8-17-2017

# Learning extreme verification latency quickly with importance weighting: FAST COMPOSE & LEVEL_IW

Muhammad Umer
*Rowan University*

# LEARNING EXTREME VERIFICATION LATENCY QUICKLY WITH IMPORTANCE WEIGHTING: FAST COMPOSE & LEVEL$_{IW}$

by

Muhammad Umer

A Thesis

Submitted to the
Department of Electrical & Computer Engineering
College of Engineering
In partial fulfillment of the requirement
For the degree of
Master of Science in Electrical & Computer Engineering
at
Rowan University
August 09,2017

Thesis Chair: Robi Polikar, Ph.D.

## Dedication

*I would like to dedicate this work to my parents and my family*

## Acknowledgements

I would like to thank my advisor and mentor, Professor Robi Polikar, whose enthusiasm and vision in research has greatly inspired me, and whose constant guidance and encouragement has made this work possible. I also would like to extend my gratitude to Dr. Ravi Prakash Ramachandran and Dr. Umashangar Thayasivam for being part of my committee and for their generous time and commitment. Special thanks to Christopher Frederickson, who has been very helpful in paper reviews and code development. Finally, I would like to thank my family who has always supported and encouraged me to continue moving forward throughout my academic career.

## Abstract

Muhammad Umer
LEARNING EXTREME VERIFICATION LATENCY QUICKLY WITH IMPORTANCE WEIGHTING: FAST COMPOSE & LEVEL$_{IW}$
2016-2017
Robi Polikar, Ph.D.
Master of Science in Electrical & Computer Engineering

One of the more challenging real-world problems in computational intelligence is to learn from non-stationary streaming data, also known as concept drift. Perhaps even a more challenging version of this scenario is when – following a small set of initial labeled data – the data stream consists of unlabeled data only. Such a scenario is typically referred to as learning in initially labeled nonstationary environment, or simply as extreme verification latency (EVL). This thesis introduces two different algorithms to operate in this domain. One of these algorithms is a simple modification of our prior work, COMPOSE (COMPacted Object Sample Extraction), that allows the algorithm to work without its extremely computationally expensive core support extraction module. We call this modified algorithm FAST COMPOSE. The other algorithm we propose that works in this setting is based on the importance weighting domain adaptation approach. We explore importance weighting to match distributions between two consecutive time steps, and estimate the posterior distribution of the unlabeled data using importance weighted least squares probabilistic classifier. The estimated labels are then iteratively used as the training data for the next time step. We call this algorithm LEVEL$_{IW}$, short for Learning Extreme VErification Latency with Importance Weighting. An additional important contribution of this thesis is a comprehensive survey and comparative analysis of competing algorithms to point out the weaknesses and strengths of different approaches from three different perspectives: classification accuracy, computational complexity and parameter sensitivity using several synthetic and real world datasets.

# Table of Contents

Abstract . . . . . . . . . . v

List of Figures . . . . . . . . . . viii

List of Tables . . . . . . . . . . ix

Chapter 1: Introduction . . . . . . . . . . 1

1.1 Motivation: Learning in Non-Stationary Environments . . . . . . . . . . 1

1.2 Problem Statement . . . . . . . . . . 2

1.3 Scope of Thesis . . . . . . . . . . 3

1.4 Research Contributions . . . . . . . . . . 4

1.5 Organization of the Thesis . . . . . . . . . . 5

Chapter 2: Background . . . . . . . . . . 7

2.1 Concept Drift . . . . . . . . . . 7

2.1.1 Single Classifier vs. Ensemble Classifier Based Approaches for Concept Drift . . . . . . . . . . 8

2.1.2 Active vs. Passive Approaches for Concept Drift . . . . . . . . . . 9

2.2 Domain Adaptation . . . . . . . . . . 10

2.2.1 Unsupervised Domain Adaptation and Importance Weighting . . . . 12

2.2.2 Supervised Domain Adaptation . . . . . . . . . . 20

2.3 Transfer Learning . . . . . . . . . . 22

2.3.1 Inductive Transfer Learning . . . . . . . . . . 22

2.3.2 Transductive Transfer Learning . . . . . . . . . . 23

2.4 Verification Latency . . . . . . . . . . 23

Chapter 3: Preliminary and Related Work . . . . . . . . . . 27

3.1 Arbitrary Sub-Population Tracker Algorithm (APT) . . . . . . . . . . 27

**Table of Contents(Continued)**

3.2 Stream Classification Algorithm Guided by Clustering (SCARGC) . . . . . 32

3.3 Micro-Cluster for Classification (MClassification) . . . . . . . . . . . . . . 36

3.4 COMPOSE.V1 (Original COMPOSE With $\alpha$-Shape Construction) . . . . . 39

3.5 COMPOSE.V2 (COMPOSE With Gaussian Mixture Model (GMM) or Any Density Estimation Technique) . . . . . . . . . . . . . . . . . . . . . . . . . 42

Chapter 4: Improving Learning Concept Drift Under Extreme Verification Latency: Learning Extreme Verification Quickly With FAST COMPOSE and With Importance Weighting (LEVEL$_{\text{IW}}$) . . . . . . . . . . . . . . . . . . . . 46

4.1 The Underlying Problems With Current Approaches . . . . . . . . . . . . . 46

4.2 Learning Extreme Verification Latency Quickly: FAST COMPOSE . . . . . 51

4.3 LEVEL$_{\text{IW}}$: Learning Extreme VErification Latency With Importance Weighting . . . . . . . . . . . . . . . . . . . . . . . . . 53

4.3.1 Importance Weighted Least-Squares Probabilistic Classifier . . . . . 55

4.3.2 LEVEL$_{\text{IW}}$ . . . . . . . . . . . . . . . . . . . . . . . . . . . . 61

Chapter 5: Experiments, Results and Comprehensive Analysis of Algorithms for Learning Under Extreme Verification Latency . . . . . . . . . . . . . . 64

5.1 Analysis of Three Versions of COMPOSE . . . . . . . . . . . . . . . . . . 71

5.2 Analysis of SCARGC . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 75

5.3 Analysis of MClassification . . . . . . . . . . . . . . . . . . . . . . . . . 77

5.4 Analysis of LEVEL$_{\text{IW}}$ . . . . . . . . . . . . . . . . . . . . . . . . . 82

5.5 Analysis on two Additional Real World Datasets . . . . . . . . . . . . . . . 85

Chapter 6: Conclusion and Future Work . . . . . . . . . . . . . . . . . . . . . 95

6.1 Summary of Future Work . . . . . . . . . . . . . . . . . . . . . . . . . . . 98

References . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 99

## List of Figures

Figure | Page
Figure 1. Graphical representation of covariate shift and class imbalance . . . . . . 18
Figure 2. Graphical representation of verification latency . . . . . . . . . . . . . . 24
Figure 3. Graphical representation of extreme verification latency scenario . . . . . 25
Figure 4. Block diagram and Graphical Representations of APT . . . . . . . . . . . 31
Figure 5. Block diagram and Graphical Representations of SCARGC . . . . . . . . 34
Figure 6. Block diagram and Graphical Representations of MClassification . . . . . 39
Figure 7. Block diagram and operation workflow of COMPOSE . . . . . . . . . . . 43
Figure 8. Example depicting core support extraction procedure in COMPOSE . . . 50
Figure 9. Block diagram and graphical illustration of FAST COMPOSE . . . . . . 53
Figure 10. Block diagram and graphical representation of LEVEL$_{\text{IW}}$ . . . . . . . . . 62
Figure 11. Progress of drift for 1CDT, 1CHT, 2CDT, and 2CHT datasets . . . . . . . 67
Figure 12. Three different snapshots of GEARS_2C_2D data . . . . . . . . . . . . 68
Figure 13. Three different snapshots of UG_2C_3D data . . . . . . . . . . . . . . . . 68
Figure 14. Progress of drift for 4CR, 4CRE-V2, 1Csurr, and 4CE1CF datasets . . . . 69
Figure 15. Four different snapshots of *5CVT* dataset . . . . . . . . . . . . . . . . . . 72
Figure 16. Accuracy Comparison of COMPOSE on balanced 5CVT dataset . . . . . 73
Figure 17. Classification accuracy comparison of SCARGC on 5CVT dataset . . . . 76
Figure 18. Six Snapshots of $FG\_2C\_2D$ data . . . . . . . . . . . . . . . . . . . . . . 78
Figure 19. Accuracy of Algorithms on $FG\_2C\_2D$ data . . . . . . . . . . . . . . . 79
Figure 20. Six Snapshots of $MG\_2C\_2D$ data . . . . . . . . . . . . . . . . . . . . . 80
Figure 21. Accuracy of Algorithms on $MG\_2C\_2D$ data . . . . . . . . . . . . . . . 81
Figure 22. UG_2C_2D data for six different snapshots . . . . . . . . . . . . . . . . . 83
Figure 23. Accuracy of algorithms on $UG\_2C\_2D$ data . . . . . . . . . . . . . . . . 84
Figure 24. Accuracy of algorithms on real world *weather* data . . . . . . . . . . . . 87
Figure 25. Sample images of traffic scenes streaming from a traffic camera . . . . . . 87
Figure 26. Accuracy of algorithms on *Traffic* dataset using various batch sizes . . . . 89

## List of Tables

Table Page

Table 1. Dataset descriptions . . . . . . . . . . . . . . . . . . . . . . . . . . . . 66

Table 2. Average classification accuracy . . . . . . . . . . . . . . . . . . . . . . . 90

Table 3. Average execution time (in seconds) . . . . . . . . . . . . . . . . . . . . 91

Table 4. Statistical significance at $\alpha = 0.05$ for classification accuracy . . . . . . 91

Table 5. Statistical significance at $\alpha = 0.05$ for execution time . . . . . . . . . . 92

Table 6. Accuracy with three different values of k (SCARGC) . . . . . . . . . . 92

Table 7. Accuracy with three different values of $r$ for (MClassification) . . . . . 93

Table 8. Accuracy with three different values of k (COMPOSE) . . . . . . . . . 93

Table 9. Accuracy with three different values of sigma (LEVEL$_{\text{IW}}$) . . . . . . . . 94

# Chapter 1

# Introduction

## 1.1 Motivation: Learning in Non-Stationary Environments

The fundamental goal in machine learning is to learn from data. Depending on the availability of labeled data, machine learning can be broadly categorized into three categories, namely supervised learning (where sufficient amount of labeled data are available for training), unsupervised learning (where only unlabeled data are available), and semi-supervised learning (where some labeled data and some unlabeled data are available). Most machine learning algorithm, regardless of the availability of labeled data, make a fundamental assumption that data are drawn from a fixed but unknown distribution. This assumption implies that test or field data come from the same distribution as the training data. In reality, this assumption simply does not hold in many real world problems that generate data whose underlying distributions change over time. Network intrusion, web usage and user interest analysis, natural language processing, speech and speaker identification, spam detection, anomaly detection, analysis of financial, climate, medical, energy demand, or pricing data, as well as the analysis of signals from autonomous robots and devices, brain signal analysis, and bio-informatics are just a few examples of the real world problems where underlying distributions may – and typically do – change over time.

In machine learning, the challenge of making decisions in a changing environment is referred to as non-stationary learning. This is a challenging problem, because the classifier needs to adapt to a new concept in the changing environment, while retaining the previously acquired knowledge that is still relevant to ensure a stable learning environment, a phenomenon commonly referred to as the stability-plasticity dilemma in literature [1].

The fixed distribution assumption, essentially requiring the data to be drawn independently from an identical distribution (also referred to as independent and identically distributed - i.i.d.) renders traditional learning algorithms that make this assumption ineffective at best, misleading and inaccurate at worst on non-stationary distribution problems.

## 1.2 Problem Statement

Concept drift techniques [2–7] and domain adaptation approaches [8, 9] have been developed to tackle two related but different issues related to non-stationary distributions: domain adaptation techniques are designed to handle mismatched training and test distribution over a single time-step, while concept drift approaches are designed to track the data distributions over a streaming setting. However, both approaches assume that there is (preferably ample) labeled training data, and the potential scarcity or the high cost of obtaining labeled data is a major obstacle faced by these approaches.

In an effort to reduce the amount of required labeled data, semi supervised learning (SSL) approaches have also been employed, where a hypothesis is formed using modest amount of labeled data and more abundant unlabeled data. SSL approaches, of course, also require labeled data at each time step [10], albeit in smaller quantities. Active learning (AL) is another approach to combat the limited availability of labeled data [11], where the learner actively chooses which data instances – if labeled – would provide the most benefit. The goal in AL algorithms is therefore to find the minimum number of labeled examples that provide the maximum benefit. This is most commonly achieved by assuming that there is an oracle or expert that can be queried for the labels of any example on demand. Active learning approaches cannot function, however, if the requested labels cannot be provided on demand, a potentially restricting limitation.

