complexity of $\mathcal{O}(N)$, while that of $\boldsymbol{\beta}$ computation and drift detection is each $\mathcal{O}(Nm)$. Overall, the time complexity of each density estimation model is $\mathcal{O}(Nm + m^2)$. We assume that the classifier time complexity is $\mathcal{O}(N + m)$. In terms of N and k, the overall time complexity is $\mathcal{O}((\frac{k+1}{k^2})N^2)$ since each density ratio estimation module can potentially run in parallel.

In the case of space complexity, we use memoization techniques that save previously computed terms since the streaming updates only involve a single new data instance within each density ratio model (the queuing update illustrated in Figure 4.9). Therefore, this consumes $\mathcal{O}(N^2 + m^2)$ space. We assume that the classifier consumes linear space with respect to the data instances in the source. Therefore, to save k density ratio estimation models, the overall space complexity is $\mathcal{O}((\frac{k^2+1}{k})N^2)$.

With the value of k as constant, both space and time complexity is $\mathcal{O}(N^2)$. But, in this paper, we achieve scalability of performing classification on the multistream setting by choosing an appropriate value for N and k.

Table 4.5: Dataset description for EnsFusion.

Identifier	**# features**	**# classes**	**# source instances**	**# target instances**
ForestCover	54	7	14,774	133,537
PAMAP	53	6	14,856	134,368
KDD	34	23	19,866	179,445
Electricity	6	2	4,296	38,916
Syn002	70	7	9,890	89,579
Syn003	70	7	9,835	89,249
SynEd	40	12	9,724	87,828

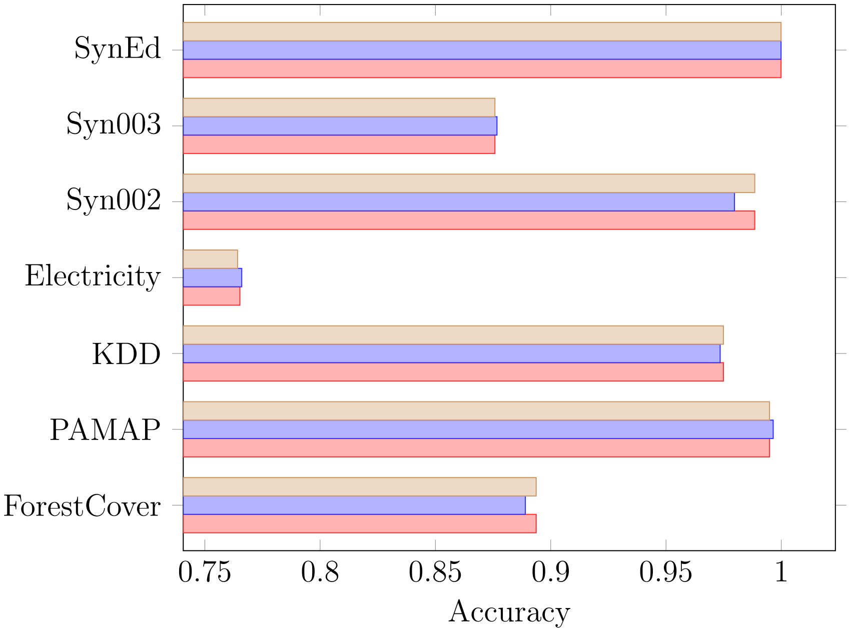


Figure 4.10: Accuracy of Multistream Classification, comparing EnsFusion-One (🟥), EnsFusion-Max (🟦), and Fusion (🟫).

4.2.5 Empirical Evaluation of EnsFusion

In this section, we discuss the dataset used to evaluate multistream classification, present experimental settings and corresponding results.

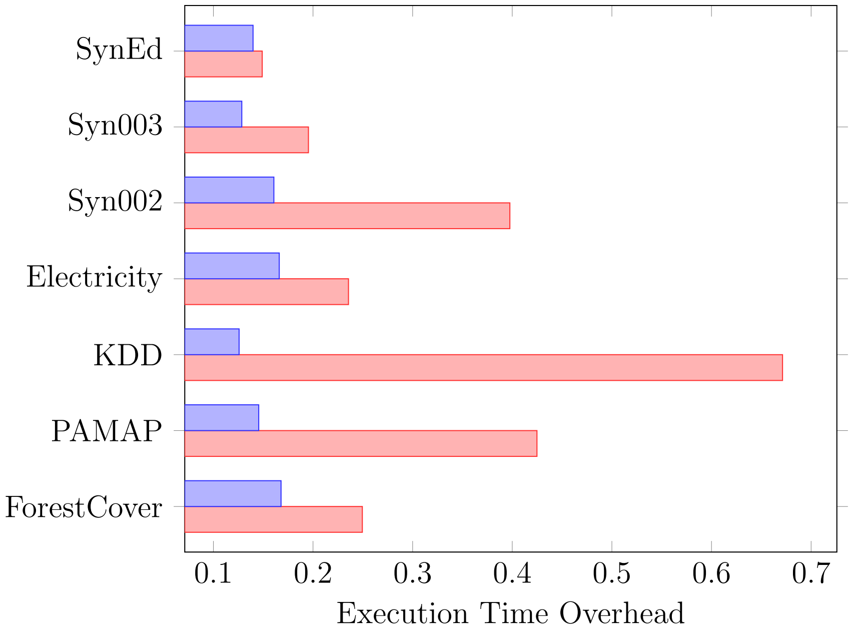
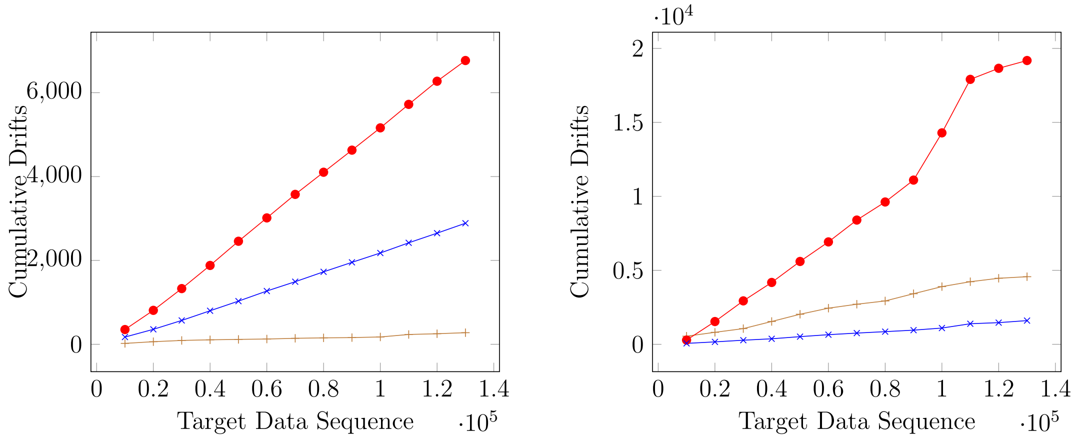


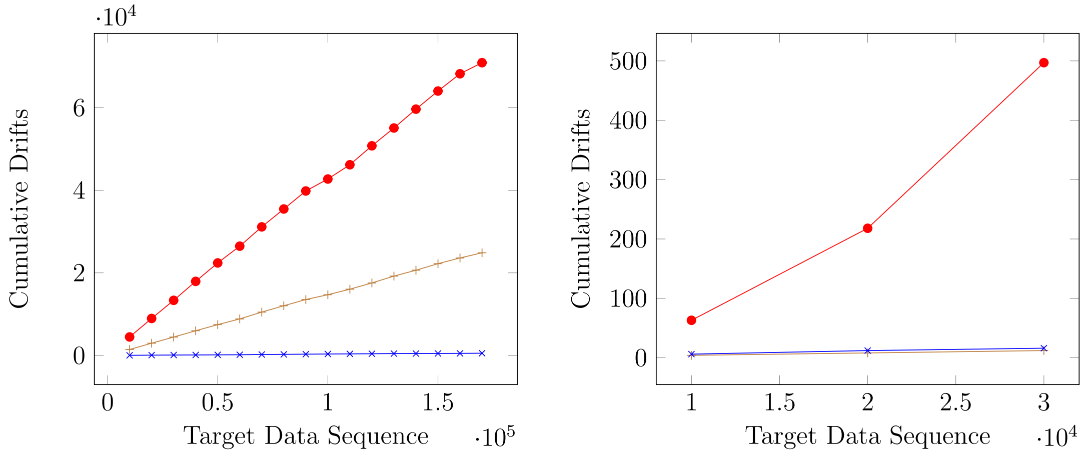
Figure 4.11: Execution time overhead of Multistream Classification, comparing EnsFusion-One (🟥) and EnsFusion-Max (🟦).

Dataset

Table 4.5 lists the publicly available (Lichman, 2013) real-world and synthetic datasets used to evaluate the proposed approach, similar to the dataset using previously for ease of comparison. Here, we normalize all datasets and re-shuffle them randomly, independent of its class label. Importantly, we split each data set into a biased-source stream and a target stream following recommendations from previous studies in covariate shift adaptation (Huang et al., 2006). Concretely, we train a Naive Bayes classifier to predict class labels in online minibatches, while monitor its performance using ADWIN (Bifet and Gavalda, 2007). If a change is detected by ADWIN within a minibatch, we probabilistically associate each instance x in the minibatch to either $\mathbf{W}_S$ or $\mathbf{W}_T$ according to a variable denoted by ζ, i.e., $p(\zeta = 1|x) = \exp \frac{||x-\bar{x}||^2}{2\sigma^2}$ indicates association to S. Here, $\bar{x}$ is the mean value of instances in the minibatch, and σ is its standard deviation. We select a small portion (denoted by $\%n$) of $\mathbf{W}_S$ to be concatenated to S that forms the source stream. Whereas, we concatenate all



(a) ForestCover (b) PAMAP



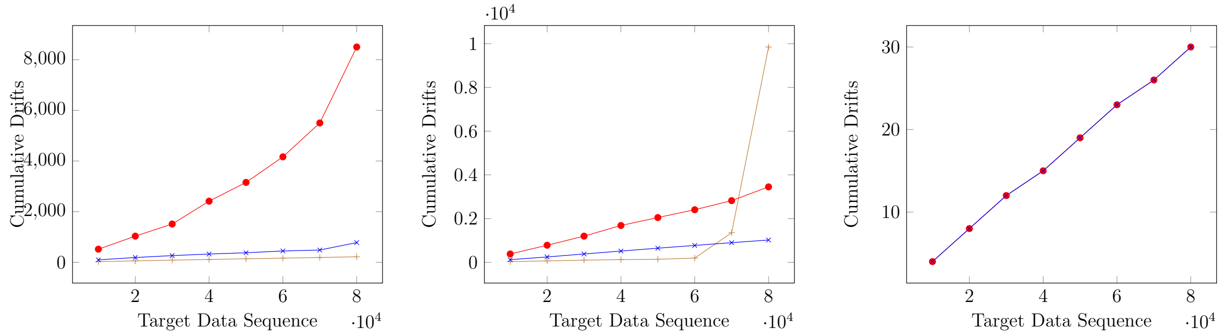
(c) KDD (d) Electricity

Figure 4.12: Cumulative number of drifts detected along T for real-world data sets by competing methods on multistream classification: —+— *Fusion*; —●— *EnsFusion-One*; —×— *EnsFusion-Max*.

instances in $\mathbf{W}_T$ to form the target stream. Table 4.5 mentions the size of the source and target data stream we consider during evaluation.