The unavailability of labeled data, particularly in streaming applications, gives rise to another problem, commonly referred to as *verification latency* in the literature [12], where labeled data are not available at every time step. More specifically, verification latency refers to the scenario where labels of the training data becoming available only certain or some unspecified amount of time later, significantly complicating the learning process. The duration of the lag in obtaining labeled data may not be known a priori, and/or may vary with time. The extreme case of this phenomenon, aptly named as the *extreme verification latency*, is perhaps the most challenging case of all machine learning problems: labels for the training data are never available - except perhaps those provided initially, yet the classification algorithm is asked to learn and track a drifting distribution with no access to labeled data. This thesis explores solutions to this problem of learning from non-stationary and streaming environments in the presence of extreme verification latency.

## 1.3 Scope of Thesis

The primary goal of this thesis is to develop effective (in terms of classification performance) and efficient (in terms of computational cost associated with the algorithm) approaches for learning in extreme verification latency (EVL). EVL refers to the scenario where obtaining labeled data is expensive or impractical and – perhaps beyond an initial investment – only unlabeled data are available in all future time-steps of a non-stationary data stream. We refered to this scenario as *initially labeled non-stationary environment* (ILNSE) in our prior work [13]. The existing approaches to work in this setting include i)

Arbitrary Sub-Population Tracker (APT); [14], ii) COMPacted Object Sample Extraction (COMPOSE) [13]; iii) Stream Classification Algorithm Guided by Clustering (SCARGC) [15]; iv) and Micro-cluster for Classification (MClassification) [16].

The specific focus of this thesis is not only to develop cost effective and time efficient approaches in this setting, but also to compare and contrast existing approaches, determine if they can be improved through appropriate modifications. Within this setting, we also investigate whether domain adaptation approaches – typically designed to work for single time-step distribution mismatch problems – can be modified to work in streaming setting, and more importantly under EVL.

## 1.4 Research Contributions

The primary focus of this thesis, as mentioned above, is to develop effective and efficient approaches for learning under extreme verification latency in non-stationary environments, and compare the performances of the small set of algorithms that are designed to work in similar settings. The core contributions and findings of this work are as follows:

1. FAST COMPOSE is introduced as a very efficient algorithm that can work under extreme verification latency. FAST COMPOSE is a modification of the algorithm COMPOSE, previously developed by Dyer and Polikar [13], where its computationally expensive core support extraction module is replaced by using all of the instances labeled by the algorithm in the previous time-step.

2. We observe that *importance weighting* based domain adaptation approaches can be used for streaming data concept drift problems associated with extreme verification latency when the class conditional distributions at the consecutive time steps share

support.

3. A modification of the well-known *importance weighted least squares probabilistic classifier* is introduced so that it can work within a) streaming data environment and b) when there is extreme verification latency. The proposed approach is called Learning Extreme VErification Latency with Importance Weighting (LEVEL$_{IW}$).

4. One of the most important contributions of this thesis is to provide a detailed and comprehensive comparison and analysis of competing algorithms used in extreme verification latency setting from three different perspectives: accuracy, computational complexity, and parameter sensitivity. We find that FAST COMPOSE is the best algorithm among others with respect to accuracy and computational complexity, while LEVEL$_{IW}$ is the best algorithm with respect to the parameter sensitivity.

## 1.5 Organization of the Thesis

Chapter 2 provides an overview and background for learning in nonstationary environments, concept drift, domain adaptation and ensemble approaches used for concept drift. Existing approaches for learning in non-stationary environments under extreme verification latency setting are discussed in detail in Chapter 3. Chapter 4 introduces the new algorithms developed as part of this thesis, specifically, FAST COMPOSE developed to learn in extreme verification latency setting quickly; and LEVEL$_{IW}$ that extends covariate shift based domain adaptation approaches to learning under EVL. Chapter 5 presents the experimental setup and results, comparing and analyzing competing algorithms for non-stationary learning under extreme verification latency from three different perspectives: accuracy, computational complexity, and parameter sensitivity. Finally, the conclusions and the suggestions for future work are discussed in Chapter 6.

# Chapter 2

# Background

This chapter provides a comprehensive background review and technical details of two primary areas that related to learning in non-stationary environments, namely concept drift and domain adaptation. The general overview of these topics along with the connection and concerns with verification latency are also discussed in this chapter.

## 2.1 Concept Drift

Concept drift refers to the scenario where the statistical properties of the data change over time in unforeseen ways. Concept drift is not a trivial problem because it occurs within the streaming data which is usually unlabeled and unstructured. The drift scenarios can be abrupt or gradual, slow or fast, random or systematic, cyclical or otherwise. Changes can also be perceived, rather than real, due to insufficient, unknown or unobservable features - referred to as hidden context, where an underlying unknown phenomenon provides a true and static description over time [17],[18]. Concept drift problems typically assume at least a gradual (or limited) drift assumption, but do not require stationary posteriors or same support. So in concept drift we normally have $p_{t+1}(y|x) \neq p_t(y|x)$ where $p_{t+1}(x) = p_t(x)$ may or may not be satisfied. IN other words, the posterior distribution of the data at time $t + 1$ may be different from that at time $t$, while marginal distribution may or may not remain the same as well. This scenario is also known as *real drift*.

Data is presented in streams to the concept drift handling algorithms in normally two different ways: *i. Online setting* - where a single instance is provided to the learner at each time-step, and the learner has to adapt to the change using this single instance; and *ii. batch setting* - where several instances are accumulated from the stream, which are then presented to the learner. Online setting is normally considered to be a more challenging learning scenario than the batch setting because less data (i.e. single instance at each time-step) makes it difficult for the learner to adapt to the changes easily. On the other hand, batch learners often lag in reacting to changing concepts because it is often assumed that concept does not change within the given batch of data. Of course, the stationarity within a batch assumption is rarely true.

Concept drift algorithms can be characterized in various ways; such as single classifier vs. ensemble-based approaches, or active vs. passive approaches.

**2.1.1 Single classifier vs. Ensemble classifier based approaches for Concept drift.** Single-classifier approaches learn the drifting concept by either replacing the current classifier with a new classifier trained on newly received data, or updating the adjustable parameters of a given classifier to reflect changes present in newly received data [19],[20], [21]. Single classifier approaches are more prone to stability-plasticity dilemma: an entirely stable learner would not be able to learn changing environment and an entirely plastic learner would not be able to deal with catastrophic forgetting [22], i.e. it would not be able to retain any of the previous knowledge that may still be relevant. Learning algorithms based on single-classifier approaches strive to balance stability and plasticity.

On the other hand, ensemble based approaches use a combination of several classifiers to make a decision, hence minimizing the stability-plasticity problems, albeit at increased computational cost. Ensemble approaches track the environment by adding new classifiers with each incoming dataset to build a family of classifiers. These approaches minimize the stability-plasticity dilemma concerns by providing both stability (a subset or all of the prior classifiers can be retained or reweighted) and plasticity (learning new information by adding new classifiers). A fixed or dynamic ensemble size may be used by these approaches. For fixed ensemble size, either the oldest member [23], [24] or the least contributing ensemble member is replaced with a new one as done in Dynamic Weighted Majority (DWM) algorithm in [25]. If dynamic ensemble size is being used, additional classifiers can be added without removing existing classifiers (though they would often be reweighted to reduce their impact). Weighted [26] or simple majority [27] voting are the most common approaches for combining the classifiers when an ensemble approach is used. Abdulsalam et al.'s random forests with entropy [28], Masud et al.s concept drift with time constraints [29], Bifets integration of a Kalman filter with Adaptive Sliding Window (ADWIN) [30], and Massive Online Analysis [31] represent ensemble approaches that combine ensemble of classifiers and sliding window techniques. Learn$^{++}$.NSE [32], [4], and Learn$^{++}$.NIE [5] represent a more modern family of approaches for mining data streams with concept drift that do not rely on sliding window, and can dynamically determine which ensemble members are relevant at any given time.

**2.1.2 Active vs. Passive approaches for Concept drift.** In active approaches, the algorithm continuously monitors the data to determine if and when change occurs. If – and only if – a change is detected, the algorithm takes an appropriate action, such as updating the classifier with the most recent data or simply creating a new classifier to learn the current data, depending on the nature of the algorithm. Passive approaches, on the other hand, do not explicitly monitor the data for change, but rather assume change may occur at any time new data become available. A passive algorithm therefore updates the model every time new data arrive, regardless of the presence of change/drift. There are a multitude of active drift detection approaches. Many of the earliest algorithms for concept drift were Window based approaches, such as STAGGER [33], FLORA [17], and their variants. These algorithms use a sliding window to choose a block of new data to train a new classifier when change was detected and are example of active drift detection approaches. Other approaches include statistical control charts as used in Alippi and Roveri's just-in-time (JIT) classifiers[34], and the more recent intersection of confidence intervals (ICI) rule [35]. Information theoretic measures [36], Hoeffding bounds or Hellinger distance [37], [38] are other active approaches that are based on monitoring classifier's accuracy or some metric to detect the change and updating the classifier.

## 2.2 Domain Adaptation

Domain adaptation refers to the learning scenario when the data distribution used to train the model is different than that of the data on which the learner needs to predict. Within the context of domain adaptation, the training data is referred to as *source data*, whereas the test or field data is known as the *target data*, with the corresponding data distributions being referred to as source and target data distributions, respectively. Domain adaptation problems are typically not associated with streaming data as there is only a single time step. Examples of domain adaptation problems include, e.g., *speaker identification* where the source data distribution may vary from that of the target data due to the recording environment change, physical conditions/emotions, and session-dependent variations [39]. Another example is *brain-computer interface (BCI)*, which allows direct communication from human brain to machine to control an external entity [40]. Electroencephalography (EEG) is often the signal of choice in BCI applications, and EEG signals are known to be extremely nonstationary [41]. In BCI experiments, training samples and unlabeled test samples are usually gathered in different recording sessions, and the non-stationarity in the brain signals can cause a change in the distributions rendering the classifier update necessary in this setting. *Natural language processing (NLP)* is another case, where the NLP system is trained using data collected from the target domain in which the system is operated, however, due to the difference in vocabulary and writing style, the target domain data is often not useful to train a system, requiring some domain adaptation intervention [42]. Age prediction from face images has also been an application, where the type of camera, the camera calibration, and lighting variations significantly influence the accuracy of age prediction systems. The system is usually trained on the publicly available databases which are mainly collected in a semi-controlled environments with appropriate illumination. However, in the real-world testing environments, lighting conditions vary considerably: there may be either not enough light or strong light. For this reason, training and test data tend to have different distributions [43].

The aim of domain adaptation techniques is therefore to build a hypothesis that is robust to the changes (or drift) between training (source) and test (target) distributions. Domain adaptation approaches can also be characterized in several ways: Supervised domain adaptation refers to the scenario where labeled data are available both in source and target domains, whereas unsupervised domain adaptation typically refers to the case where both labeled and unlabeled examples are available in the source domain, but only unlabeled data are available in the target or test domain. The intermediate scenario where there is some, but very limited labeled data are available in the target domain, is also commonly treated using the approaches designed for unsupervised domain adaptation, but with ideas borrowed from semi-supervised learning.

### 2.2.1 Unsupervised domain adaptation and Importance weighting.

Importance weighting is perhaps the most common approach used to tackle unsupervised domain adaptation where no labeled examples are available in the target domain.

The essential cause of domain adaptation problem is the difference between the joint distribution $p_t(x, y)$ of features $x$ and labels $y$ in the target domain and the joint distribution $p_s(x, y)$ in the source domain [44]. One possible solution to this problem is to weigh (or transform) the training instances such that their distribution behaves more like that of the target distribution.