Experiments

We measure the performance of multistream classification using prediction accuracy and execution time. We compare the results of our approach with a baseline approach that does not employ an ensemble of Gaussian kernel models (Haque et al., 2017). This is denoted



(a) Syn002 (b) Syn003 (c) SynEd

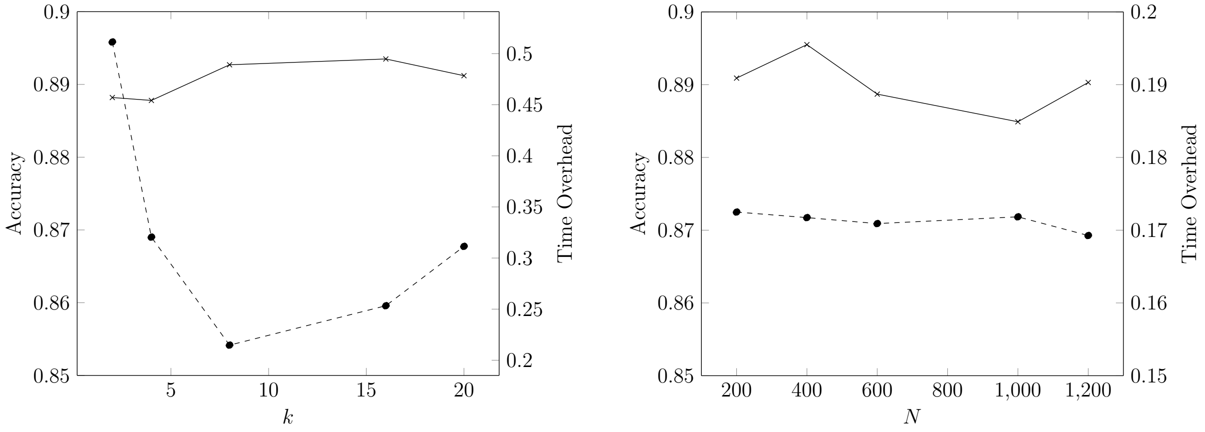
Figure 4.13: Cumulative number of drifts detected along T for synthetic data sets by competing methods on multistream classification: —+— *Fusion*; —●— *EnsFusion-One*; —×— *EnsFusion-Max*.

by *Fusion*, while our proposed approach is denoted by *EnsFusion*. As mentioned in §4.2.3, we study the performance of our ensemble method on two different global change detection strategy. We denote the method where we follow the strategy according to Equation 4.15 by *EnsFusion-One*. Whereas, we denote the method that follow the majority-vote strategy according to Equation 4.16 by *EnsFusion-Max*.

Since our approach requires the user to specify a number of parameters, we study the sensitivity of EnsFusion to different parameter values. These parameters include density ratio ensemble size (k), size of the source and target window (N), drift detection threshold (τ), and classifier ensemble size (L). Particularly, we assume the size of each t-minibatch $m = \frac{N}{k}$. Therefore, lower value of k implies a larger value of m and visa versa.

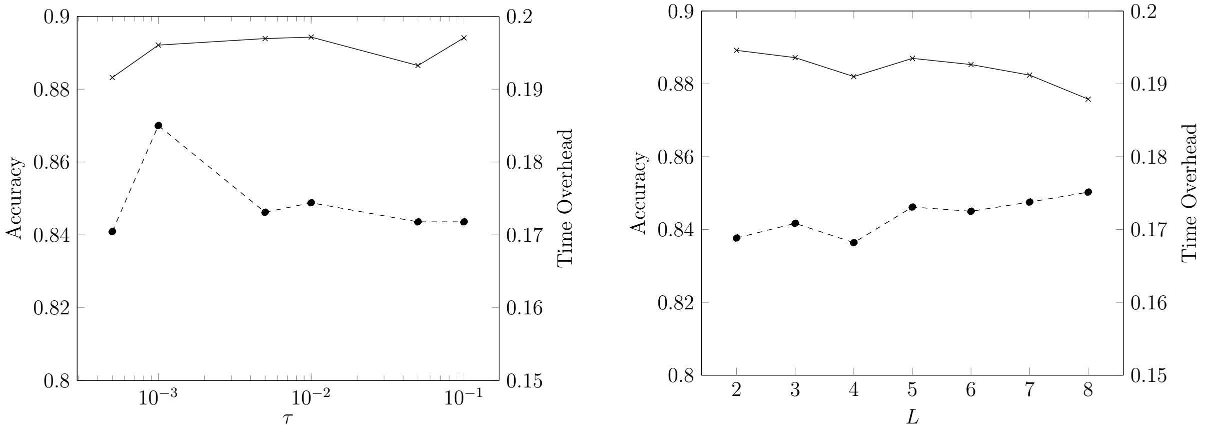
Setup

We have implemented our approach using Python v2.7. Our experiments were performed on a system running Ubuntu 16.04 and having 8 cores with 2.4Ghz and 64GB of RAM. We use the weighted SVM with RBF kernel as our base classifier, similar to (Haque et al., 2017). We select the best Gaussian kernel width in each model of density ratio ensemble $\mathcal{V}$ through likelihood cross-validation. By default, we use $N = 800$, $L = 1$, $\tau = 0.0001$, and



(a) Size of $\mathcal{E}$.

(b) Window Size



(c) Drift Detection Threshold

(d) Size of $\mathcal{M}$

Figure 4.14: Parameter sensitivity of EnsFusion-Max on ForestCover data set as an example. (—×— Accuracy; - ● - Time Overhead).

$k = 10$. Furthermore, we use the regularization parameter $\lambda = 0.01$ and learning rate of $\eta = 1$, following (Kawahara and Sugiyama, 2012) and (Haque et al., 2017).

Results

We now present the results of our experiments. Instead of reporting exact execution time, we report the time overhead, which is the ratio of execution time by EnsFusion (EnsFusion-One or EnsFusion-Max) to that of Fusion, i.e., $o = \frac{time(EnsFusion)}{time(Fusion)}$. This provides better insights on the computational time improvements of our approach compared to Fusion.

Overall Figure 4.10 shows the overall accuracy obtained on each competing method for different datasets. Clearly, each method has equivalent performance in terms of classification accuracy, with EnsFusion-Max performing marginally better than Fusion on all datasets. However, the major benefit of our approach is in its execution time. Figure 4.11 shows the execution time overhead of EnsFusion-One and EnsFusion-Max with respect to Fusion of different datasets. A time overhead value closer to 1 indicates equivalent execution time with the Fusion method, and any value less than 1 indicates performance gain. Clearly, both strategies of our method have a significant gain in execution time performance, with a maximum overhead of 67% by EnsFusion-One on the KDD data set. Particularly, EnsFusion-Max exceedingly performs better than EnsFusion-One on all data sets, with a maximum overhead of 16.7% on the ForestCover data set. This implies that EnsFusion-Max is at least 83% faster than Fusion to classify all target data instances in each data set while achieving equivalent accuracy. Note that the actual execution time (averaged over all data sets) to classify a single target data instance for Fusion was $1.37s$. While average execution time per instance for EnsFusion-One and EnsFusion-Max was $0.45s$ and $0.20s$ respectively.

Figure 4.12 shows the cumulative number of concept drifts detected on the target stream over time for each real-world data set, while Figure 4.13 shows those for synthetic data sets. In both these figures, the cumulative number of drifts is denoted by *Cumulative Drifts*. Since our approach locally detects concept drifts on each density ratio model in $\mathcal{V}$, the density ratio in Equation 4.12 is affected by the size of $\mathbf{W}_T^i$ for each $i \in [1, k]$. As mentioned in §4.2.3, EnsFusion-One performs global changes even when one of the density ratio models detects a change locally. This can be observed in the figure where EnsFusion-One detects more number of concept drifts than Fusion over time. Particularly in PAMAP and KDD data sets, the number of drifts detected by EnsFusion-One is extremely high. However, a number of drifts detected by EnsFusion-Max are less than EnsFusion-One, and also less than Fusion in the PAMAP and KDD datasets. Note that for SynEd data set in Figure 4.13, the number of concept drifts detected in each method is equal. Similarly for the Electricity dataset in Figure 4.12, the cumulative drift count for Fusion and EnsFusion-Max are nearly equal. Therefore, these figures show overlapping plots of cumulative drifts.

Parameter Sensitivity The EnsFusion approach has multiple parameters as user input. We now study its sensitivity to classification accuracy and execution time. Here, we report results obtained on the ForestCover dataset as an example. We observe similar behavior on all other data sets. Note that we keep the default settings for all experiments except when specified. Also, we measure time overhead with the Fusion approach executed on default settings.

Size of $\mathcal{E}$ Size of the ensemble containing density ratio estimation models (denoted by k) is set by the user. Clearly, a larger value of k provides smaller minibatch size but creates more number of density ratio models in the ensemble $\mathcal{V}$. Figure 4.14a shows that the accuracy of the model on different values of k are marginally equivalent. Though the accuracy is not largely affected by the choice of k, the execution time differs significantly when the size of the data set is large. The figure shows that most gain in execution time is at $k = 8$. However, the gain reduces with increase in k due to the overhead of computations performed in the increased number of density ratio estimation models. This can be overcome by performing model computations through parallel execution at time t. We leave this for future work.

Window size In our experiments, we assume that the source and target windows ($\mathbf{W}_S$ and $\mathbf{W}_T$) are of size N. This size is the length of the time period containing the latest data instances in respective streams. When a new data instance arrives, the oldest instance is removed and the new instance is added to the queue. Figure 4.14b shows that our approach is not sensitive to N and the user can choose an appropriate value depending on the available memory.

Drift Detection Threshold Similar to (Haque et al., 2017), we select $\mu = -\log \tau$ for drift detection threshold in Equation 4.12. Figure 4.14c shows the performance of our approach on different values of τ. It indicates that the proposed method is not significantly sensitive to this parameter since the accuracy and execution time overhead is similar to that shown in Figure 4.10 and Figure 4.11.

Size of $\mathcal{M}$ By default, we only use a single classifier for predicting label of target data instances. Here, we measure the performance of EnsFusion-Max when using different sizes of classifier ensemble. Both accuracy and time measurements shown in Figure 4.14d indicates that our approach is marginally sensitive to this parameter. However, we observe that the classification accuracy decreases with a larger value of L across all data sets. This may be due to a correlation between the classifier model errors which uses similar data instances due to the streaming update. Importantly, the execution time is similar across different values of L since the ensemble update time is overshadowed by density ratio estimator updates, and is practically insignificant.

CHAPTER 5

DEFENDING AGAINST ADVERSARIAL THREATS ON LEARNING[1]

In this chapter, we will discuss a strategy to defend against adversarial threats when deploying machine learning algorithms, such as MSC (see Chapter 3) and FUSION (see Chapter 4), on an untrusted third-party resource.

5.1 Securing Data Analytics

In a data stream classification problem, predicted labels of each data instance are expected to be available soon after the data is observed on the stream. Techniques discussed in Chapter 4 is key to achieving a scalable solution, in the multistream setting, with a goal of handling fast data streams. When data that occur in a stream contains security-sensitive or secret information, concerns over deployment (or where is the computation performed?) becomes an additional factor in adopting these algorithms in a real-world application. While one may not have sufficient resources to handle computations locally, various cloud services are available in which data evaluation (analytics) can be performed. Yet, security against various attacks, including insider threat, may be inadequate or unsatisfactory. In this part of the dissertation, we will focus on simultaneously addressing the two key concerns of using machine learning solutions over a security-sensitive data, i.e., providing a secure mechanism against strong adversarial threats without significantly affecting execution overhead (scalability).