For classification problems, the goal is to find a good mapping function $f$ between inputs $x$ and outputs $y$ among a set of all candidate functions in the hypothesis space $H$. The optimal choice $f^*$ should then minimize the expected loss with respect to the true distribution $p(x, y)$. Specifically in domain adaptation problem setting, the optimal function $f_t^*$ for the target domain, the one that minimizes the expected loss with respect to the target domain is

$$f_t^* = \underset{f \in H}{\operatorname{argmin}} \sum_{(x,y) \in X \times Y} p_t(x, y) L(x, y, f) \tag{2.1}$$

where $L(x, y, f)$ is the loss function. Since we do not have sufficient (or any) labeled instances in the target domain, we can not obtain a good approximation of the actual target distribution, i.e., empirical target distribution $\widetilde{p_t}(x, y)$ from the target domain instances. By empirical distribution, we refer to the approximation of the true distribution estimated by using sufficient amount of labeled examples, i.e., $(x_i, y_i)$. In domain adaptation setting we do, of course, have access to a sufficient set of labeled instances from the source domain, but since these instances are drawn from the source distribution $p_s(x, y)$, the empirical distribution estimated from these instances, $\widetilde{p_s}(x, y)$, can not directly help us approximate $p_t(x, y)$. We can, however, rewrite Equation 2.1 in a different way that can indirectly help us use labeled examples from source distribution [45].

$$\begin{aligned} f_t^* &= \underset{f \in H}{\operatorname{argmin}} \sum_{(x,y) \in X \times Y} \frac{p_t(x, y)}{p_s(x, y)} p_s(x, y) L(x, y, f) \\ &\approx \underset{f \in H}{\operatorname{argmin}} \sum_{(x,y) \in X \times Y} \frac{p_t(x, y)}{p_s(x, y)} \widetilde{p_s}(x, y) L(x, y, f) \\ &= \underset{f \in H}{\operatorname{argmin}} \frac{1}{N_s} \sum_{i=1}^{N_s} \frac{p_t(x_i, y_i)}{p_s(x_i, y_i)} L(x_i, y_i, f) \end{aligned} \tag{2.2}$$

where, $\widetilde{p_s}(x, y)$ is the empirical source distribution estimated using sufficient ($N_s$) number of labeled instances in the source domain. The expected loss value is then estimated by calculating simple mean of the loss across source domain instances $(x_i, y_i)$ weighted by $\frac{p_t(x_i, y_i)}{p_s(x_i, y_i)}$. This ratio, computed by using $N_s$ source instances, is known as the *importance ratio*. Equation 2.2 implies that importance weighting provides a good approximation and justified solution to the domain adaptation problem through providing the estimated optimal function value for the target domain.

There are two main lines of work in the literature [44] to compute this ratio, which are discussed below.

**2.2.1.1** ***Covariate shift.*** Shimodaira [8] introduced the term *covariate shift* in which source and target distribution are related to each other by making the assumption that conditional distributions of class $y$ given the same instance $x$ in both source and target domain are the same, while marginal distributions of $x$ may change, i.e., $p_s(y|x) = p_t(y|x)$ but $p_s(x) \neq p_t(x)$. Covariate Shift is conceptually illustrated in Figure 1(a). Under covariate shift assumption the importance ratio can be rewritten as

$$\frac{p_t(x,y)}{p_s(x,y)} = \frac{p_t(y|x)p_t(x)}{p_s(y|x)p_s(x)} = \frac{p_t(x)}{p_s(x)} \tag{2.3}$$

In this scenario, we only need to estimate $p_t(x)/p_s(x)$. Hardle et al. [46] use a non-parametric kernel density estimation (KDE) approach with Gaussian Kernel to estimate importance ratio by estimating two densities individually, i.e., $p_t(x)$ and $p_s(x)$, however, they find that KDE suffers from curse of dimensionality. Therefore, KDE based approaches are not typically reliable in high-dimensional problems. Sugiyama et al. [43] propose to directly estimate this $p_t(x)/p_s(x)$ ratio, by minimizing the *Kullback Leibler divergence* between the estimated importance value and true importance value, where the estimation of the true importance value is calculated using a linear model. Bickel et al. [47] estimate the importance ratio directly using the probabilistic classifier. The problem of directly estimating the importance ratio value can also be transformed into a kernel mean matching problem (KMM) in reproducing kernel Hilbert space as done by Huang et al [48]. The key idea of covariate shift adaptation is to use informative training samples by considering their importance in predicting test output values. The two primary approaches for estimating the importance ratio, i.e., kernel density estimation to individually calculate the distributions and then estimating the importance ratio, and probabilistic technique to directly estimate the importance ratio, are briefly discussed below:

1. Kernel Density Estimation: Kernel density estimation is a non-parametric technique used to estimate probability density function $p(x)$ from its i.i.d. samples $\{x_i; i =$

$1, ., n\}$. KDE can be expressed as follows for the Gaussian Kernel in d-dimensional case

$$\hat{p}(x) = \frac{1}{n(2\pi\sigma^2)^{\frac{d}{2}}} \sum_{k=1}^{n} k_\sigma(x, x_i) \tag{2.4}$$

where $k_\sigma(x, x_i) = \exp(-\frac{(||x-x_i||)^2}{2\sigma^2})$ is a Gaussian Kernel, centered at instance $x_i$. As mentioned above, Hardle et al. proposed individually calculating the two distributions $p_t(x)$ and $p_s(x)$ using Gaussian kernel density estimation, and then use these calculated distributions to estimate the importance ratio $p_t(x)/p_s(x)$ [46]. The performance of Gaussian kernel density estimation depends on the choice of the kernel width $\sigma$, for which the authors propose the standard cross-validation procedure: essentially they chose the value of $\sigma$ that maximizes the average of the following holdout log-likelihood probability

$$\frac{1}{|\chi_r|} \sum_{x \in \chi_r} \log \hat{p}_{\chi_r}(x) \tag{2.5}$$

where $|\chi_r|$ denotes the number of elements in the set $\chi_r$. The procedure is repeated for $r = 1, 2, ..., k$, where $k$ represents the number of disjoint subsets into which the samples $x_{i}{}_{i=1}^{n}$ are divided. As with most cross-validation approaches, the procedure of individually calculating the distributions is computationally very expensive and not feasible for high dimensional problems.

2. Logistic Regression: Another approach to directly estimate importance ratio $\frac{p_t(x)}{p_s(x)}$ is to use a probabilistic classifier. Bickel et al. show that importance can be expressed in terms of the variable $\eta$, and propose to rewrite target and source distributions as follows [47]

$$p_t(x) = p(x|\eta = 1); p_s(x) = p(x|\eta = -1) \tag{2.6}$$

where $\eta$ is a selector variable, $\eta = -1$ means that samples are drawn from the training distribution and $\eta = 1$ means that they are drawn from the test distribution. Using Bayes rule we then have

$$\frac{p_t(x)}{p_s(x)} = \frac{p(x|\eta = 1)}{p(x|\eta = -1)} = \frac{p(\eta = 1|x)p(x)}{p(\eta = 1)} \frac{p(\eta = -1)}{p(\eta = -1|x)p(x)} \tag{2.7}$$

ultimately the importance ratio is simplified to

$$\frac{p_t(x)}{p_s(x)} = \frac{p(\eta = -1)}{p(\eta = 1)} \frac{p(\eta = 1|x)}{p(\eta = -1|x)} \tag{2.8}$$

In this formulation, instead of individually estimating the marginal distributions, the conditional probability of a single binary variable $\eta$ needs to be modeled. The likelihood probability $p(\eta|x)$ can be approximated using a probabilistic model that discriminates training examples from the test examples, and outputs how much more likely an instance is to occur in the training data than it is to occur in the test data. The authors propose to use a *logistic regression* as a probabilistic model to estimate $p(\eta|x)$, and use the empirical approximation $\frac{p(\eta=-1)}{p(\eta=1)} \approx \frac{n_s}{n_t}$, where $\frac{n_s}{n_t}$ is the ratio of number of training examples (in the source domain) to the number of test examples (in the target domain).

**2.2.1.2** ***Class imbalance.*** In calculating the importance ratio, another scenario for establishing a relationship between source and target distributions is to assume that class conditional likelihood probabilities of the features are the same (i.e., given the label $y$, the conditional distribution of features $x$ are the same in both domains), whereas the prior probabilities of the class labels may be different. Mathematically, this scenario can be described as $p_s(x|y) = p_t(x|y)$ but $p_s(y) \neq p_t(y)$, a phenomenon also known as class imbalance problem [49] and depicted in Figure 1(b). Under the class-imbalance assumption, the importance ratio can be rewritten as

$$\frac{p_t(x,y)}{p_s(x,y)} = \frac{p_t(x|y)p_t(y)}{p_s(x|y)p_s(y)} = \frac{p_t(y)}{p_s(y)} \tag{2.9}$$

Class imbalance problem is typically addressed by resampling (over sampling the minority class or under sampling the majority class) [50]. In the context of domain adaptation, training instances are resampled from the source domain so that the re-sampled instances have approximately the same data distribution as the test domain. In other words, underrepresented classes are over-sampled and overrepresented classes are under-sampled. As an example, resampling technique is used by [51] to solve time-series forecasting problem, a challenging task as time-series data often exhibit systematic changes in the distribution of the observed values. It becomes even more challenging when the time-series data possess significant imbalance, i.e., certain ranges of values are over-represented in comparison to others, and the user is particularly interested in the predictive performance on values that are the least represented. An example of such case is financial data analysis, where number of legitimate transactions far outnumber those of fraudulent transactions.

**2.2.1.3 *Semi supervised learning.*** Semi supervised learning is the branch of machine learning that makes use of both labeled and unlabeled data to build a hypothesis. If we associate the source domain with labeled data (as we typically have sufficient labeled data from the source domain), and associate the target domain with unlabeled data (as we often have abundant unlabeled data but little or no labeled data from the target distribution), the domain adaptation problem can be recast as a semi-supervised learning problem. The primary difference, however, in domain adaptation there is typically abundant labeled data

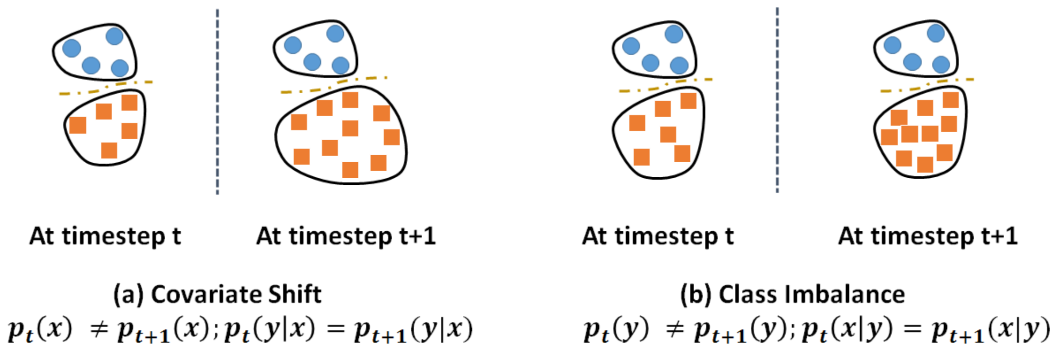

At timestep t
At timestep t+1
At timestep t
At timestep t+1
(a) Covariate Shift
$p_t(x) \neq p_{t+1}(x); p_t(y|x) = p_{t+1}(y|x)$
(b) Class Imbalance
$p_t(y) \neq p_{t+1}(y); p_t(x|y) = p_{t+1}(x|y)$

*Figure* 1. Graphical representation of covariate shift and class imbalance; (a) In covariate shift, marginal distributions change between timesteps while posterior distribution remains the same; (b) In class imbalance, Class priors change between timesteps, while the likelihood distributions remain the same

from the source distribution, whereas most SSL algorithms assume little or no labeled data availability. There is a significant body of work in using semi-supervised learning to handle domain adaptation problems, some of which are briefly discussed below.

1. Semi-supervised Domain Adaptation with Subspace Learning (SDASL): A novel domain adaptation framework that jointly employs three *regularizers* is proposed in [52]. This integration of three different regularizers attempt to correct the distribution mismatch by projecting the original features from both source and target domains to a lower dimensional subspace using linear predictive model. The first aim is to explore invariant low dimensional structures across domains, minimize the domain divergence through empirical risk minimization with a regularization penalty over the linear predictive model parameters, and ultimately seek a decision boundary that achieves a small classification error. This procedure is called *structural risk regularization*. While learning a good feature subspace, the distance between mapping of similar samples in both source and target domain is restricted by incorporating a dis-

criminative regularization term in the empirical risk minimization objective function through the procedure known as *structural preservation regularizer*. Finally *manifold regularizer*, based on the smoothness assumption of semi-supervised learning, is utilized to measure the smoothness of predicted data along with the inherent structure of unlabeled target data. In other words, the outputs of the predictive function are restricted to have similar values for similar examples.

2. Expectation Maximization Algorithm for Domain Adaptation: An expectation maximization (EM) algorithm is proposed in [53] for domain adaptation, where the initial model is estimated from the source data under source distribution. The initial model is treated as the poor estimation of the target distribution for target data. The EM algorithm is applied to find a local optimum in the hypothesis space over target distribution, where the estimation should gradually approach the target distribution. Kullback Leibler (KL)-divergence between source and target domains is used to estimate the trade-off parameter $p_t(D_i)$ between labeled and unlabeled data, where $(p_t(D_i); i \in (s,t))$ is the probability of data (either source data or target data) under target distributions, i.e., the probability of source data $D_s$ under target distribution $p_t(D_s)$ or probability of target data $D_t$ under target distribution $p_t(D_t)$.

3. Generalized Distillation Semi-supervised Domain Adaptation (GDSDA): GDSDA is proposed in [54] to effectively transfer knowledge from the source domain to the target domain using unlabeled data. A framework consisting of two models, the *teacher (source) model* and the *student (target) model* is used by GDSDA. The knowledge can be directly transferred from the teacher (source) model to the student (target)

model without directly accessing the data used to train the teacher. Specifically, the target model is trained using originally unlabaled target data obtained from the target distribution, but with "soft" labels of this data as obtained from (predicted by) the teacher model. The target model is also trained to minimize the difference between the soft labels and the hard (actual) labels. The importance between hard labels and soft labels is balanced by *imitation parameter*, whose value is generally determined using the brute force search or domain knowledge but the authors propose a novel imitation parameter estimation method for GDSDA, called GDSDA-SVM, which uses SVM as the base classifier and determines the imitation parameter efficiently. In particular, the mean square error loss for GDSDA-SVM is used and leave-one-out cross validation (LOOCV) loss is computed. The optimal parameter is found by minimizing the LOOCV loss.

4. Semi-supervised Instance Weighting: Traditional instance weighting for domain adaptation, as discussed above, only uses weighted source domain instances as training data. An alternate approach is proposed in [42] to not only include weighted source domain instances but also weighted unlabeled target domain instances in the training data to handle domain adaptation problem. Hence this approach is closer to the spirit of true semi-supervised instance weighting.

**2.2.2 Supervised domain adaptation.** Recall that domain adaptation is caused by the difference in joint probability distribution $p_s(x, y)$ of the source data and the joint probability distribution $p_t(x, y)$ of the target data. Covariate shift is the most commonly used domain adaptation scenario to characterize the difference in the distributions by estimating the ratio of these two distributions as described above, but covariate shift makes the additional assumption that posterior probability distributions are the same in both source and target domains, i.e. $p_s(y|x) = p_t(y|x)$, however it is possible that this assumption does not hold in many practical cases.

When the assumption of posterior distribution remaining the same across the domains is not satisfied, the importance ratio can not be simplified as was the case in Equation 2.3, and therefore, must be written as

$$\frac{p_t(x,y)}{p_s(x,y)} = \frac{p_t(x)p_t(y|x)}{p_s(x)p_s(y|x)}. \tag{2.10}$$

The optimal function $f_t^*$ described above must then be obtained as

$$f_t^* = \underset{f \in H}{\operatorname{argmin}} \frac{1}{N_s} \sum_{i=1}^{N_s} \frac{p_t(x_i)p_t(y_i|x_i)}{p_s(x_i)p_s(y_i|x_i)} L(x_i, y_i, f) \tag{2.11}$$

The authors in [45] propose a heuristic solution for those cases where the posterior distribution of source and target data differ. Instead, a different assumption of availability of some labeled data in the target domain along with the source domain is made. A logistic regression model $p_t(y|x;\theta_t)$ is first learned from the available labeled target data, where $\theta_t$ is the model parameter. Then, the labels of source domain examples under target domain i.e. $p_t(y_i|x_i)$ are predicted using the trained model from labeled target examples. Another logistic regression model $p_s(y|x;\theta_s)$ is also trained this time using labeled examples from the source domain and using this model predict the labels of source domain examples under source domain i.e. $p_s(y_i|x_i)$. Finally a direct estimation of the term $\frac{p_t(y_i|x_i)}{p_s(y_i|x_i)}$ in equation 2.11 is proposed. In other words, the estimation of the ratio between posterior distribution of the source instances under target domain and posterior distribution of the source instances under source domain is given as follows

$$\frac{p_t(y_i|x_i)}{p_s(y_i|x_i)} = \frac{p_t(y_i|x_i;\theta_t)}{p_s(y_i|x_i;\theta_s)} \tag{2.12}$$

## 2.3 Transfer Learning

Transfer learning refers to a machine learning procedure, where knowledge learned in the previous tasks are applied to novel tasks that are new but related domains. Transfer learning also refers to the ability of a system to recognize those new domains that share some commonality. In other words, the goal in transfer learning is to identify the commonality between the given target task and the previous (source) tasks, and then transfer the knowledge from the source tasks to the target task. Two useful surveys on transfer learning from two different perspectives of machine learning can be found in literature; one is the survey on transfer learning for reinforcement learning applications [55], whereas the other is a survey on transfer learning for classification and regression problems [56]. Transfer learning can be categorized into two important categories as inductive transfer learning and transductive transfer learning [56], as briefly discussed below.

**2.3.1 Inductive transfer learning.** In inductive transfer learning, regardless of the similarity of the source and target domains, the target task is distinctly different from the source task. When abundant labeled source domain data are available, inductive transfer learning is similar to the multi-task learning [57] with one main difference: inductive transfer learning attempts to transfer knowledge from the source domain to the target domain with the goal of achieving high performance in the target domain, whereas multi-task learning attempts to simultaneously learn the source and target tasks. On the other hand, when no labeled data is available in the source domain, inductive transfer learning can be considered similar to the self-taught learning [58], which uses unlabeled data to learn higher level representations of input, and use such representations to significantly improve the classification performance. One example of inductive transfer learning, where source and target task is different, is trying to recognize trucks (target task) by applying knowledge gained while learning to recognize cars (source task).

**2.3.2 Transductive transfer learning.** In transductive transfer learning (TTL), the source and target *domains* are different but the source and the target *tasks* are the same. In other words, either input features are different in two domains, for example classification of two sets of documents described in different language, or the marginal probability distribution of input features is different in two domains, for example in the same document classification problem, source domain documents and target domain documents focus on different topics. Furthermore, in TTL, no labeled data in the target domain is available, while abundant labeled data in the source domain are available. When the difference in the source and target domain is due to the difference in the marginal probability distribution of the source and target data, TTL can be related to the covariate shift (domain adaptation) as discussed above, or sample selection bias as discussed in [59].

## 2.4 Verification Latency

While unlabeled data are available in abundance, obtaining labeled data at every time step of a streaming environment is often problematic in many real world applications as it is either time consuming (e.g., document classification that requires human experts or annotators to classify each document), expensive (e.g., medical diagnostics that require medical professionals and monetary cost associated with running various diagnostic tests), or even possibly dangerous (e.g., obtaining label information for land-mine detection).

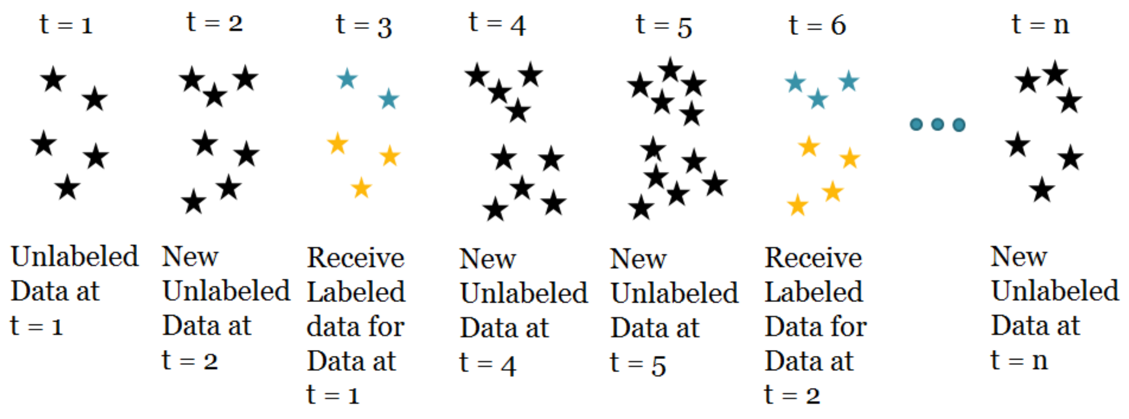

t = 1
t = 2
t = 3
t = 4
t = 5
t = 6
t = n
Unlabeled Data at t = 1
New Unlabeled Data at t = 2
Receive Labeled data for Data at t = 1
New Unlabeled Data at t = 4
New Unlabeled Data at t = 5
Receive Labeled Data for Data at t = 2
New Unlabeled Data at t = n

*Figure* 2. Graphical representation of verification latency: unlabeled data are received during first two time steps $t = 1$ and $t = 2$, with labels for the $t = 1$ data are received at $t = 3$. New unlabeled data are received at $t = 4$ and $t = 5$, followed by labels for data received at $t = 2$. The process continues receiving label for data for previous timesteps in a possibly irregular intervals.

In such cases, limited amount of data may get labeled, and event then, only with a delay, and not immediately after data first becoming available. Such a scenario is referred to as *verification latency*, which acknowledges an additional and important constraint that must be addressed in streaming environments: labeled data may not be available at every time-step, nor even in regular intervals, which in turn significantly complicates the learning process. Verification latency, as denoted in [12], describes a scenario where true class labels are not made available until sometime after the classifier has made a prediction on the current state of the environment. The duration of this lag may not be known a priori, and may vary with time; yet, classifiers must propagate information forward until the model can be verified. The graphical representation of this phenomenon is shown in Figure 2.

In the *extreme verification latency* scenario, this lag becomes infinite, meaning that no labeled data are ever received after initialization, as illustrated in Figure 3. We call such an environment as an *initially labeled non stationary environment* (ILNSE) or simply initially labeled streaming environment (ILSE) [13].

Real-world examples of such an extreme learning setting are rapidly growing because of massive automated and autonomous acquisition of sensor, web user, weather, financial transaction, energy usage, and other data. Furthermore, such applications can be increasingly important. For example, network intrusion with malicious software (malware) attacks, where malware programmers are able to modify the malware faster than network security can identify and neutralize it, is a major current day challenge. Creating a labeled database for this scenario is difficult and expensive.

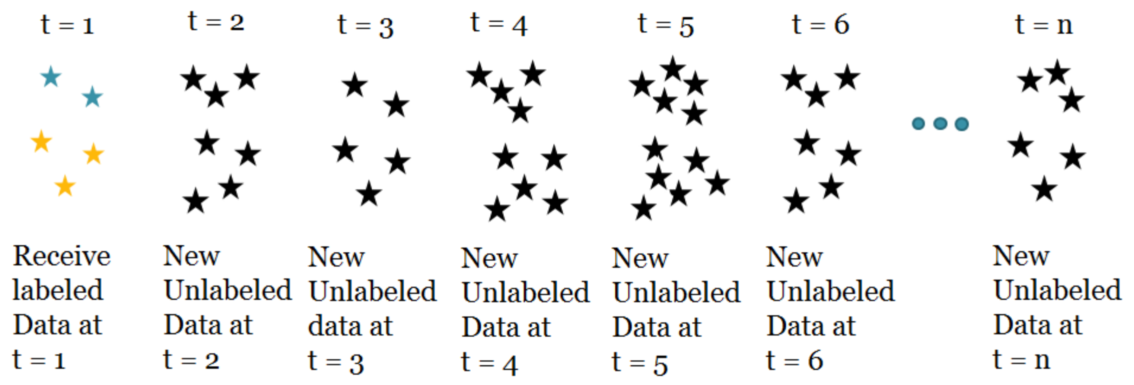

t = 1
t = 2
t = 3
t = 4
t = 5
t = 6
t = n
Receive labeled Data at t = 1
New Unlabeled Data at t = 2
New Unlabeled data at t = 3
New Unlabeled Data at t = 4
New Unlabeled Data at t = 5
New Unlabeled Data at t = 6
New Unlabeled Data at t = n

*Figure* 3. Graphical representation of extreme verification latency scenario: labeled data are received initially at $t = 1$; then only unlabeled data are received thereafter for $t = 2, 3, ...., n$

Many automation applications provide other examples, such as robotic control systems, drones, and autonomous vehicles. Just the recent popularization of drones opens new challenges to the computerized automation of flights of these aerial vehicles. Given a drone initially trained in a known environment, they need to incrementally adapt to changes in speed and direction of the wind, altitude, temperature, and atmospheric pressure in an unsupervised manner.

# Chapter 3

# Preliminary and Related Work

In this chapter, we describe in detail the algorithms currently available in the literature for learning from a streaming nonstationary data in the presence of extreme verification latency (EVL). As a relatively new field of machine learning, there are only a handful of algorithms that can address learning in an EVL scenario. These algorithms are Arbitrary Sub-Population Tracker (APT), Stream Classification Algorithm Guided by Clustering (SCARGC), Micro-cluster for Classification (MClassification) and Compacted Object Sample Extraction (COMPOSE).