As mentioned in §2.3, a commercially available hardware solution, called Intel SGX, can be used to secure computation against a strong adversary who may control the system used for execution. Concretely, a cryptographically secure computational region called an

---

[1] Reprint by permission from Springer Nature: Chandra S., Karande V., Lin Z., Khan L., Kantarcioglu M., Thuraisingham B. (2017) Securing Data Analytics on SGX with Randomization. In: Foley S., Gollmann D., Snekkenes E. (eds) Computer Security ESORICS 2017. ESORICS 2017. Lecture Notes in Computer Science, vol 10492. Springer, Cham ©2017

enclave, is created. Any computation or data accessed within an enclave is encrypted to an external adversary who may control the OS. However, there exists side-channel information leak when used in a naive manner. The occurrence of a side-channel during computation within an enclave is due to the shared resources between the trusted and untrusted regions. In general, studies in the past has utilized information from side-channels to infer execution flow of an application or to perform an attack (Chandra et al., 2014).

In the scenario considered in this chapter, the external adversary may use the side-channel information to observe the behavior of computational units such as page faults, CPU usage, memory usage, etc. during computation to infer sensitive information being protected by the enclave. Studies in the recent past have described various strategies (Ohrimenko et al., 2016; Schuster et al., 2015; Shinde et al., 2016) to defend against such adversarial inference. Typically, it is the algorithmic designer's burden to incorporate such execution strategies when using SGX. Unfortunately, these incur large computational overhead in certain types of data analytics. Particularly when the machine learning algorithm uses a tree-like data structure (e.g., Decision tree, Naive Bayes, Graphical Models (Haque et al., 2014)). When using such strategies in data classification algorithms using, such as a multistream setting, the computational speed-up gained by scalable solutions is offset by the computations involved in such strategies. In this chapter, we develop an alternative defense strategy that has a significantly lower computational overhead.

5.1.1 Threat Model

Analytics on data containing sensitive information is performed on a third-party untrusted server with Intel SGX support. While data-owners have no control over this server, they may establish a cryptographically secure connection to an enclave in the server. Similar to (Lee et al., 2016), we assume that an attacker controls the untrusted server, and has the ability to interrupt the enclave as desired, by modifying the OS and SGX SDK, to obtain

Table 5.1: List of symbols for studying RAND defense.

Symbols	**Description**
k	Number of Clusters
T	Set of Clusters
L	Number of Dummy Data

Table 5.2: List of public and confidential parameters. Here, `Tree` indicates model structure, and P indicates probability function.

Model	Parameters	
	Public	Confidential
Decision Tree	n, v, K	x, y, `Tree`
Naive Bayes	n, v, K	$x, y, P(y\|x), P(y)$
K-Means	n, v, K, k, I	x, y, T

side-channel information from page or cache accesses, page faults, and log files. Nonetheless, code and data within the enclave cannot be modified, except by the data-owner.

The primary goal of an attack is to obtain sensitive information leaked through side-channels from a benign machine learning application running within the SGX enclave. Sensitive information may include model parameters, feature values of input data, and data distribution statistics. For example, the structure of a decision tree (denoted by `Tree`) may be revealed if nodes in the tree are present on different pages, while the attacker tracks the order of execution during evaluation. Similarly, the proportion of each cluster (denoted by T) in the k-means clustering algorithm may reveal sensitive data patterns. We term this set of sensitive attributes as *confidential*. A defense mechanism aims to prevent the attacker from inferring confidential attributes through side-channel information. Nevertheless, each learning algorithm has parameters which are data invariant. For example, height of a decision tree (H), number of features in each data instance (v), domain and range of feature values (f), number of class labels (K), number of clusters in k-means clustering (k), and number of iterations for learning (I), remains constant for a given dataset. These parameters can be easily inferred from analyzing algorithmic execution. We assume that the code for each algorithm is publicly available, along with its data invariant parameters. Table 5.2 lists

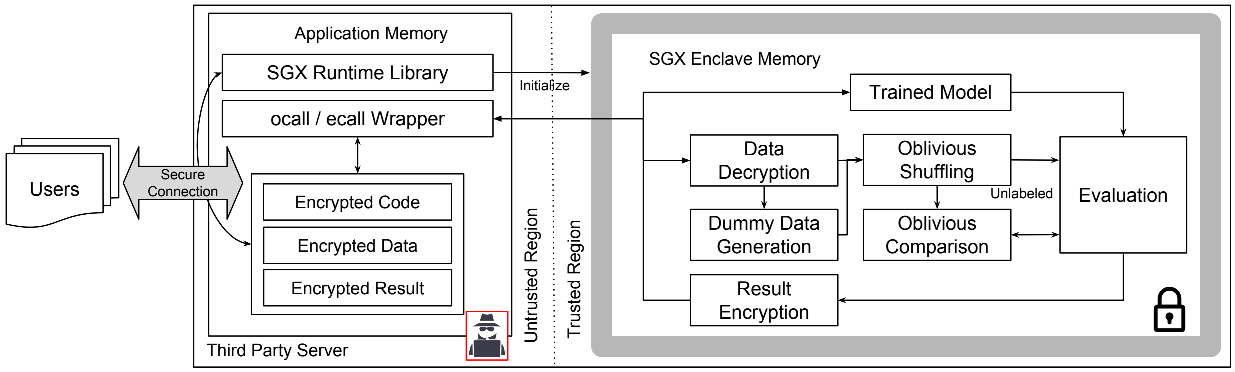


Figure 5.1: Overview of Data Analytics on SGX using randomization.

the associated confidential and public parameters for each algorithm considered. We use the same symbols as Table 3.1. Since we use specific algorithms to study the defense strategy proposed in this chapter, Table 5.1 lists associated additional symbols.

5.1.2 Overview

Figure 5.1 illustrates the overall defense methodology we present in this chapter. A user provides cryptographically secure encrypted data (containing sensitive information) to a third-party untrusted server, along with a pre-trained model. An enclave is established, and the pre-trained model initialized. By requesting a set of data instances into the enclave from application memory through an `ocall`, we decrypt these instances and empirically evaluate the domain and range statistics of each feature. Since we desire that computation involving dummy data instances produce access patterns similar to that of user-given data instances, we generate v feature values uniformly at random within its empirical range to create a dummy instance. After generating L such instances, we shuffle them with user-given data instances in a data-oblivious manner and evaluate each instance in the shuffled dataset sequentially using the pre-trained model that is fully encapsulated within the enclave. By obliviously ignoring results associated with dummy data instances, we obtain the results for user-given data instances. We then encrypt these results in a cryptographically secure manner and save it in the untrusted application memory via an `ocall`. Here, data-oblivious shuffling of dummy and user-given instances is crucial since it introduces uncertainty in access patterns observed from side-channels by the attacker.

The crux of the above solution is in the way we generate dummy data instances and use data-oblivious mechanisms for shuffling and ignoring results of computation associated with dummy data instances. If we only employ the shuffled (contaminated) dataset for evaluation in a naive implementation of a data analytics algorithm, i.e., by ignoring results from dummy instances, it may not be possible to conceal all sensitive model parameters and data patterns. Each learning algorithm has an inductive bias, different from one another, which prevents the universal application of a naive strategy by itself. For example, the inductive bias of a decision tree is that data can be divided in the form of a tree structure. Whereas, the bias in k-means clustering assumes that instances having similar properties are closer to each other than those with dissimilar properties. In both these cases, the structural representation of data is different and is input-dependent. We address this challenge by utilizing dummy data instances to conceal model structure and parameters as well. This indicates that computation involving dummy data instances need to be tracked, but in a data-oblivious manner so that uncertainty in resource access trace observed by the attacker is preserved. We first introduce the primitives of our defense strategy, i.e., dummy data generation and data-oblivious comparison, in §5.1.3, and describe data analytics algorithms that utilize them for defense, in §5.1.4.

5.1.3 Primitives

Dummy Data Generation.

Algorithm 6 illustrates our dummy data generation process. Using public parameters of user-given dataset D, we choose a random number uniformly within the range of each feature (i.e., values between `MIN` and `MAX`) in D. This choice limits the bias of dummy data instances

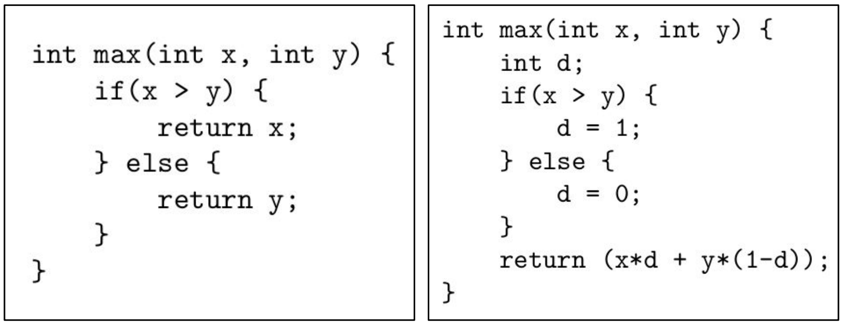
Algorithm 6: A primitive for generating dummy data instances.

```
Input: $D$: Dataset, $n$: Dataset Size, $d$: No. Features
Result: $\hat{D}$: Shuffled Data Instances
begin
    $\text{MAX}, \text{MIN} = \text{get\_range}(D, n, d)$
    $\hat{D} = D$
    while $|\hat{D}| < (n + L)$ do
        $v = \text{array}(d)$    // Initialization
        for $i \in \{0, d\}$ do
            $v[i] = \text{random}(\text{MAX}_i, \text{MIN}_i)$
        $\hat{D} \leftarrow v$
    return oblivious_shuffle($\hat{D}$)
```

and prevents them from having distinguishing characteristics compared to user-given data instances. If not, an attacker may be able to identify such characteristics and discard access traces associated with dummy data instances, thereby defeating our defense mechanism. We generate L dummy data instances and initially append them to the set of user-given data instances, forming $\hat{D}$. We then shuffle $\hat{D}$ in an oblivious manner and sequentially process each data instance from the shuffled dataset during evaluation. One corner case is when `MIN` = `MAX`. With the goal of increasing variance of each feature in $\hat{D}$, we add an appropriate margin to `MAX` such that `MIN` < `MAX` is always true. In §5.2, we present the implementation details of oblivious data shuffling.

Data-Oblivious Comparison.

We use a data-oblivious comparison primitive for checking whether a data instance is a dummy or not. Typically, we first compute using a data instance and then decide whether to ignore or retain the result of such computation depending on the type of data instance involved. We only desire to ignore results involving dummy data instances in a data-oblivious fashion. This ensures that the attacker observes resource access traces from both user-given and dummy data instances, which are indistinguishable.



```
int max(int x, int y) {
    if(x > y) {
        return x;
    } else {
        return y;
    }
}
```

```
int max(int x, int y) {
    int d;
    if(x > y) {
        d = 1;
    } else {
        d = 0;
    }
    return (x*d + y*(1-d));
}
```

a) Non-oblivious max

b) Oblivious max

Figure 5.2: Illustration of data-oblivious comparison.