## 3.1 Arbitrary Sub-Population Tracker Algorithm (APT)

The Arbitrary Sub-Population Tracker (APT) is proposed by Krempl [14] to handle extreme verification latency problem under certain assumptions and specific scenarios, and is based on the principle that each class in the data can be represented as a mixture of arbitrarily distributed sub-populations. The APT algorithm makes the following assumptions [60]:

1. The underlying population of the feature space contains several sub-populations, each of which drifts (possibly) differently over time;

2. The data generated from this feature space can be represented with a mixture model of several drifting components;

3. Initial labeled data are used to represent each sub-population of the feature space, where a sub-population is defined as a mode in the class-conditional distribution

$p(y|x)$, with $p(y)$ representing the prior distribution of the class labels, and $p(x)$ representing the marginal feature distribution;

4. A multimodal class distribution is represented by individual sub-populations to be tracked within a single class; furthermore every instance of the feature space must be labeled at the initialization;

5. The drift only affects the conditional feature distributions $p(x|z)$, where $p(z)$ represents the components' prior distributions, i.e., the mixing proportions of components used in the mixture model to represent data;

6. The drift is gradual and systematic that can be represented as a piecewise linear function;

7. The conditional posterior distribution $p(y|z)$ remains fixed, i.e., a components class label cannot change

8. The prior distribution of components, $p(z)$, is static

9. The posterior distribution is independent of the (latent) component membership, $p(y|z) = p(y|z, x)$; and

10. Co-variance of each component remains constant.

Non-parametric kernel density estimation is used to estimate conditional feature distributions $p(x|z)$ of the components, using $M$ samples, $X = x_1, x_2, .., x_M$. Krempl uses the common choice of Gaussian (radial basis) kernel for estimation, however any standard kernel estimator can be used for this purpose, such as the polynomial kernel. The standard kernel estimator modeling $\hat{p}(x)$ is given as

$$\hat{p}(x) = \frac{1}{M}\sum_{m=1}^{M} K_X(x - x_m) \tag{3.1}$$

where, $K_X(x - x_m)$ is the kernel function. When a D-dimensional Gaussian kernel is used as the kernel, we then have

$$K_X(x - x_m) = (2\pi)^{\frac{D}{2}}|\mathbf{C}^{-1}|^{\frac{1}{2}} \exp\{-\frac{1}{2}(x - x_m)^T\mathbf{C}^{-1}(x - x_m)\} \tag{3.2}$$

where $\mathbf{C}$ is the covariance or generally referred to as bandwidth of the Gaussian kernel function.

A modification to the standard Gaussian kernel is proposed to actually model the conditional feature distribution $\hat{p}(x|z)$, instead of simply modeling the feature distribution $\hat{p}(x)$. The modified Gaussian kernel incorporates each component $z$ of the data, allows different bandwidth matrix for each component, and also accounts for the drift present in the data. In other words, the Gaussian kernel is modified to better fit APT to work in the non-stationary environments. The adjusted kernel estimator accounting for drift present in the data is given as

$$\hat{p}(x|z) = \hat{p}(x|z,t) = \frac{1}{M}\sum_{m=1}^{M} G_m(x,t) \tag{3.3}$$

where $G_m(x,t)$ is the modified Gaussian Kernel and is represented as

$$G_m(x,t) = (2\pi)^{\frac{D}{2}}|\mathbf{C}_{z_m}^{-1}|^{\frac{1}{2}} \exp\{-\frac{1}{2}d_m^T\mathbf{C}_{z_m}^{-1}d_m\} \tag{3.4}$$

where $\mathbf{C}_{z_m}$ allows there to be a different bandwidth matrix for each component $z$, and $d_m = x - (\widetilde{x}_m)(t)$ is the difference between position $x$ and the estimated position $\widetilde{x}_m$ of the $m^{th}$ component at time $t$. Here, the estimated position is computed as $(\widetilde{x}_m)(t) =$

$x_m + (t - t_m) * \mu^{\Delta}_{z_m}$, where $\mu^{\Delta}_{z_m}$ represents the component movement vector of the $m^{th}$ component center. The initial cluster position is indicated by $\mu^{0}_{z_m}$.

The learning strategy of APT is twofold; first, the optimal one-to-one assignment between labeled instances in time-step $t$ and unlabeled instances in time-step $t+1$ is determined using expectation maximization (EM) algorithm. The EM algorithm begins with the expectation step by predicting which instances are most likely to correspond to a given sub-population. During the maximization step, the algorithm determines which drift parameters maximize the expectation. Then, the classifier is updated to reflect the population parameters of the newly received data and drift parameter relating the previous time step to the current one. Following the assumption that $p(z)$ remains static, the algorithm creates a one-to-one mapping of an instance in time step $t$ to a corresponding instance in time step $t+1$. Given a set of $M$ known examples and a set of $N$ new observations at positions $X = x_1, x_2, .., x_N$ at times $T = t_1, t_2, .., t_N$, the problem corresponds to the following likelihood maximization problem

$$L(\Theta, X, T) = \prod_{n=1}^{N} \prod_{m=1}^{M} G_m(x_n, t_n)^{z_{nm}} \tag{3.5}$$

where $\Theta = \{\mu^0_1, ...., \mu^0_k, \mu^{\Delta}_1, ...., \mu^{\Delta}_k\}$, and $z_{nm}$ is the latent instance-to-exemplar correspondence, which is equal to 1 if instance $n$ corresponds to exemplar $m$ and 0 otherwise.

Establishing a one-to-one relationship while identifying drift requires an impractical assumption that the number of instances remains constant throughout all time steps. Krempl relaxes this assumption by establishing a relationship in a batch method - matching a random subset of exemplars to a subset of new observation until all new observations have been assigned a relationship to an exemplar.

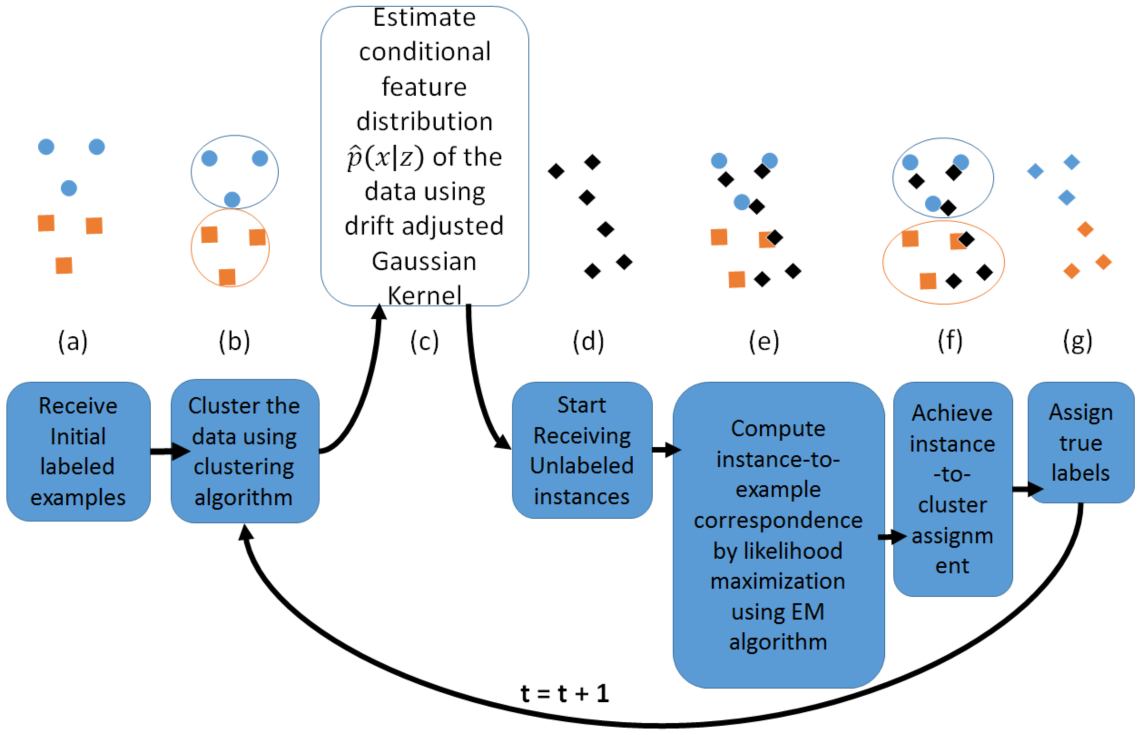

Estimate conditional feature distribution $\hat{p}(x|z)$ of the data using drift adjusted Gaussian Kernel
(a)
(b)
(c)
(d)
(e)
(f)
(g)
Receive Initial labeled examples
Cluster the data using clustering algorithm
Start Receiving Unlabeled instances
Compute instance-to-example correspondence by likelihood maximization using EM algorithm
Achieve instance -to- cluster assignm ent
Assign true labels
t = t + 1

*Figure* 4. Block diagram and Graphical Representations of APT; (a) Receive initial labeled examples (represented by blue circles and orange rectangles), (b) Perform clustering of the data (represented by blue and orange circles around the data), (c) Estimate the conditional feature distribution of the data $p(x|z)$ using modified Gaussian Kernel given in equation 3.3, (d) Start receiving unlabeled examples (represented by black diamonds), (e) Maximize the likelihood function given in equation 3.5 to compute instance-to-example correspondence, (f) Pass the same cluster assignment from the examples to its assigned instances to achieve instance-to-cluster assignment, (g) Assign same label of the example to its assigned instance.

Krempl suggests a bootstrap method that can make the one-to-one assignments more robust, but at an additional computational cost. When the assumptions are satisfied, APT works very well. However, APT has two primary weaknesses: 1) some of its assumptions often do not hold true, causing a decrease in performance, and 2) it is computationally very expensive [13].

The pseudocode for APT algorithm is given in Algorithm 1, while the graphical representation of the algorithm illustrating its corresponding stages through block diagrams is given in Figure 4.

Algorithm 1. **Arbitrary Subpopulation Tracker (APT)**

```
Inputs: Initial labeled data $D_{init}$; A clustering algorithm with its own free parameters; a suitable bandwidth matrices calculation algorithm; a suitable expectation-maximization (EM) algorithm with its free parameters
1: Receive $M$ training examples form $D_{init} = \{x_i; y_i\}$; $i = 1, ..., M$ ; $x \in X; y \in Y = \{1, ..., c\}$;
2: Run clustering algorithm to partition the data into $K$ disjoint subsets and associate each cluster to one class among $c$ classes ;
3: Estimate the conditional feature distribution of the data $\hat{p}(x|z)$ using equation 3.3;
4: Receive new unlabeled instances $U^t$ = $\{x_u^t \in X$ , $u = 1, ..., N\}$ and assume $N = M$ to associate each new instance to one previous example;
5: Compute instance-to-exemplar correspondence by maximizing the likelihood given in equation 3.5 using EM algorithm;
6: Pass the cluster assignment from the example to their assigned instances to achieve instance-to-cluster assignment;
7: Pass the class of an example $x_i$ i.e. $y_i$ to the class of its assigned instance;
8: Go to step 2 and Repeat.
```

## 3.2 Stream Classification Algorithm Guided by Clustering (SCARGC)

Souza et al. proposed an alternate algorithm, SCARGC, to solve the extreme verification latency problem [15]. SCARGC is a clustering-based algorithm that repeatedly clusters unlabeled input data, and then classifies the clusters using the labeled clusters from the previous time-step. SCARGC also makes several assumptions:

1. A small amount of labeled data is available initially to define the problem;

2. The drift is gradual / incremental, which allows tracking of the classes with only unlabeled information. Incremental drift assumption as used in SCARGC requires

significant overlap between class distributions in subsequent time steps and short intervals of time;

3. The number of classes is known and fixed ahead of time.

Given the aforementioned assumptions, the algorithm builds an initial classification model using the available labeled data from $c$ classes, and then divide the initial labeled data into $k \geq c$ clusters where $k$ is a user-selected free parameter. If user selects $k = c$, SCARGC uses $c$ classes as initial clusters, otherwise a clustering subroutine finds clusters and associates each cluster with one class. Souza denotes this initial set of $k$ clusters as $C^0 = C_1^0, C_2^0, ., C_k^0$. As new unlabeled data are received, the algorithm stores each example in a pool, and predicts its label using the initial classification model. After a fixed number of examples, also pre-determined by the user, are received and stored in the pool, the pool of examples is clustered into $k$ clusters in the same way as initial labeled data are clustered, i.e., by using $c$ classes as initial clusters if $k = c$, otherwise running a clustering subroutine to associate each cluster with one class. The new set of clusters are denoted as $C^1 = C_1^1, C_2^1, , C_k^1$. Each new cluster $C_i^1 \in C^1$ is then associated with (linked to) one of the previous clusters $C_j^0 \in C^0$ to assign each cluster to one class. The classification model is updated using the recently labeled examples. The algorithm then repeats the loop, alternating between clustering and classification. The labels are decided by associating clusters $C^t$ in the current iteration with the labels of clusters $C^{t-1}$ from the previous iteration. The mapping between the clusters is performed by centroid similarity between current and previous iterations using Euclidean distance. Given the current centroids from the most recent unlabeled clusters and past centroids from the previously labeled clusters, one-nearest neighbor algorithm (or support vector machine) is used to label the centroid from current unlabeled clusters.