Figure 5.2 illustrates the difference between non-oblivious and oblivious `max` function as an example of comparison primitive. Figure 5.2b is oblivious at the element-level since both conditional branch statements access the same set of variables. Whereas, Figure 5.2a is non-oblivious since either x or y is accessed when the `max` function returns depending on the conditional statement executed. In the case of an array, we access all elements in the array sequentially to remain data-oblivious. The mechanism proposed in (Ohrimenko et al., 2016) uses a more efficient compiler-based approach to perform oblivious comparison and array access at cache-level granularity instead of element-level granularity. We leave its adaptation to our proposed approach for future work.

5.1.4 Learning Algorithms

Decision Tree Classifier.

It is a tree-based model that uses an information-theoretic measure for data classification. In training a popular variant called ID3 (Bishop, 2006), a feature with the largest information gain, with respect to the class label, is selected for partitioning the dataset into disjoint subsets. By iteratively performing this data partitioning on each residual data subset, a tree structure is created. Each feature value used for partitioning (or rule) then becomes either the root or an internal node of this tree. A leaf is formed when further partitioning

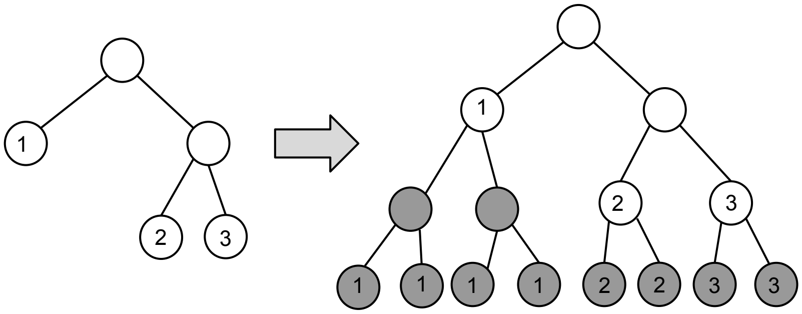


Figure 5.3: Creating an obfuscated decision tree. Shaded nodes are formed using dummy data while others are formed using user-given data. Labels (denoted by $\{1, 2, 3\}$) of the original tree's leaf node is replicated in its descendant leaf nodes of the obfuscated tree.

is discontinued or unnecessary, i.e., when either all features are used in a path from the root, all data instances within the residual data subset has the same class label, or a user-defined maximum tree height is achieved. The last stopping condition is typically used to reduce overfitting (Bishop, 2006). During the evaluation, class label of a test data instance is predicted as the majority label at a leaf that is encountered by following tree branches, starting from the root, according to its feature value consistent with the associated rule of intermediate tree nodes.

When a naive implementation of the above algorithm is employed within an SGX enclave, the attacker may track data-dependent tree node accesses during evaluation. This reveals the tree structure as well as the path of each test data instance. A typical strategy to defend against this side-channel inference-based attack is to balance the tree by adding dummy nodes and access all nodes during evaluation of each test instance. As mentioned in (Ohrimenko et al., 2016), such a strategy has a runtime complexity of $\mathcal{O}(n\alpha)$ during evaluation, where α is the number of tree nodes. However, the complexity in a naive implementation is $\mathcal{O}(n \log \alpha)$. Clearly, data-obliviousness is achieved at the cost of computational efficiency, especially when α is large.

Instead, we utilize the dummy data generation primitive to obtain a contaminated dataset and use the naive evaluation algorithm for class label prediction. During training, we learn a decision tree using user-given training data instances (with known class labels) and create a balanced tree using dummy data instances, offline. Figure 5.3 illustrates an example of a balanced decision tree. Here, a tree (we term as *original*) resulting from user-given training data instances is obfuscated with nodes created from dummy data instances to obtain a balanced tree. Leaf nodes in the obfuscated tree reflect the class label of its ancestor node that forms a leaf in the original tree. Clearly, the predicted class label of a test data instance on the obfuscated tree is the same as the original decision tree. Since dummy data instances are obliviously shuffled with user-given test data instances, access traces obtained by the attacker for dummy data instances are indistinguishable from that of user-given test instances. Therefore, the true data access path is hidden in the overall noisy access path obtained by the attacker. With L dummy data instances in the contaminated dataset, the time complexity of evaluating n user-given test data instances is $\mathcal{O}((n + L) \log \alpha)$.

Naive Bayes Classifier.

It is a Bayesian model trained with an assumption of feature independence, given class labels (Bishop, 2006). Similar to the decision tree model, we train a Naive Bayes classifier offline with a user-given training dataset and evaluate test data instances online, i.e., within an SGX enclave. During the evaluation, the predicted label of a test data instance is a class with the largest conditional probability, given its feature values. Such a classifier is typically used in the field of text classification that has a large number of discrete-valued features. The product of class conditional probability is computed for each feature value of user-given test data instance. Naively, one can pre-compute conditional probability for each feature value during training and access appropriate values during evaluation. In this case, an attacker may infer class and feature proportions of a given test dataset by tracking access sequence of pre-computed values. In a purely data-oblivious defense strategy, every element in the pre-computed array is accessed for evaluating each test data instance. If each of the d features have a discrete range of size f, computational time overhead for evaluation is $n \times d \times f$, whereas that of the original naive evaluation is $n \times d$. Clearly, this is a bottleneck in execution time when the range f is large. Instead, we utilize our dummy data generation primitive during evaluation by employing the naive method for accessing pre-computed array elements, inducing access patterns that are alike for both user-given and dummy data instances. The overhead in computational time for our modified version of Naive Bayes is $(n + L) \times d$. If $L \ll f$, our proposed defense is more efficient than the pure data-oblivious solution.

K-Means Clustering.

The goal of k-means clustering is to group data instances into k disjoint clusters, where each cluster has a d-dimensional centroid whose value is the mean of all data instances associated with that cluster. Clusters are built in an iterative fashion. We follow a streaming version of Lloyd's method (Bishop, 2006) for constructing clusters and evaluating user-given test data instances since they are suitable for handling large datasets. During training, k cluster centroids are created by iteratively evaluating its value with least mean squared Euclidean distance, and re-evaluating cluster association of user-given data instances using the computed centroid. Evaluation is performed online, i.e., within an SGX enclave. The user provides learned centroid and a set of test data instances. While cluster association of each data instance is evaluated by computing the minimum Euclidean distance to centroids, we re-compute the centroid of its associated cluster using the test data instances.

In a naive implementation of k-means clustering, the attacker can infer sensitive information, such as cluster associated with each data instance by tracking the centroid being accessed during assignment, and cluster proportions during centroid re-computation. The pure data-oblivious solution addresses this problem by performing dummy access to each centroid. On the contrary, we utilize the dummy data generation primitive to perform cluster assignment of both dummy and user-given data instances in an oblivious manner and use the unmodified naive cluster re-computation method. This adds noise to cluster proportions inferred by the attacker. Since the number of clusters is fixed and is typically small, the time complexity remains the same as the original algorithm (Ohrimenko et al., 2016).

5.2 Implementation

One possible attack on the proposed defense strategy is to collect access traces of identical test data instances during evaluation, and use a statistical method to identify execution pattern of user-given test data instances in them. The main idea is that though these traces will be poisoned with execution involving random dummy data instances, execution of identical test data instances remain same. An attacker may produce such identical test instances by capturing an encrypted user-given instance at the application side, and providing identical copies of this data as input to the enclave application. We use a simple technique for discouraging this replay-based statistical attack by associating each data instance with a unique ID (called *nonce*), whose value is generated from a sequential counter. When data instances are passed to the enclave in response to an `ocall`, we check for data freshness within the enclave by comparing the internal nonce state to the nonce of each input. We proceed with evaluation if each new nonce value is greater than the previous one, else we halt execution. Since an attacker cannot change the nonce value of an encrypted data instance, this can detect stale instances used for a replay attack. We are aware that there exists superior methods for generating dummy data instances to thwart replay-based attacks in related domains (Li et al., 2007), and leave its exploration for future work.

An important technique for reducing the effectiveness of inferring sensitive information from side-channels is the random shuffling of dummy data with user-given data instances in a data-oblivious manner. For simplicity, we assume that domain of each feature in the dataset is either discrete or continuous real-valued numbers. Nominal features are converted into binary vector using one-hot encoding (Murphy, 2012). Data shuffling is performed as follows. For brevity, we call the array containing data instances within the enclave as *data-array*. We associate a random number to each element of the data-array. Initially, dummy data instances are appended to the data-array as soon as they are created. We utilize `sgx_read_rand` for random number generation. We then shuffle this array using an oblivious sorting mechanism over these random numbers. Similar to (Ohrimenko et al., 2016), we implement the Batcher's odd-even sorting network (Batcher, 1968) for data-oblivious sorting, utilizing data-oblivious comparison during data swap when necessary. The runtime of this sorting method is $\mathcal{O}((n+L)(\log(n+L))^2)$. There are other shuffling algorithms with more efficient runtime complexity. We leave its applicability for future work. Meanwhile, we use a Boolean array, of size equal to the data-array, where value of each element indicates whether the corresponding instance in data-array is dummy or otherwise. Using oblivious comparison primitive, we identify and ignore computational results involving dummy data instances while sequentially evaluating the shuffled dataset.

5.3 Evaluation

Next, we analyze privacy guarantee of our proposed method and empirically evaluate computational overhead on various datasets.

5.3.1 Quantification of Privacy Guarantee

In our attack model, the attacker obtains execution traces in terms of sequential resource access while performing data analytics with user-given data instances. An attack on data privacy is successful when the attacker infers sensitive information from these traces by identifying distinguishing characteristics. However, the attack is unsuccessful if such distinguishing characteristics are either eliminated or significantly reduced via a defense mechanism. Such defenses are effective when they can provide quantifiable guarantees on data privacy. The primary question is how to measure privacy? Authors in (Ohrimenko et al., 2016) measure

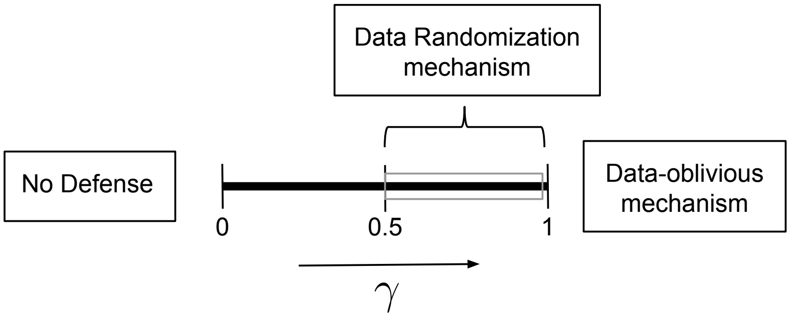


Figure 5.4: Measuring privacy guarantee of SGX defense mechanisms.

data privacy in terms of indistinguishability of a trace against a randomly simulated one. Since our defense mechanism primarily consists of performing non-essential or fake resource accesses, we define this indistinguishability in terms of *trace-variants* that is possible in a data analytics model. A trace-variant can be viewed as a sequence of page (or cache line) access when evaluating a test data instance. If N is the total number of trace-variants observed by an attacker from the model, we compute *Privacy-Guarantee* (denoted by γ) as the ratio of fake trace-variants to the total number of observed trace-variants. The value of N may depend on the variance in data and model. From a defense strategy perspective, every new data instance can provide a different access sequence at best. In this case, $N = n$ where n is the user-given dataset size. The following analysis assumes this case for simplicity, including the defense against replay attack mentioned in §5.2.