SCARGC is computationally efficient, but its performance is highly dependent on the clustering phase. It also requires some prior knowledge such as the number of classes and the number of modes for each class in the data, the latter of which may limit the use of this algorithm when such information is not available.

The block diagram representing different stages of SCARGC with accompanying illustrations is given in Figure 5.

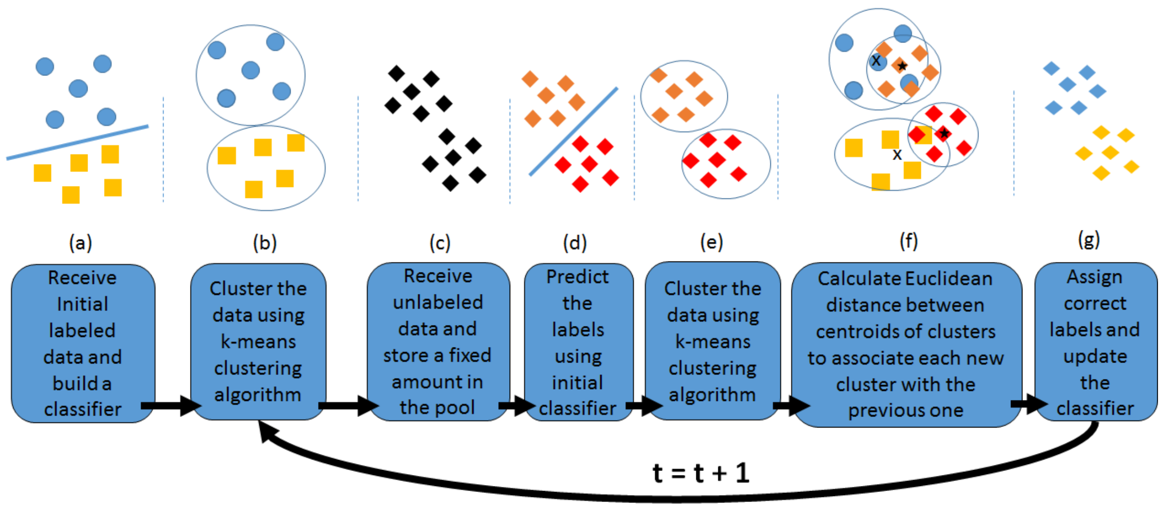

(a)
(b)
(c)
(d)
(e)
(f)
(g)
Receive Initial labeled data and build a classifier
Cluster the data using k-means clustering algorithm
Receive unlabeled data and store a fixed amount in the pool
Predict the labels using initial classifier
Cluster the data using k-means clustering algorithm
Calculate Euclidean distance between centroids of clusters to associate each new cluster with the previous one
Assign correct labels and update the classifier
t = t + 1

*Figure* 5. Block diagram and Graphical Representations of SCARGC; (a) Receive initial labeled data and classify it using 1-NN or SVM classifier $\phi$, (b) Cluster the initial data into $k$ clusters to form initial clusters, (c) start receiving unlabeled examples and store them in a pool, (d) initial classifier model $\phi$ is used to predict their labels, (e) cluster the unlabeled examples labeled by $\phi$ using $k$-means clustering to create new clusters at current iteration, (f) Perform mapping between clusters from previous and current iteration using centroid similarity, (g) Assign correct labels to unlabeled examples and update $\phi$. The process repeats by clustering the newly labeled data in the previous step

The algorithm receives initial labeled data and classifies it using 1-NN or SVM classifier $\phi$ as shown in Figure 5(a). In Figure 5(b), SCARGC clusters the data into $k$ clusters to form initial clusters. The algorithm then receives unlabeled examples and stores them in a pool as shown in Figure 5(c), and uses the initial classifier model $\phi$ to predict their labels (Figure 5(d)). In Figure 5(e), the algorithm clusters the unlabeled examples labeled by $\phi$ using $k$-means clustering to create new clusters at current iteration. The mapping between clusters from previous and current iteration are then obtained using centroid similarity (Figure 5(f)), the labels are assigned, and the classifier is updated $\phi$. This process continues as long as new unlabeled data are received.

The pseudocode for SCARGC algorithm is given in Algorithm 2

Algorithm 2. **SCARGC**

```
Inputs: Initial training data $D_{init}$, maximum pool size $N$, number of clusters $k$;
 1: Receive initial labeled data $D_{init} = \{x_i; y_i\}$ ; $i = 1, ..., M$ ; $x \in X; y \in Y = \{1, ..., c\}$
 2: Build initial classifier $\phi$ using $D_{init}$
 3: Run $k$-means clustering algorithm to divide the data into $k$ clusters; $\{C^t = C_1^t, C_2^t, ..., C_k^t\}$ and associate each cluster with one of the $c$ classes
 4: Start receiving new unlabeled examples from unlabeled data stream $U = \{x_u \in X\}$
 5: Store the next batch of $N$ examples in a pool
 6: Predict labels of stored examples using classifier $\phi$ as $D_{new} = \{x_u; \phi(x_u)\}; u = 1, ..., N$
 7: Run $k$-means clustering algorithm on $D_{new}$ to obtain $\{C^{t+1} = C_1^{t+1}, C_2^{t+1}, ..., C_k^{t+1}\}$
 8: Establish a mapping between current and previous clusters: the current clusters $C^{t+1}$ are associated to previous clusters $C^t$ by measuring similarity between their centroids $q_t^i; i = \{1, ..., k\}$ using Euclidean distance, i.e., $Dist(q_t, q_{t+1})$ where $Dist$ represents Euclidean distance
 9: Assign current centroid $q_{t+1}^i$ the label $\hat{y}_i$ which is same label $y_i$ of the closest past centroid $q_t^i$
10: The current dataset now has the updated correct labels from the previous step as $D_{t+1} = \{x_u; \hat{y}_u)\}; u = 1, ..., N$
11: Update the initial classifier $\phi$ using $D_{t+1}$
12: Go to step 4 and repeat
```

## 3.3 Micro-Cluster for Classification (MClassification)

Souza et al. also proposed *MClassification*, an algorithm that uses the idea of micro clusters (MC) [16] to adapt to the changes in the data over time, and learn the concepts under extreme verification latency. A Microcluster (MC) is a compact representation of the data points $\vec{x_i}; i = \{1, ..., N\}$, that includes the sufficient statistics of the data and are represented in triplets $(N, \vec{LS}, \vec{SS})$, where $N$ is the number of data points in the cluster, $\vec{LS}$ is the linear sum of $N$ data points represented as $\vec{LS} = \{\vec{x_1} + \vec{x_2} + ..... + \vec{x_n}\}$, and $\vec{SS}$ is the square sum of data points represented as $\vec{SS} = \{\vec{x_1}^2 + \vec{x_2}^2 + ..... + \vec{x_n}^2\}$. Thus a MC summarizes the information about the set of $N$ data points, from which we can calculate the centroid and radius of the MC using the following equations

$$centroid = \frac{\vec{LS}}{N} \tag{3.6}$$

$$Radius = \sqrt{\frac{\vec{SS}}{N} - (\frac{\vec{LS}}{N})^2} \tag{3.7}$$

There exist two interesting properties of MC, referred to as *incrementality* and *additivity*, which make them suitable for the streaming problems. The *incrementality* property states that if we are given a set of data points whose statistics are stored in a micro-cluster $A$ as $MC_A = (N_A, (\vec{LS}_A), (\vec{SS}_A))$, we can incrementally add a new example $\vec{x}$ in $MC_A$ updating the statistics of data points in the following way

$$(\vec{LS}_A) \leftarrow (\vec{LS}_A) + \vec{x} \tag{3.8}$$

$$(\vec{SS}_A) \leftarrow (\vec{SS}_A) + (\vec{x})^2 \tag{3.9}$$

$$N_A \leftarrow N_A + 1 \tag{3.10}$$

whereas the *additivity* property provides that, if we have two disjoint Micro-clusters $MC_A$ and $MC_B$, the union of these two groups is equal to the sum of its parts. Thus the sufficient statistics of a new Micro-Cluster $MC_C = (N_C, (\vec{LS_C}), (\vec{SS_C}))$, that stores the information of $MC_A \cup MC_B$ are computed as:

$$(\vec{LS_C}) \leftarrow (\vec{LS_A}) + (\vec{LS_B}) \tag{3.11}$$

$$(\vec{SS_C}) \leftarrow (\vec{SS_A}) + (\vec{SS_B}) \tag{3.12}$$

$$N_A \leftarrow N_B + N_C \tag{3.13}$$

Although MC is efficient and appropriate for data streaming problems, the authors observe that MC representation has been commonly used in clustering problems. In order to use MC to classify evolving data streams, the authors modify the representation to store information about the class of data points, thus their representation is a 4-tuple $(N, \vec{LS}, \vec{SS}, y)$, where $y$ is the label for a set of data points. The working of the algorithm is presented below.

The algorithm begins by receiving the initial labeled data $D_{init}$, using which it builds a set of labeled MCs, where each MC has information about only one example. The algorithm then starts receiving the unlabeled data stream. A label $\hat{y}_t$ is then predicted for each example $\vec{x_t}$ from the stream based on its nearest MC, computed with respect to Euclidean distance in the classification phase. The example $\vec{x_t}$ is added to its corresponding nearest MC, say $MC_N$, using the *incrementality* property of MC. Now the updated radius of $MC_N$ is computed and the algorithm checks if the updated radius of $MC_N$ exceeds the maximum micro-cluster radius threshold $r$ defined by the user. If the radius does not exceed the threshold $r$, the example $\vec{x_t}$ remains added in $MC_N$ and its updated centroid is also computed. The centroid position of the updated MC, i.e., $MC_N$ is therefore slightly moved in direction of the newly emerging concept of the class for new example added. On the other hand, if the radius exceeds the threshold, a new MC say $MC'_N$ carrying the predicted label $\hat{y}_t$ is created to allocate the new example $\vec{x_t}$. The process is repeated for each newly received unlabeled example.

The descriptive diagram illustrating different stages of the process is given in Figure 6. The pseudocode for MClassification algorithm with the implementation details is provided in Algorithm 3.