In a purely data-oblivious solution (Ohrimenko et al., 2016), there are $N - 1$ fake trace-variants during evaluation since all possible cache-lines are accessed so that access pattern is the same for all data instances. For example, all nodes in a decision tree are accessed for evaluating the class label of each data instance. Here, each node may reside on a different cache-line or page. Therefore, $\gamma = \frac{N-1}{N}$. Note that $\gamma \simeq 1$ with large N; privacy is guaranteed on large N when this defense mechanism is applied. On the other end of the privacy-guarantee spectrum, $\gamma = 0$ when no defense is applied, i.e., no fake trace-variants are possible.

At this extreme, no privacy is guaranteed to the user's data. Figure 5.4 illustrates this privacy-guarantee spectrum.

Our proposed solution provides asymptotic privacy guarantee in terms of the number of dummy data instances used. Since L dummy data instances are generated, there are at most L fake trace-variants with $N + L$ observed trace-variants. Therefore, the associated privacy-guarantee is $\gamma = \frac{L}{N+L}$. Clearly, a larger value of L provides greater privacy guarantee; it tends towards the γ value of purely data-oblivious solution (i.e., $\gamma \simeq 1$) for large L. If $L < N$, then an attacker can simply guess each trace to be true and infer sensitive information with a higher probability than random. Therefore, we choose $L \geq N$ to limit probability of a correct guess by the adversary to $\frac{1}{2}$ at best (as shown in Figure 5.4), similar to (Ohrimenko et al., 2016). We now empirically demonstrate our proposed technique and showcase the trade-off between privacy guarantee and computational efficiency with different choices of L.

5.3.2 Datasets

We measure execution time overhead of the proposed defense strategy using 3 publicly available real-world datasets (Repository, 1998) and a synthetic dataset. Table 5.3 lists these popular datasets with corresponding data statistics. The *Arrhythmia* dataset consists of medical patient records with confidential attributes and ECG measures. The problem is to predict the ECG class of a given patient record. The *Defaulter* dataset consists of financial records containing sensitive information regarding clients of a risk management company. The problem is to predict whether a client (i.e., a data instance) will default or not. While the above datasets contain security-sensitive information that need protection, we also use *ForestCover* and *Synthetic* as benchmark datasets, similar to dataset mentioned in Chapters 3 and 4.

These datasets may contain continuous and discrete-valued features. For simplicity of implementation, we evaluate the decision tree and Naive Bayes classifiers using a quantized

Table 5.3: Dataset statistics and empirical time overhead with $L = n$.

Dataset	Statistics			Time Overhead					
	Size (n)	Features (d)	Classes (C)	Decision Tree		Naive Bayes		K-Means	
				SGX +Obliv	SGX +Rand	SGX +Obliv	SGX +Rand	SGX +Obliv	SGX +Rand
Arrhythmia (A)	452	280	13	52.49	**9.37**	319.15	**6.11**	**4.16**	6.36
Defaulter (D)	30,000	24	2	4.13	**1.11**	1.56	**1.10**	**1.07**	1.17
ForestCover (F)	50,000	55	7	2.72	**1.09**	3.13	**1.08**	**1.05**	1.07
Synthetic (S)	50,000	71	7	2.53	**1.09**	3.47	**1.07**	1.22	**1.09**

version of each dataset. We divide each feature range into discrete bins of equal width. For decision tree, we use $f = 10$ bins. However, for Naive Bayes, we use $f = 1000$ bins to reflect the dimensionality mentioned in §5.1.4. Nevertheless, we use the original form of each dataset to evaluate the k-means clustering algorithm.

5.3.3 Results and Discussion

The goal of empirical evaluation is to study and demonstrate the applicability of our defense strategy in various settings. We implement a pure data-oblivious strategy, similar to (Ohrimenko et al., 2016), using data-oblivious comparison and array access over the naive implementation of each data analytics algorithm. This baseline defense strategy is denoted by *Obliv*, whereas our proposed implementation is denoted by *Rand*. For each modified data analytics algorithm (i.e., Obliv and Rand), the computational *time overhead* is measured as the ratio of time taken by the modified algorithm executed within an SGX enclave to that of a naive implementation executed without SGX support. We perform all experiments on an SGX-enabled 8-core i7-6700 (Skylake) processor operating at 3.4GHz, running Ubuntu 14.04 system with a 64GB RAM.

Table 5.3 lists the time overhead measured on each dataset for decision tree and Naive Bayes classifiers, as well as k-means clustering, averaged over 5 independent runs. Note that we denote the defense strategies with *SGX+x*, where $x = \{\text{Obliv}, \text{Rand}\}$, to emphasize that they are executed within an SGX enclave. Since SGX currently supports limited enclave memory, we evaluate in a streaming fashion by dividing the dataset into small disjoint sets or chunks. Evaluation is performed over each chunk of size 64, over the given pre-trained model.

From the table, Rand clearly performs significantly better than Obliv in the case of the decision tree and Naive Bayes classifiers. For example, Rand has only 11% overhead when class labels are evaluated using a decision tree in 16.76s, compared to Obliv that takes 62.02s, on the Defaulter dataset. When executing without any defense within the SGX enclave, it took 16.13s. This shows that overhead due to enclave operations is small, as expected (Karande et al., 2017). A higher overhead is observed in the Arrhythmia dataset due to smaller dataset size. For example, the naive implementation of decision tree on this dataset takes 0.01s, compared to 0.79s in Obliv, and 0.14s in Rand. Also, it took 0.08s on the implementation within SGX enclave, but without employing any defense strategy. As another example, the execution time of Naive Bayes classifier on ForestCover dataset with Rand took 51.63s, compared to 149.48s with Obliv, and 50.16s without any defense within the SGX enclave. Clearly, the cost of dummy data operations in Rand can be observed in the larger execution time compared to the naive implementation, yet it is much lower than Obliv.

Limitations.

For both decision tree and Naive Bayes classifiers, the number of fake resource access in Obliv is greater than that of Rand. Evaluating every test data instances in Obliv accesses each branch in a decision tree, and each of the $d \times 1000$ elements in the pre-computed probability array of Naive Bayes. Meanwhile, corresponding resource access in Rand is significantly small. However, when resource access patterns in both Obliv and Rand is similar during evaluation, the compromise on privacy with little or no trade-off in computational time of Rand is not very enticing. Time overhead is shown in Table 5.3 for k-means clustering algorithm indicates one such example. Here, every cluster has to be accessed when searching for the nearest centroid to a given test data instance. While in Obliv, centroid re-computation of cluster assignment may be performed for each cluster, the time taken for the oblivious shuffling of $n + L$ elements in Rand seems to surpass this re-computation time overhead. Except for the Synthetic dataset, Obliv outperforms Rand in all other datasets. In this situation, it is better to use Obliv defense strategy that guarantees better data privacy than the Rand strategy which provides a sub-optimal privacy guarantee.

Cost of More Privacy.

The above results for Rand use an equal number of dummy and user-given data instances, i.e., $L = n$. If L is increased to provide better privacy according to §5.3.1, the cost of oblivious data shuffling, in terms of execution time, increases since $n + L$ data instances are to be shuffled. Figure 5.5 illustrates this increase in time overhead when using a decision tree classifier with Rand defense on various datasets as an example. This indicates that the value of L can be chosen appropriately by a programmer with a desirable trade-off between computational overhead and data privacy. For example, a larger value of L for higher γ may be appropriate when the model has larger search space, similar to the Naive Bayes classifier discussed in this paper. In such cases, a higher value of γ reduces the likelihood of dummy data instances producing unique patterns, with respect to user-given data instances.

5.3.4 Security Evaluation

The goal of our security evaluation is to empirically address the two main questions regarding Rand's data privacy guarantee; 1) Are access traces observed by the attacker randomized?, and 2) Are traces obtained from evaluating user-given and dummy data instances indistinguishable? Using Pin Tool (Luk et al., 2005), we generate memory access traces (sequence

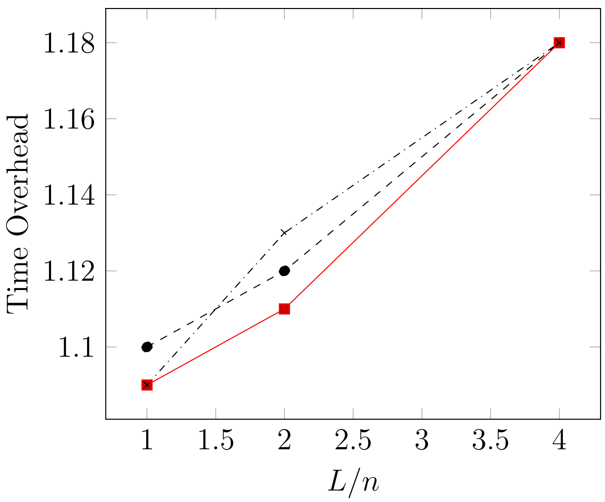
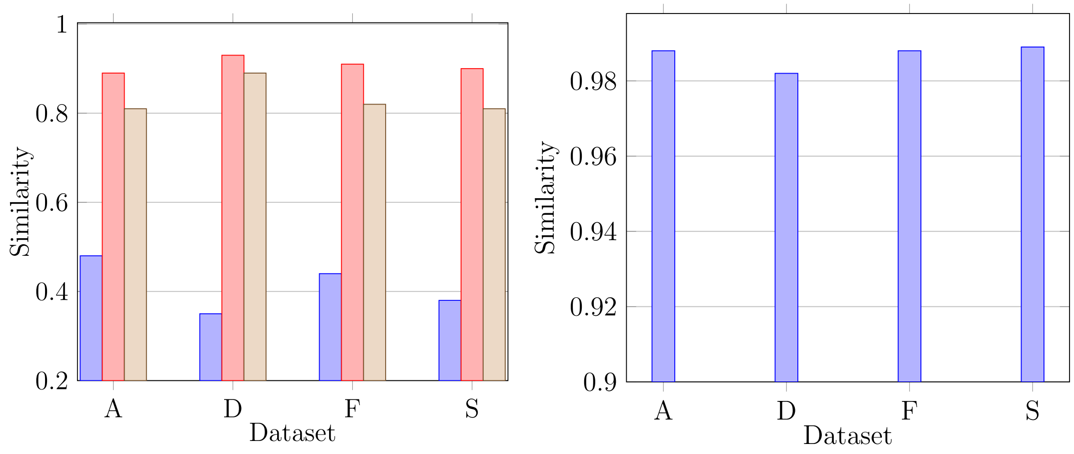


Figure 5.5: Time overhead with increasing L (in proportion of n) on decision tree classifier with Rand, on -●- D, —■— F and -·×·- S datasets.



(a) Across similarity.

(b) Within similarity.

Figure 5.6: (a) Shows similarity scores between access traces across different sets of instances when evaluated on the same classifier. Here, comparison between different defenses are shown, i.e., 🟦 Rand, 🟥 Obliv, and no defense (🟫). (b) Shows similarity between traces of user-given and dummy data instances within a set of instances evaluated on 🟦 Rand.

of read and write) of each classifier implementation when executing it in the SGX simulation mode. Here, we create 5 disjoint sets of 16 randomly chosen data instances for each dataset.

To answer the first question, we obtain traces by independently evaluating the 5 sets of data instance on a classifier, for each dataset. We perform different experiments on classifier implemented with no defenses (naive), Obliv, and Rand, for comparison. We then compute Levenshtein similarity (Navarro, 2001), as a surrogate to measure noise addition, between traces from the 5 sets on each dataset. Here, more similarity implies less randomization (i.e., added noise). Figure 5.6a shows an example result on trace comparisons obtained by evaluating a decision tree with corresponding defenses. In the figure, we can observe that traces from Obliv are more similar to each other (across the 5 sets) than those from the naive implementation, as mentioned in (Ohrimenko et al., 2016). For example, in the Arrhythmia dataset, we obtain a similarity measure of 0.89 for Obliv, compared to 0.81 for naive. However, traces from Rand are more dissimilar to each other compared to Obliv and naive approaches, indicating more data variance and randomization. On the contrary, we address the second question by comparing traces within a single set of 16 data instances. Concretely, we compute Levenshtein similarity between traces obtained by evaluating user-given data instances only, and those of dummy data instances only, in each set. Figure 5.6b illustrates an example on decision tree classifier with Rand. The high similarity scores between traces corresponding to the two types of data instances indicate indistinguishability.

5.4 Related Works

Studies on applications using Intel SGX have focused on an untrusted cloud computing environment. The first study in this direction (Baumann et al., 2015) executed a complete application binary within an enclave. However, using this method on applications requiring large memory caused excessive page-faults that revealed critical information (Sinha et al., 2015), thereby violating data privacy. To address this challenge, a recent study (Schuster et al., 2015) used Hadoop as an application to split its interacting components between SGX trusted and untrusted regions. The main idea was to reduce TCB memory usage within the enclave for decreasing page faults. Challenges in executing data analytics within an SGX enclave was first recently described by Ohrimenko et al. (Ohrimenko et al., 2016). They propose a pure data-oblivious solution to guarantee privacy at cache-line granularity. We have compared our approach with a similar defense strategy. Alternative to algorithmic solutions, studies have proposed mechanisms to detect and prevent page faults attacks via malicious OS verification (Fu et al., 2017) and transactional synchronization (Shih et al., 2017).

A large group of studies in privacy-preserving mechanisms deals with designing algorithms to preserve data privacy before data is shared with an untrusted environment (Aggarwal and Philip, 2008). Particularly, these studies focus on problems where identification of individual records is undesirable. Typically, the data is modified by the addition of noise to features, regularization conditions, use of anonymization (Brickell and Shmatikov, 2008), and randomization (Kargupta et al., 2003) techniques. Instead, we focus on using a trusted hardware environment to protect privacy by using cryptographic methods to maintain confidentiality and trustworthiness (Bauman and Lin, 2016). We randomize side-channel information rather than user data for preserving privacy.

Use of dataset contamination to defend against adversaries is not new in machine learning settings. Studies on anomaly detection and intrusion detection (Huang et al., 2011) have discussed various types of attacks and defenses with regard to poisoning a user-given dataset with random data (Barreno et al., 2006). Particularly, a process called *Disinformation* is used to alter data seen by an adversary as a form of defense. This corrupts the parameters of a learner by altering decision boundaries in data classification. The process of *randomization* is used to change model parameters to prevent an adversary from inferring the real parameter values. These methodologies, however, limit the influence of user-given data in the learning process and may affect model performance on prediction with future unseen data instances. In all these cases, the adversary does not have control over the execution environment and is weak. We instead leverage the effect of randomization to defend against side-channel attacks from a powerful adversary while performing data analytics on an Intel SGX enabled processor.

CHAPTER 6

CONCLUSION

In this final chapter of the dissertation, we briefly outline our key contributions and provide future research directions to investigate.

6.1 Summary

This dissertation concerns with providing scalable solutions to learning tasks under the complementary lens of machine learning and security. Specifically focusing on the task of label prediction, we first explore a novel composition of problems in a data stream setting that frequently occur in the wild. We then explore security problems in deploying solutions to the data stream classification problems in an untrusted environment. The main contributions resulting from this investigation is as follows.

- In exploring the problem of label prediction over a concept-drifting data stream in which availability of labeled data is limited, we presented a framework (called Multi-Stream Classification or *MSC*) to address the challenges of bias correction and drift detection over time, in Chapter 3. This provided a platform to leverage available labels having a biased distribution compared to the population for predicting class labels of unlabeled instances occurring along the stream, and maintaining predictive performance over long periods of time. We studied its theoretical properties and empirically demonstrated its ability to classify over benchmark datasets.

- We then investigated the use of various bias correction mechanisms within the MSC framework. Particularly, we focused on two popular mechanisms, i.e., Kernel Mean Matching (KMM) and Kullback-Liebler Importance Estimation Procedure (KLIEP). Our investigation showed that both these methods are not scalable for naive use over a fast data stream. In Chapter 4, we propose solutions to address this issue.

  - We proposed a sampling-based bootstrap mechanism over KMM, called *VFKMM*, and analyzed its theoretical properties. Our results demonstrate a clear advantage over competing methods.
  - Unlike KMM, we proposed an ensemble-based online approach over KLIEP to perform label prediction in the multistream setting, called *EnsFusion*. Our theoretical and empirical results demonstrated its scalable property.

- We then focused on addressing the concerns of adversarial threats when deploying a machine learning system in an untrusted third-party resource, in Chapter 5. Here, we explored a strategy to overcome adversarial threats through side-channels leaks when utilizing a cryptographically secure commodity-hardware solution called Intel SGX. Our thorough evaluation demonstrated that by using noise cleverly during computation, a significant reduction in computational overhead can be achieved with a small discount on privacy.

6.2 Future Research Direction

While the above contributions provide new insights into the multistream setting and associated security concerns, there exists a large scope for future research.

6.2.1 Classification with Limited Supervision

Over the past decade, studies on active learning (Fan et al., 2004; Žliobaitė et al., 2014) and other semi-supervised learning (Dyer et al., 2014; Haque et al., 2015) have been addressing the classification problem under limited supervision in various settings. Unlike these approaches, the multistream setting aims to address the drift detection, correction, and label prediction problems simultaneously. In this dissertation, we primarily focus on bias correction methods where the source and target instances occur from the same domain. However, other domain adaptation setting can be possible in the multistream setting, i.e., source and target instances occur from different but related domains. Though existing techniques may complement our framework, a naive application of such techniques in the multistream setting may not be possible due to the concept-drifting nature of data streams. We leave its exploration for future work.

Interestingly, a parallel approach to lack of labels is to synthetically generate labels from various labeling function and amortize the noise among them (Ratner et al., 2017). This approach can be used in the multistream setting as well where the labeling function can be seen as a biased source. Though we only explore a single source and target stream, multiple labeling functions may create independent sources of data. We leave this exploration for future as well.

6.2.2 Adversarial attacks on data classification

In Chapter 5, we focused on adversarial threats to a black-box model where the adversary is external to the application. However, the adversary may also be a user of the application. In such cases, recent studies (Huang et al., 2011; Hitaj et al., 2017; Grosse et al., 2017) have shown effective techniques to perform an adversarial attack on machine learning models. Essentially, these techniques generate adversarial examples with minimal data perturbation with a goal of forcing the trained classifier to misclassify. The motivation of an adversary, in this case, is to evade the classifier without being detected by a defense mechanism (Xu et al., 2016). Examples of such adversarial attacks include evasion of intrusion detection, evade malware detectors, or trick image classifiers used in applications such as self-driving cars to classify road signs incorrectly. The consequences of such attacks are devastating to the naive users of applications employing these fragile classifiers. Studies have also developed specific defense techniques such as robust training (Carlini and Wagner, 2017), ensemble models (Liu et al., 2016), and model smoothening (Papernot et al., 2016). Yet, such defenses are not effective since the attacker can always create adversarial examples that break specific defense strategies. Moreover, it has been shown that adversarial examples have a transferability property (Papernot et al., 2016), i.e., the generated adversarial examples can utilize any classifier to successfully attack a different classifier. This creates a black-box attacks scenario (Papernot et al., 2016) where hiding the model and computation from the adversary is insufficient as a defense mechanism. Moreover, it has been shown that the black-box attack scenario can reveal training data in a machine-learning-as-a-service model (Shokri et al., 2017).

The success of such attacks demonstrates the fragility of classification output from classifiers, particularly when directly employed in mission-critical applications. Therefore, it is important to understand the effect of such attacks on the multistream setting, which is currently unknown. Particularly, the attacker may inject samples into the target stream for the purpose of inducing a distribution change. This may dramatically affect the classifier output for all target instances, i.e., including non-adversarial instances as a reaction to change affects the classifier ensemble. We leave the investigation of defenses against such attacks for the future.

6.2.3 Applications

In the dissertation, we have claimed that the multistream setting can be observed easily in the real world. One such example is of location estimation in a political news article. Particularly, a news article detailing an atrocity committed at a physical location may contain references to multiple locations. When analyzing these articles, one key element is to identify the location at which the atrocity was committed. This is called *focus location*. In this scenario, a stream of online news articles may be considered. Here, there is a need for labeled data to train a supervised classifier. Unfortunately, such labels are sparsely available or require large manual efforts. Recently, we considered one such scenario where labeled data may occur only from a single news agency, and the unlabeled data occurs from other agencies (Imani et al., 2017). It may be a case that the writing style of labeled articles is different than other articles. We applied a bias correction mechanism in a batch-wise fashion to observe superior results on various datasets. We leave its applicability to the multistream setting for the future. A similar application can be applied for location estimation of Twitter users (Chandra et al., 2011), where true user locations are given only by a small number of users. This can form a source stream while the rest can form the target stream.

Apart from location estimation, the lack of labeled data over data stream can be seen in security applications such as website fingerprinting (Al-Naami et al., 2016) and malware detection. In website fingerprinting, the task is to determine the websites accessed by a user using only encrypted network traffic from an anonymized network. Here, traffic patterns of a few websites may be known, but others unknown. Whereas in malware detection (Masud et al., 2008), sufficient DLL's may not be available for feature extraction during the training phase. We leave this exploration for future work.

REFERENCES

Aggarwal, C. C. and S. Y. Philip (2008). A general survey of privacy-preserving data mining models and algorithms. In *Privacy-preserving data mining*, pp. 11–52. Springer.

Al-Naami, K., S. Chandra, A. Mustafa, L. Khan, Z. Lin, K. Hamlen, and B. Thuraisingham (2016). Adaptive encrypted traffic fingerprinting with bi-directional dependence. In *Proceedings of the 32nd Annual Conference on Computer Security Applications*, pp. 177–188. ACM.

Anati, I., S. Gueron, S. Johnson, and V. Scarlata (2013). Innovative technology for cpu based attestation and sealing. In *Proceedings of the 2nd international workshop on hardware and architectural support for security and privacy*, Volume 13.

Barreno, M., B. Nelson, R. Sears, A. D. Joseph, and J. D. Tygar (2006). Can machine learning be secure? In *Proceedings of the 2006 ACM Symposium on Information, computer and communications security*, pp. 16–25. ACM.

Batcher, K. E. (1968). Sorting networks and their applications. In *Proceedings of the April 30–May 2, 1968, spring joint computer conference*, pp. 307–314. ACM.

Bauman, E. and Z. Lin (2016, December). A case for protecting computer games with sgx. In *Proceedings of the 1st Workshop on System Software for Trusted Execution (SysTEX'16)*, Trento, Italy.