Algorithm 3. **MClassification**

```
Inputs: Maximum micro-cluster radius $r$;
1: Receive initial labeled data $D_{init} = \{x_i; y_i\}$ ; $i = 1, ..., T$ ; $x \in X; y \in Y = \{1, ..., c\}$
2: Build $T$ micro-clusters as $MC_i = (N_i, LS_i, SS_i, y_i); i = 1, ..., T$ where $N$ = number
   of data points ; $LS = \sum_{j=1}^{N} x_j$ ; $SS = \sum_{j=1}^{N} (x_j)^2$
3: Calculate sufficient statistics of each micro-cluster as follows $centroid_i =$
   $\frac{\vec{LS_i}}{N_i}; Radius_i = \sqrt{\frac{\vec{SS_i}}{N_i} - (\frac{\vec{LS_i}}{N_i})^2}$
4: Receive one new unlabeled example $\vec{x_t}$ from the unlabeled data stream
   $U = \{x_u \in X\}$
5: Measure distance between $\vec{x_t}$ and each micro-cluster centroids $centroid_i; i =$
   $\{1, ..., T\}$ i.e. $Dist(centroid_i, \vec{x_t})$ to find closest micro-cluster say $MC_N$, where $Dist$
   represents the Euclidean distance
6: Assign label of $MC_N$ i.e. $\hat{y}_t$ to classify example $\vec{x_t}$
7: Add example $\vec{x_t}$ to $MC_N$ and compute its sufficient statistics $radius_N$ ; and $centroid_N$
8: if $radius_N > r$ then
9:     Create a new micro-cluster for example $\vec{x_t}$ say $MC'_N = (N'_N, LS'_N, SS'_N, \hat{y}_t)$
10: else
11:     Add example $\vec{x_t}$ to $MC_N$ and update its statistics as $(\vec{LS}_N) \leftarrow (\vec{LS}_N) +$
        $\vec{x_t}; (\vec{SS}_N) \leftarrow (\vec{SS}_N) + (\vec{x_t})^2; N_N \leftarrow N_N + 1$
12: end if
13: Go to step 4 and repeat
```

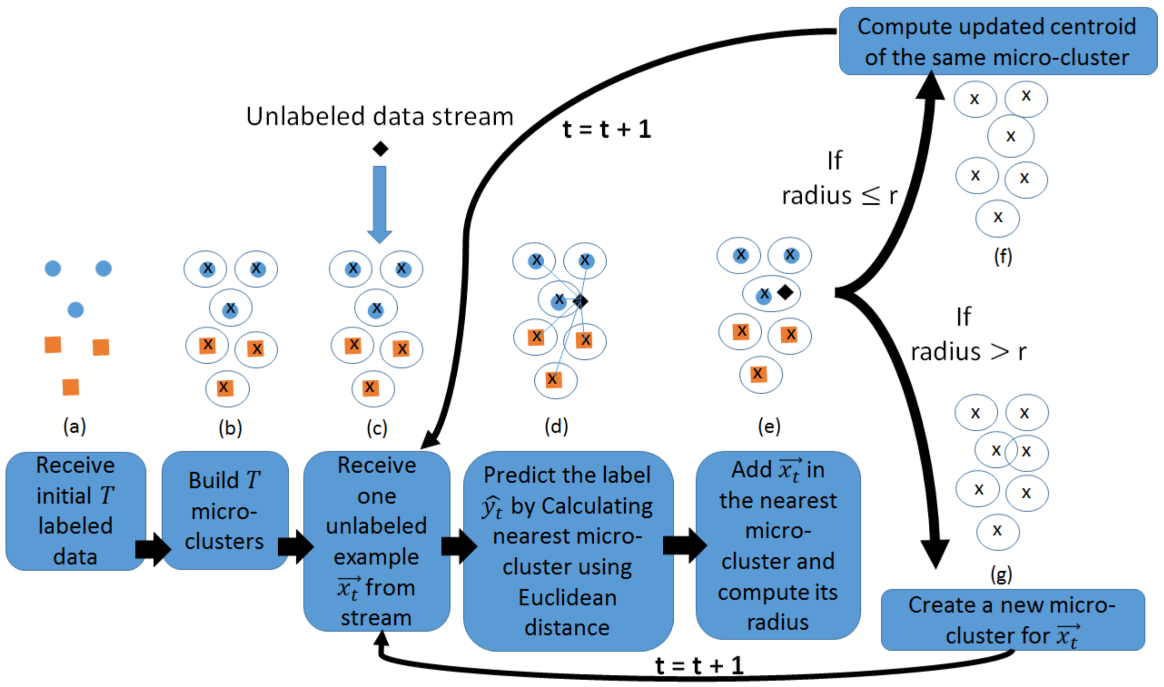

Unlabeled data stream
t = t + 1
Compute updated centroid of the same micro-cluster
If radius ≤ r
If radius > r
(a)
(b)
(c)
(d)
(e)
(f)
(g)
Receive initial $T$ labeled data
Build $T$ micro-clusters
Receive one unlabeled example $\vec{x_t}$ from stream
Predict the label $\hat{y}_t$ by Calculating nearest micro-cluster using Euclidean distance
Add $\vec{x_t}$ in the nearest micro-cluster and compute its radius
Create a new micro-cluster for $\vec{x_t}$
t = t + 1

*Figure* 6. Block diagram and Graphical Representations of MClassificaion; (a) Receive initial $T$ labeled examples (represented by blue circles and orange rectangles), (b) build $T$ micro-clusters (represented by circles around each example) from the initial data and compute their sufficient statistics (black cross represents the centroid of a particular micro-cluster), (c) start receiving one unlabeled example $\vec{x}_t$ (represented by a black diamond) from the unlabeled data stream, (d) compute nearest micro-cluster from $\vec{x}_t$ using Euclidean distance, (e) Add $\vec{x}_t$ in the nearest micro-cluster and calculate its updated radius, (f) If radius does not exceed the threshold radius $r$, update the sufficient statistics of the same micro-cluster and also compute its updated centroid which will be slightly dislocated towards the new concept, (g) If radius exceeds the threshold radius $r$, create new micro-cluster for $\vec{x}_t$ and update its corresponding statistics.

### 3.4 COMPOSE.V1 (Original COMPOSE With $\alpha$-Shape Construction)

The **COMP**acted **O**bject **S**ample **E**xtraction (COMPOSE) framework is introduced in [13] to address the extreme verification latency problem in an ILSE setting, i.e., learn drifting concepts from a streaming non stationary environment that provides only unlabeled data after initialization. The algorithm only makes an assumption of gradual/limited drift in the data, and consists of two important modules: semi-supervised learning algorithm (SSL) and the core-support extraction (CSE) module. It is an iterative procedure that uses an SSL algorithm to label the current unlabeled data using the initial labeled data. It then uses the core support extraction module to construct $\alpha$ shapes for each class and thus represent the current class conditional distribution. The $\alpha$ shape is then compacted (shrunk), creating the core support region, and instances that fall inside this region are extracted as the core supports that represent the geometric center (core support region) of each class distribution. These now-labeled instances are used as the labeled information – along with the incoming new unlabeled data – to train the SSL algorithm during the next time step. This process is repeated every time there is a new batch of data.

$\alpha$-shape can be described as a generalization of the convex hull of the dataset, where the convex hull of a dataset $X \in \mathbb{R}^d$ is the convex shape with minimum area that contains all of the observations in $X$, and can be described as the set of all possible convex combinations of the points in $X$, or

$$\{\sum_{i=1}^{|X|} a_i x_i | (\forall i : a_i \geq 0) \wedge \sum_i a_i = 1\} \tag{3.14}$$

for all possible $a_i$. The $\alpha$-shape first finds a set of adjacent d-simplices from the data forming a partitioned version of the convex hull known as the Delaunay tesselation [61] of the dataset, and then sets a threshold on the maximum radius of the circumsphere of any simplex belonging to the shape. Only those simplices and their corresponding observations form the final $\alpha$-shape whose circumsphere is not too large while remaining simplices are removed.

In order to obtain the core support region, the $\alpha$-shape is compacted (shrunk) by iteratively stripping away its outermost layer of simplices until the desired number of observations remain. The CSE procedure then returns the indices of the remaining observations as core supports to be used as labeled data by semi-supervised learning (SSL) algorithm for next time-step. COMPOSE is really a framework, thus it can make use of any Semi-Supervised Learning (SSL) algorithm that the user believes to match the characteristics of the data to improve the performance of the algorithm. The pseudocode and implementation details of the original COMPOSE version that uses $\alpha$-shape construction to extract core supports can be seen in Algorithm 4.

COMPOSE.V1 requires the following as input: i) an SSL algorithm such as cluster and label, label propagation [62], or semi-supervised support vector machines [63] with relevant free parameters; and ii) a CSE algorithm, i.e., $\alpha$ shape creation algorithm with parameters $\alpha$-shape detail level, $\alpha$, and a compaction percentage, $CP$, that represents the percentage of current labeled instances to use as core supports. The algorithm is seeded with initial labeled data $D_{init}$ in step 1. COMPOSE starts by receiving $N$ unlabeled instances $U^t$ in each time-step. The SSL algorithm is then trained using the current unlabeled and labeled instances, which returns an hypothesis $h^t$ that classifies all unlabeled instances of the current time-step in step 4. The hypothesis is then used to generate a combined set of data, $D^t$, in step 5, and the combined data for each class is used as the input for the CSE routine in step 8. The resulting core supports $CS_c$, for each class $c$, are appended to be used as current labeled data in the next time-step in step 9. The block diagram explaining different stages of the algorithm is given in Figure 7, where in Figure 7(a) COMPOSE receives initial labeled data, in Figure 7(b) it starts receiving unlabeled data (represented as black diamonds) in Figure 7(c) it classifies unlabeled data using SSL, in Figure 7(d) COM-

POSE constructs $\alpha$-shapes or boundary object around each class, in Figure 7(e) compact the boundary object to extract core supports while in Figure 7(f) extracted core supports finally become labeled data for next time-step and the process is repeated.

Algorithm 4. **COMPOSE.V1**

```
Inputs: SSL algorithm - SSL with relevant free parameters; CSE algorithm - CSE; α-
    shape detail level-α Compaction percentage - CP
 1: Receive initial labeled data $D_{init} = \{x_i; y_i\}$ ; $i = 1, ..., M$ ; $x \in X; y \in Y = \{1, ..., c\}$
    Set $L^0$ = $\{x_i^t\}$ ; initial instances
    Set $Y^0$ = $\{y_i^t\}$ ; corresponding labels of initial instances
 2: for $t = 0, 1, ....$ do
 3:     Receive unlabeled data $U^t$ = $\{x_u^t \in X , u = 1, ..., N\}$
 4:     Run SSL with $L^t$ , $Y^t$, and $U^t$
        to obtain hypothesis, $h^t$ : X → Y
 5:     Let $D^t = \{(x_l^t, y_l^t) : x \in L^t \forall l\} \cup$
        $\{(x_u^t, h^t(x_u^t)) : x \in U^t \forall u\}$
 6:     Set $L^{t+1} = \emptyset, Y^{t+1} = \emptyset$
 7:     for each class $c = 1, 2, ...., C$ do
 8:         Run CSE with $CP$ , $\alpha$ and $D_c^t$
            to extract core supports, $CS_c$
 9:         Add core supports to labeled data
            $L^{t+1} = L^{t+1} \cup CS_c$
            $Y^{t+1} = Y^{t+1} \cup \{y_u : u \in [|CS_c|], y = c\}$
10:     end for
11: end for
```

## 3.5 COMPOSE.V2 (COMPOSE With Gaussian Mixture Model (GMM) or Any Density Estimation Technique)

One of the central processes of COMPOSE is the core support extraction, where the algorithm predicts which data instances of the current environment will be useful and relevant for classification in future time-steps, where the underlying data distributions may

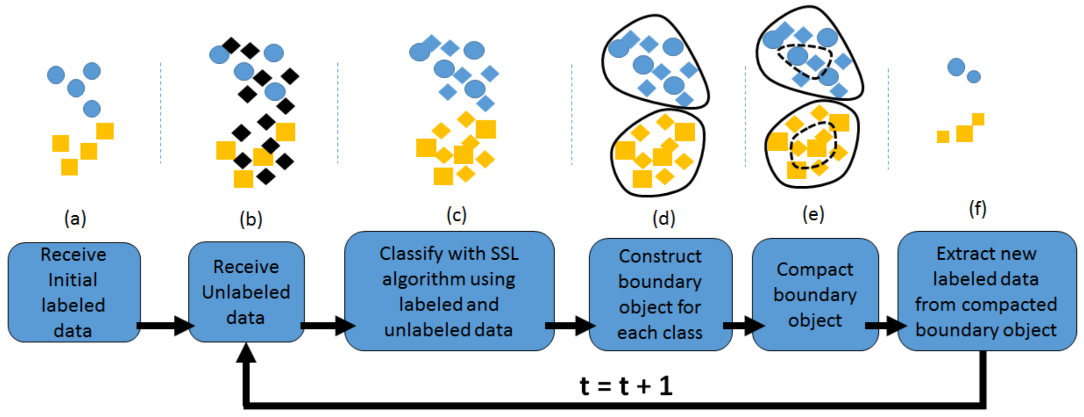

(a)
(b)
(c)
(d)
(e)
(f)
Receive Initial labeled data
Receive Unlabeled data
Classify with SSL algorithm using labeled and unlabeled data
Construct boundary object for each class
Compact boundary object
Extract new labeled data from compacted boundary object
t = t + 1

*Figure* 7. Block diagram and operation workflow of COMPOSE; (a) receive initial labeled data; (b) start receiving unlabeled data (represented as black diamonds); (c) classify unlabeled data using SSL; (d) construct $\alpha$-shapes or boundary object around each class (e); compact (shrink) the boundary object to extract core supports; (f) extracted core supports become labeled data for next time-step.

have changed. In the original version of COMPOSE, $\alpha$-shape construction is used for this process, but $\alpha$-shape construction is a computationally very expensive process, especially when the dimensionality of the data increases. This is because $\alpha$-shape construction requires Delaunay tessellation of the data, and the algorithm used for this purpose is the Quickhull algorithm [13]. This algorithm is of order $O(n^{(d+1)/2})$ where $n$ is the number of observations and $d$ is the dimensionality of the data. Hence, the algorithm is exponential in dimensionality. In order to reduce the computational complexity of the algorithm, we make use of the fact that the goal of the CSE is to extract the labeled data from each class by creating an object or shape around the data and by compacting that object. This process is essentially equivalent to density estimation.Therefore, more efficient density estimation techniques can be used. One such approach is Gaussian Mixture Model (GMM), though any other density estimation technique can also be used here such as Parzen windows or kNN. We observe that GMM are significantly more computationally efficient than $\alpha$-shape.