Baumann, A., M. Peinado, and G. Hunt (2015). Shielding applications from an untrusted cloud with haven. *ACM Transactions on Computer Systems (TOCS) 33*(3), 8.

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

Bickel, P. J. and A. Sakov (2008). On the choice of m in the m out of n bootstrap and confidence bounds for extrema. *Statistica Sinica*, 967–985.

Bifet, A. and R. Gavalda (2007). Learning from time-changing data with adaptive windowing. In *Proceedings of the 2007 SIAM International Conference on Data Mining*, pp. 443–448. SIAM.

Bifet, A. and R. Gavaldà (2009). Adaptive learning from evolving data streams. In *International Symposium on Intelligent Data Analysis*, pp. 249–260. Springer.

Bifet, A., G. Holmes, R. Kirkby, and B. Pfahringer (2010). Moa: Massive online analysis. *The Journal of Machine Learning Research 11*, 1601–1604.

Bishop, C. M. (2006). *Pattern recognition and machine learning.* springer.

Boyd, S. and L. Vandenberghe (2004). *Convex optimization.* Cambridge university press.

Brickell, J. and V. Shmatikov (2008). The cost of privacy: destruction of data-mining utility in anonymized data publishing. In *Proceedings of the 14th ACM SIGKDD international conference on Knowledge discovery and data mining*, pp. 70–78. ACM.

Carlini, N. and D. Wagner (2017). Towards evaluating the robustness of neural networks. In *Security and Privacy (SP), 2017 IEEE Symposium on*, pp. 39–57. IEEE.

Chandra, S., A. Haque, L. Khan, and C. Aggarwal (2016a). An adaptive framework for multistream classification. In *Proceedings of the 25th ACM International on Conference on Information and Knowledge Management*, CIKM '16, New York, NY, USA, pp. 1181–1190. ACM.

Chandra, S., A. Haque, L. Khan, and C. Aggarwal (2016b). Efficient sampling-based kernel mean matching. In *Data Mining (ICDM), 2016 IEEE 16th International Conference on*, pp. 811–816. IEEE.

Chandra, S., A. Haque, H. Tao, J. Liu, L. Khan, and C. Aggarwal (2018). Ensemble direct density ratio estimation for multistream classification. In *Data Engineering (ICDE), 2018 IEEE 34rd International Conference on*. IEEE.

Chandra, S., V. Karande, Z. Lin, L. Khan, M. Kantarcioglu, and B. Thuraisingham (2017). Securing data analytics on sgx with randomization. In *European Symposium on Research in Computer Security*, pp. 352–369. Springer.

Chandra, S., L. Khan, and F. B. Muhaya (2011). Estimating twitter user location using social interactions–a content based approach. In *Privacy, Security, Risk and Trust (PASSAT) and 2011 IEEE Third Inernational Conference on Social Computing (SocialCom), 2011 IEEE Third International Conference on*, pp. 838–843. IEEE.

Chandra, S., Z. Lin, A. Kundu, and L. Khan (2014). Towards a systematic study of the covert channel attacks in smartphones. In *International Conference on Security and Privacy in Communication Systems*, pp. 427–435. Springer.

Chandra, S., J. Sahs, L. Khan, B. Thuraisingham, and C. Aggarwal (2014). Stream mining using statistical relational learning. In *Data Mining (ICDM), 2014 IEEE International Conference on*, pp. 743–748. IEEE.

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

Chawla, N. V., L. O. Hall, K. W. Bowyer, and W. P. Kegelmeyer (2004). Learning ensembles from bites: A scalable and accurate approach. *Journal of Machine Learning Research 5*(Apr), 421–451.

Costan, V. and S. Devadas (2016). Intel sgx explained. *IACR Cryptology ePrint Archive 2016*, 86.

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

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

Datar, M., A. Gionis, P. Indyk, and R. Motwani (2002). Maintaining stream statistics over sliding windows. *SIAM J. Comput. 31*(6), 1794–1813.

Ditzler, G., M. Roveri, C. Alippi, and R. Polikar (2015). Learning in nonstationary environments: A survey. *Computational Intelligence Magazine, IEEE 10*(4), 12–25.

Domingos, P. and G. Hulten (2000). Mining high-speed data streams. In *Proceedings of the Sixth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '00, New York, NY, USA, pp. 71–80. ACM.

Dyer, K. B., R. Capo, and R. Polikar (2014). Compose: A semisupervised learning framework for initially labeled nonstationary streaming data. *IEEE Transactions on Neural Networks and Learning Systems 25*(1), 12 – 26.

Efron, B. (1992). *Bootstrap methods: another look at the jackknife*. Springer.

Fan, R.-E., K.-W. Chang, C.-J. Hsieh, X.-R. Wang, and C.-J. Lin (2008). Liblinear: A library for large linear classification. *The Journal of Machine Learning Research 9*, 1871–1874.

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

Fu, Y., E. Bauman, R. Quinonez, and Z. Lin (2017). Sgx-lapd: Thwarting controlled side channel attacks via enclave verifiable page faults. In *20th International Symposium on Research in Attacks, Intrusions, and Defenses (RAID).*

Gama, J., P. Medas, G. Castillo, and P. Rodrigues (2004). Learning with drift detection. In *Advances in artificial intelligence–SBIA 2004*, pp. 286–295. Springer.

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

Gentry, C. et al. (2009). Fully homomorphic encryption using ideal lattices. In *STOC*, Volume 9, pp. 169–178.

Götzfried, J., M. Eckert, S. Schinzel, and T. Müller (2017). Cache attacks on intel sgx. In *Proceedings of the 10th European Workshop on Systems Security*, pp. 2. ACM.

Gretton, A., A. Smola, J. Huang, M. Schmittfull, K. Borgwardt, and B. Schölkopf (2009). Covariate shift by kernel mean matching. *Dataset shift in machine learning 3*(4), 5.

Grosse, K., N. Papernot, P. Manoharan, M. Backes, and P. McDaniel (2017). Adversarial examples for malware detection. In *European Symposium on Research in Computer Security*, pp. 62–79. Springer.

Haque, A., S. Chandra, L. Khan, and C. Aggarwal (2014). Distributed adaptive importance sampling on graphical models using mapreduce. In *Big Data (Big Data), 2014 IEEE International Conference on*, pp. 597–602. IEEE.

Haque, A., S. Chandra, L. Khan, K. Hamlen, and C. Aggarwal (2017). Efficient multistream classification using direct density ratio estimation. In *Data Engineering (ICDE), 2017 IEEE 33rd International Conference on*, pp. 155–158. IEEE.

Haque, A., B. Dong, S. Chandra, Z. Wang, and L. Khan (2017). Fusion - an online method for multistream classification. In *Proceedings of the ACM International on Conference on Information and Knowledge Management*, CIKM '17. ACM.

Haque, A., L. Khan, and M. Baron (2015). Semi supervised adaptive framework for classifying evolving data stream. In *Pacific-Asia Conference on Knowledge Discovery and Data Mining*, pp. 383–394. Springer.

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

Haque, A., L. Khan, M. Baron, B. M. Thuraisingham, and C. C. Aggarwal (2016). Efficient handling of concept drift and concept evolution over stream data. In *32nd IEEE International Conference on Data Engineering, ICDE 2016, Helsinki, Finland, May 16-20, 2016*, pp. 481–492.

Haque, A., H. Tao, S. Chandra, J. Liu, and L. Khan (2018, Feb). A framework for multistream regression with direct density ratio estimation. In *Thirty-Second AAAI Conference on Artificial Intelligence.*

Haque, A., Z. Wang, S. Chandra, Y. Gao, L. Khan, and C. Aggarwal (2016). Sampling-based distributed kernel mean matching using spark. In *Big Data (Big Data), 2016 IEEE International Conference on*, pp. 462–471. IEEE.

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

Hitaj, B., G. Ateniese, and F. Perez-Cruz (2017). Deep models under the gan: Information leakage from collaborative deep learning. *arXiv preprint arXiv:1702.07464*.

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

Huang, L., A. D. Joseph, B. Nelson, B. I. Rubinstein, and J. Tygar (2011). Adversarial machine learning. In *Proceedings of the 4th ACM workshop on Security and artificial intelligence*, pp. 43–58. ACM.

Imani, M. B., S. Chandra, S. Ma, L. Khan, and B. Thuraisingham (2017). Focus location extraction from political news reports with bias correction. In *Big Data (Big Data), 2017 IEEE International Conference on*, pp. 1956–1964. IEEE.

Kanamori, T., S. Hido, and M. Sugiyama (2009). A least-squares approach to direct importance estimation. *The Journal of Machine Learning Research 10*, 1391–1445.

Karande, V., E. Bauman, Z. Lin, and L. Khan (2017). Sgx-log: Securing system logs with sgx. In *Proceedings of the 2017 ACM on Asia Conference on Computer and Communications Security*, pp. 19–30. ACM.

Kargupta, H., S. Datta, Q. Wang, and K. Sivakumar (2003). On the privacy preserving properties of random data perturbation techniques. In *Data Mining, 2003. Third IEEE International Conference on*, pp. 99–106. IEEE.

Kawahara, Y. and M. Sugiyama (2012, April). Sequential change-point detection based on direct density-ratio estimation. *Stat. Anal. Data Min. 5*(2), 114–127.

Kivinen, J., A. J. Smola, and R. C. Williamson (2010). Online learning with kernels. *IEEE Transactions on Signal Processing 100*(10), 1–12.

Kleiner, A., A. Talwalkar, P. Sarkar, and M. I. Jordan (2014). A scalable bootstrap for massive data. *Journal of the Royal Statistical Society: Series B (Statistical Methodology) 76*(4), 795–816.

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

Lee, S., M.-W. Shih, P. Gera, T. Kim, H. Kim, and M. Peinado (2016). Inferring fine-grained control flow inside sgx enclaves with branch shadowing. *arXiv preprint arXiv:1611.06952*.

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

Li, F., J. Sun, S. Papadimitriou, G. A. Mihaila, and I. Stanoi (2007). Hiding in the crowd: Privacy preservation on evolving streams through correlation tracking. In *Data Engineering, 2007. ICDE 2007. IEEE 23rd International Conference on*, pp. 686–695. IEEE.

Lichman, M. (2013). UCI machine learning repository.

Liu, C., X. S. Wang, K. Nayak, Y. Huang, and E. Shi (2015). Oblivm: A programming framework for secure computation. In *Security and Privacy (SP), 2015 IEEE Symposium on*, pp. 359–376. IEEE.

Liu, Y., X. Chen, C. Liu, and D. Song (2016). Delving into transferable adversarial examples and black-box attacks. *arXiv preprint arXiv:1611.02770*.

Luk, C.-K., R. Cohn, R. Muth, H. Patil, A. Klauser, G. Lowney, S. Wallace, V. J. Reddi, and K. Hazelwood (2005). Pin: building customized program analysis tools with dynamic instrumentation. In *ACM Sigplan Notices*, Volume 40, pp. 190–200. ACM.

Masud, M. M., J. Gao, L. Khan, J. Han, and B. Thuraisingham (2008). A practical approach to classify evolving data streams: Training with limited amount of labeled data. In *Data Mining, 2008. ICDM'08. Eighth IEEE International Conference on*, pp. 929–934. IEEE.

Masud, M. M., L. Khan, and B. Thuraisingham (2008). A scalable multi-level feature extraction technique to detect malicious executables. *Information Systems Frontiers 10*(1), 33–45.

McDiarmid, C. (1989). On the method of bounded differences. *Surveys in combinatorics 141*(1), 148–188.

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

Murphy, K. P. (2012). *Machine learning: a probabilistic perspective.* MIT press.

Navarro, G. (2001). A guided tour to approximate string matching. *ACM computing surveys (CSUR) 33*(1), 31–88.

Ohrimenko, O., F. Schuster, C. Fournet, A. Mehta, S. Nowozin, K. Vaswani, and M. Costa (2016). Oblivious multi-party machine learning on trusted processors. In *USENIX Security Symposium*, pp. 619–636.

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

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

Papernot, N., P. McDaniel, and I. Goodfellow (2016). Transferability in machine learning: from phenomena to black-box attacks using adversarial samples. *arXiv preprint arXiv:1605.07277*.

Papernot, N., P. McDaniel, I. Goodfellow, S. Jha, Z. B. Celik, and A. Swami (2016). Practical black-box attacks against deep learning systems using adversarial examples. *arXiv preprint arXiv:1602.02697*.

Papernot, N., P. McDaniel, X. Wu, S. Jha, and A. Swami (2016). Distillation as a defense to adversarial perturbations against deep neural networks. In *Security and Privacy (SP), 2016 IEEE Symposium on*, pp. 582–597. IEEE.

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

Rane, A., C. Lin, and M. Tiwari (2015). Raccoon: closing digital side-channels through obfuscated execution. In *24th USENIX Security Symposium (USENIX Security 15)*, pp. 431–446.

Ratner, A., S. H. Bach, H. Ehrenberg, J. Fries, S. Wu, and C. Ré (2017). Snorkel: Rapid training data creation with weak supervision. *arXiv preprint arXiv:1711.10160*.

Repository, U. M. L. (1998). `https://archive.ics.uci.edu/ml/datasets/`.

Schuster, F., M. Costa, C. Fournet, C. Gkantsidis, M. Peinado, G. Mainar-Ruiz, and M. Russinovich (2015). Vc3: trustworthy data analytics in the cloud using sgx. In *2015 IEEE Symposium on Security and Privacy*, pp. 38–54. IEEE.

Shalev-Shwartz, S., O. Shamir, N. Srebro, and K. Sridharan (2009). Stochastic convex optimization. In *COLT*.

Shih, M.-W., S. Lee, T. Kim, and M. Peinado (2017). T-sgx: Eradicating controlled-channel attacks against enclave programs. In *Proceedings of the 2017 Annual Network and Distributed System Security Symposium (NDSS), San Diego, CA*.

Shinde, S., Z. L. Chua, V. Narayanan, and P. Saxena (2016). Preventing page faults from telling your secrets. In *Proceedings of the 11th ACM on Asia Conference on Computer and Communications Security*, pp. 317–328. ACM.

Shokri, R., M. Stronati, C. Song, and V. Shmatikov (2017). Membership inference attacks against machine learning models. In *Security and Privacy (SP), 2017 IEEE Symposium on*, pp. 3–18. IEEE.

Sinha, R., S. Rajamani, S. Seshia, and K. Vaswani (2015). Moat: Verifying confidentiality of enclave programs. In *Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security*, pp. 1169–1184. ACM.

Stefanov, E., M. Van Dijk, E. Shi, C. Fletcher, L. Ren, X. Yu, and S. Devadas (2013). Path oram: an extremely simple oblivious ram protocol. In *Proceedings of the 2013 ACM SIGSAC conference on Computer & communications security*, pp. 299–310. ACM.

Sugiyama, M., M. Krauledat, and K.-R. Müller (2007). Covariate shift adaptation by importance weighted cross validation. *The Journal of Machine Learning Research 8*, 985–1005.

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

Vapnik, V. N. (1998). *Statistical Learning Theory.* Wiley-Interscience.

Wang, H., W. Fan, P. S. Yu, and J. Han (2003). 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, New York, NY, USA, pp. 226–235. ACM.

Widmer, G. and M. Kubat (1996). Learning in the presence of concept drift and hidden contexts. *Machine Learning 23*(1), 69–101.

Wu, T.-F., C.-J. Lin, and R. C. Weng (2004, December). Probability estimates for multi-class classification by pairwise coupling. *J. Mach. Learn. Res. 5*, 975–1005.

Xu, W., Y. Qi, and D. Evans (2016). Automatically evading classifiers. In *Proceedings of the 2016 Network and Distributed Systems Symposium.*

Xu, Y., W. Cui, and M. Peinado (2015). Controlled-channel attacks: Deterministic side channels for untrusted operating systems. In *Security and Privacy (SP), 2015 IEEE Symposium on*, pp. 640–656. IEEE.

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

Zhang, K., B. Schölkopf, K. Muandet, and Z. Wang (2013). Domain adaptation under target and conditional shift. In *International Conference on Machine Learning*, pp. 819–827.

Žliobaitė, I., A. Bifet, B. Pfahringer, and G. Holmes (2014). Active learning with drifting streaming data. *IEEE transactions on neural networks and learning systems 25*(1), 27–39.

BIOGRAPHICAL SKETCH

Swarup Chandra received his Bachelor of Engineering (BE) degree in Telecommunication Engineering from P.E.S Institute of Technology, Bangalore India (then affiliated with Visveswaraya Technological University, Belgaum, India) in May 2009. He enrolled in the Master's program in Computer Science at The University of Texas at Dallas (UTD) in the Fall of 2009. During his master's program, he interned at Ericsson Inc., Plano TX and later worked for the company after graduating in the fall of 2011. He then joined the PhD program at UTD in August 2013 to pursue his research interests under the supervision of Dr. Latifur Khan and Dr. Zhiqiang Lin. Here, he focused on two research areas: data stream mining and security. Swarup has published several papers at top-tier conferences such as ICDM, CIKM, AAAI, ICDE, and ESORICS. During his doctoral studies, he worked as a data science intern at Hewlett Packard Enterprise, Palo Alto and later continued to work with them under a research project.

CURRICULUM VITAE

Swarup Chandra

Contact Information:

Department of Computer Science
The University of Texas at Dallas
800 W. Campbell Rd.
Richardson, TX 75080, U.S.A.

Email: `swarup.chandra@utdallas.edu`

Educational History:

BE, Telecommunication Engineering, P.E.S Institute of Technology (Bangalore, India), 2009
MS, Computer Science, The University of Texas at Dallas, 2011
PhD, Computer Science, The University of Texas at Dallas, 2018

Scalable and Secure Learning with Limited Supervision Over Data Streams
Ph.D. Dissertation
Computer Science Department, The University of Texas at Dallas
Advisors: Dr. Latifur Khan and Dr. Zhiqiang Lin

Employment History:

Research Assistant, The University of Texas at Dallas. Aug. 2013 – Aug. 2018
Data Science Intern, Hewlett Packard Enterprise, Palo Alto CA. Aug. 2016 – Dec. 2016
Software Svrs Engineer, Ericsson Inc., Plano TX. Jan. 2012 – Aug. 2013
Engineer Intern, Ericsson Inc., Plano TX. Jul. 2010 – Dec. 2011

Professional Recognitions and Honors:

Ericsson Graduate Fellowship Award, UTD, 2017
Student Travel Grant, IEEE International Conference on Data Mining, 2014
Outstanding Academic Performance Award, UTD, 2011
National Talent Search Examination Scholar, NCERT New Delhi, 2004-09

Publications:

Karande, V., Chandra, S., Lin, Z., Caballero, J., Khan, L., & Hamlen, K. (2018, May). BCD: Decomposing Binary Code Into Components Using Graph-Based Clustering. In Proceedings of the 2018 on Asia Conference on Computer and Communications Security (pp. 393-398). ACM.

Chandra, S., Haque, A., Tao, H., Liu, J., Khan, L., & Aggarwal, C. (2018), Ensemble Direct Density Ratio Estimation for Multistream Classification". In the 34th IEEE International Conference on Data Engineering (ICDE)

Haque, A., Tao, H., Chandra, S., Liu, J., & Khan, L. (2018), A Framework for Multistream Regression with Direct Density Ratio Estimation", In AAAI Conference on Artificial Intelligence

Imani, M. B., Chandra, S., Ma, S., Khan, L., & Thuraisingham, B. (2017, December). Focus location extraction from political news reports with bias correction. In Big Data (Big Data), 2017 IEEE International Conference on (pp. 1956-1964). IEEE.

Chandra, S., Karande, V., Lin, Z., Khan, L., Kantarcioglu, M., & Thuraisingham, B. (2017, September). Securing data analytics on sgx with randomization. In European Symposium on Research in Computer Security (pp. 352-369). Springer, Cham.

Haque, A., Wang, Z., Chandra, S., Dong, B., Khan, L., & Hamlen, K. W. (2017, November). Fusion: An online method for multistream classification. In Proceedings of the 2017 ACM on Conference on Information and Knowledge Management (pp. 919-928). ACM.

Chandra, S., Haque, A., Khan, L., & Aggarwal, C. (2016, December). Efficient Sampling-Based Kernel Mean Matching. In Data Mining (ICDM), 2016 IEEE 16th International Conference on (pp. 811-816). IEEE.

Haque, A., Wang, Z., Chandra, S., Gao, Y., Khan, L., & Aggarwal, C. (2016, December). Sampling-based distributed Kernel mean matching using spark. In BigData (pp. 462-471).

Chandra, S., Haque, A., Khan, L., & Aggarwal, C. (2016, October). An adaptive framework for multistream classification. In Proceedings of the 25th ACM International on Conference on Information and Knowledge Management (pp. 1181-1190). ACM.

Al-Naami, K., Chandra, S., Mustafa, A., Khan, L., Lin, Z., Hamlen, K., & Thuraisingham, B. (2016, December). Adaptive encrypted traffic fingerprinting with bi-directional dependence. In Proceedings of the 32nd Annual Conference on Computer Security Applications (pp. 177-188). ACM.

Chandra, S., Karande, V., & Khan, L. (2015, December). A Comparative Study of Markov Network Structure Learning Methods Over Data Streams. In Computational Intelligence, 2015 IEEE Symposium Series on (pp. 795-802). IEEE.

Chandra, S., Sahs, J., Khan, L., Thuraisingham, B., & Aggarwal, C. (2014, December). Stream mining using statistical relational learning. In Data Mining (ICDM), 2014 IEEE International Conference on (pp. 743-748). IEEE.

Haque, A., Chandra, S., Khan, L., & Baron, M. (2014, December). Mapreduce guided approximate inference over graphical models. In Computational Intelligence and Data Mining (CIDM), 2014 IEEE Symposium on (pp. 446-453). IEEE.

Haque, A., Chandra, S., Khan, L., & Aggarwal, C. (2014, October). Distributed adaptive importance sampling on graphical models using mapreduce. In Big Data (Big Data), 2014 IEEE International Conference on (pp. 597-602). IEEE.

Chandra, S., Lin, Z., Kundu, A., & Khan, L. (2014, September). Towards a systematic study of the covert channel attacks in smartphones. In International Conference on Security and Privacy in Communication Systems (pp. 427-435). Springer, Cham.

Chandra, S., Khan, L., & Muhaya, F. B. (2011, October). Estimating twitter user location using social interactions–a content based approach. In Privacy, Security, Risk and Trust (PASSAT) and 2011 IEEE Third Inernational Conference on Social Computing (SocialCom), 2011 IEEE Third International Conference on (pp. 838-843). IEEE.