The Gaussian mixture model (GMM) is a probabilistic model that describes the data as a mixture of unimodal Gaussian distributions, and tries to fit $K$ Gaussians to the data $X$ where $K$ is a user specified parameter. The probability density function is the weighted sum of the $K$ Gaussians as given by the following equation,

$$p(\theta) = \sum_{k=1}^{K} \pi_k \mathcal{N}(\mu_k, \Sigma_k) \tag{3.15}$$

where $\theta$ is the set of parameters describing the entire model, $\mu_k, \Sigma_k, \pi_k$ are the mean, covariance, and mixing coefcient (i.e., prior probability) of each Gaussian component respectively. The GMM algorithm uses the expectation maximization (EM) algorithm to fit the Gaussians to the data with maximum likelihood. The EM procedure is an iterative two step procedure that runs until convergence (or a maximum number of iterations is reached). In the expectation step, the probability that each component, $\theta_k$, can be explained by observation $x_i$ is calculated as below

$$p(\theta_k|x_i) = \frac{\pi_k p(x_i|\mu_k, \Sigma_k)}{\sum_j \pi_j p(x_i|\mu_j, \Sigma_j)} \tag{3.16}$$

for all $k, i$. The maximization step then calculates new parameters for each component to maximize the likelihood

$$\mu_k = \frac{\sum_i p(\theta_k|x_i) x_i}{\sum_i p(\theta_k|x_i)} \tag{3.17}$$

$$\Sigma_k = \frac{\sum_i p(\theta_k|x_i)\theta_k|x_i(x_i - \mu_k)(x_i - \mu_k)^T}{\sum_i p(\theta_k|x_i)} \tag{3.18}$$

$$\pi_k = \frac{\sum_i p(\theta_k|x_i)}{N} \tag{3.19}$$

The major advantage of using GMMs is that GMMs are significantly more computationally efficient than $\alpha$-shapes, particularly when $d$ is large. The computational complexity of the EM procedure for GMMs is difficult to quantify, because it is an iterative procedure, but it has been shown that the E-step and the M-step are of the order $O(NKd+NK)$ and $O(2NKd)$, respectively, for each iteration, where $N$ is the number of observations, $K$ is the number of mixture components and $d$ is the dimensionality. Our results in chapter 5 confirms that the GMM approach is indeed substantially faster than constructing $\alpha$-shapes for any given dimensionality and data cardinality.

The pseudocode and implementation detail of COMPOSE.V2 is similar to COMPOSE.V1 with the difference of using GMM instead of the $\alpha$-shapes construction for core supports extraction module. The operational flow of the algorithm is illustrated in Figure 7.

# Chapter 4

# Improving Learning Concept Drift Under Extreme Verification Latency: Learning Extreme Verification Quickly With FAST COMPOSE and With Importance Weighting (LEVEL$_{\text{IW}}$)

In this chapter, we introduce two algorithms to improve learning concept drift under extreme verification latency. We first introduce FAST COMPOSE, a modification of the COMPOSE (COMpacted Object Sample Extraction) framework that works without the computationally expensive core support extraction module, and hence dramatically improves the computational efficiency of the COMPOSE framework. We then introduce the algorithm LEVEL$_{\text{IW}}$: Learning Extreme VErification Latency with Importance Weighting, which is based on importance weighting approach commonly used for domain adaptation problems. Unlike the standard importance weighting algorithms commonly used in single time-step problems of mismatched training (source) and test (target) distributions, LEVEL$_{\text{IW}}$ is designed to work in a streaming data environment. As we will discuss in Chapter 5, the primary benefits of these algorithms are not so much improved accuracy, but rather computational efficiency, in case of FAST COMPOSE, and parameter robustness in case of LEVEL$_{\text{IW}}$.

## 4.1 The Underlying Problems With Current Approaches

Addressing extreme verification latency in non-stationary environments is not a trivial task, and it is still an open research topic in machine learning. The prior work of our group (Signal Processing and Pattern Recognition Laboratory at Rowan University), resulted in the COMPOSE [13] framework – with $\alpha$-shapes and GMMs – which works remarkably well in this setting, with the only assumption of limited drift (a common assumption of all concept drift algorithms). Unlike other algorithms, COMPOSE does not make any additional assumptions with respect to the nature of the drift or the properties of the underlying distribution. However, the ability of COMPOSE to track a non-stationary environment in an extreme verification latency scenario comes at a steep cost: COMPOSE is computationally expensive. Specifically, in the original version of COMPOSE, computational complexity is exponential in dimensionality, though in the second version this was significantly reduced by replacing the most expensive portion of COMPOSE, i.e., $\alpha$-shape construction for core support extraction with any density estimation technique whether parametric or non-parametric. Specifically the parametric Gaussian Mixture Model (GMM) worked very well for this purpose [64], but we observe that density estimation itself is not a trivial task: the process is still – relatively speaking – a computationally expensive process, and it requires significant amount of data. In the case of parametric GMM, one additional assumption is that the underlying distribution can be adequately represented with a mixture of Gaussians, whose mean, variance and mixture coefficients need to be estimated. Of course, in real world situations, arbitrary shaped distributions can be approximated with GMMs provided that they are well represented with data and a sufficiently large number of $K$ Gaussians are chosen. Furthermore, the common subroutine used with GMM for density estimation is the expectation-maximization (EM) algorithm, an iterative algorithm for maximizing the likelihood and correctly estimating the parameters. However, the EM algorithm is not guaranteed to converge to find the model with global maximum likelihood, even if correct $K$ is chosen (though the local maxima found by the algorithm may be sufficient, as has been the case in our experiments). In addition, in real world scenarios, we rarely know the true value of $K$. The common solution to the optimal choice of $K$ is essentially a trial and error based cross-validation: try a range of $K$ values and choose the one that minimizes some penalty or cost function.

There are of course, non-parametric density estimation approaches as well, which attempt to estimate the density directly from the data without assuming a particular form of the underlying distribution, such as those based on histogram approximation, or Parzen windows or kernel density estimation. Each of these approaches have their own respective shortcomings. For example, histogram approximation is extremely data hungry with the amount of needed data increasing exponentially with dimensionality; the standard Parzen windows introduce spurious discontinuities to the density being estimated, and smooth kernel based kernel density estimation have several parameters to choose - depending on the kernel chosen - to which the algorithm is typically very sensitive. Furthermore, while we specifically mentioned this for histogram approach, all non-parametric density estimation approaches suffer from curse of dimensionality, and require large amounts of data for proper training.

The specific issues related to density estimation notwithstanding, there are additional concerns with respect to either implementation of COMPOSE. These concerns are independent of the specific selection of density estimation procedure, and would remain with any density estimation approach, however efficient or effective it may be. Originally, it was thought that the core support extraction routine would extract a region that has a high probability of overlap with drifted unlabeled data at the next timestep. However, because the core support extraction routine is executed using the hypothesis at timestep $t$ along with labeled data at time-step $t$ (which themselves are the core supports extracted at timestep $t-1$), the most dense region will lie in the overlap of the labeled and unlabeled data from time-step $t$ labeled by the hypothesis at timestep $t$. Therefore, the core supports at timestep $t$ will be further away from the unlabeled data at timestep $t+1$, and in turn the CSE routine will extract a region that has a lower probability of overlap with the drifted data. Figure 8 illustrates the above described scenario, where the blue instances inside the blue circle represent the labeled data at time-step $t$ (i.e., the core supports extracted in time-step $t-1$) denoted as $CS^{t-1}/L^t$; the middle gray circle represents the distribution from which unlabeled data are drawn at time-step $t$, with the unlabeled data themselves indicated by the black diamonds and denoted as $U^t$. The right gray circle represents the distribution from which unlabeled data are drawn at time-step $t+1$, with the unlabeled data indicated with the green stars and denoted as $U^{t+1}$. Finally, the instances in the pink circle represent the core supports extracted in time-step $t$ to be used as labeled information for time-step $t+1$. Under the core support extraction process used in COMPOSE, we see that the unlabeled data (green stars) at time-step $t+1$ are further away from the core supports (inside the pink circle) than they are from all of the unlabeled data as a whole. This insight instructs us that using all of the data (once labeled by the SSL algorithm) may be more effective then the core supports used under the original COMPOSE algorithm. Furthermore, since the core support extraction – as accomplished either by $\alpha-$shape creation or density estimation – is the computational bottleneck of the algorithm, any improvement on the core support extraction has the dual benefit of increasing computational efficiency, as well as posisbly improving classification performance.

Before, we describe how the modified COMPOSE is structured and how it addresses the aforementioned issues, it is perhaps worthwhile to identify and summarize

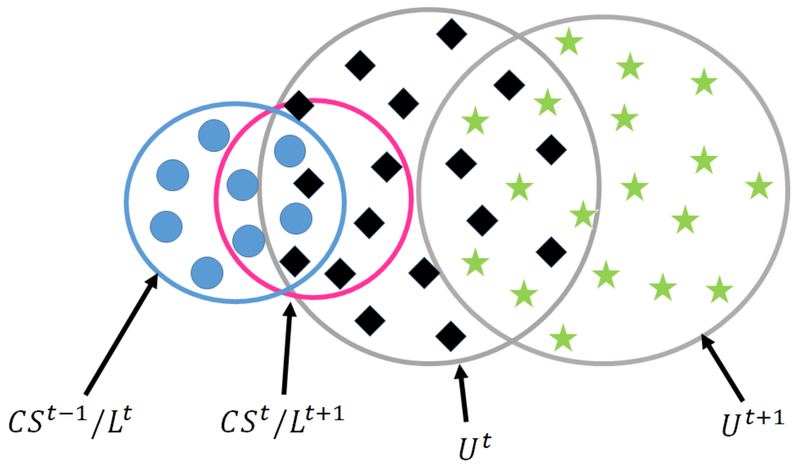


*Figure* 8. Example depicting core support extraction procedure in COMPOSE; new unlabeled instances at time $t+1$ are further away from the core supports selected at time $t$, then they are from the centroid of all unlabeled data obtained at time $t$.

the shortcomings in various algorithms in the extreme learning verification applications. While COMPOSE has the above-described issues, other algorithms are not without their own shortcomings, typically more serious than those of COMPOSE. APT, for example, [14] is extremely computationally expensive (so much so that it becomes computationally prohibitive to run it even for modestly dimensioned data), and it makes some unrealistic assumptions, such as all modes of the distribution representing the data being present at the initialization, or the drift being structured, systematic and piecewise linear, etc. SCARGC [15] is computationally efficient, but it is heavily dependent on the clustering phase, which requires some prior knowledge about the data – information is rarely available in real world scenarios. MClassification [16] requires only one parameter, but since it is an online algorithm, (i.e., processing instances one example at a time), it is also computationally prohibitive. MClassification also appears to be not a suitable algorithm for those datasets where positions and parameters of the data distributions representing different classes suddenly change along with the between class overlap over time. Clearly, there is room for improvement.

## 4.2 Learning Extreme Verification Latency Quickly: FAST COMPOSE

Our modification of COMPOSE with respect to core support extraction module is based on the following observation. Originally a significant overlap of class conditional distributions between consecutive time steps was thought to be the working definition of *gradual / limited drift*, and hence a necessary condition for COMPOSE to work. However, Sarnelle et al. showed in [65] that COMPOSE can work equally well for scenarios even when there is no overlap of distributions in consecutive time steps, as long as the distance between the unlabeled data with core supports of a given class is less than the distance from the nearest core supports of any other opposing class. We refer to this condition as *limited drift*, and now distinguish it from *gradual drift* that does require an overlap of distributions in subsequent time-steps. As a result, we show that the condition of significant overlap (or *gradual drift*) can be eliminated, and replaced with the more relaxed condition of *limited drift*. We observe that for cases where there is no significant overlap, core support extraction procedure has very little impact on accuracy because it does not change centroids in any considerable amount, and clustering based SSL algorithm can easily track the drifting distributions using nearest centroids.

Additionally, as described above, the density estimation procedure is impractical for high dimensional data due to its computational complexity. Taken together then, an obvious questions that comes to mind is whether the density estimation based core support extraction is needed at all. To answer this question, we removed the core support extraction procedure of COMPOSE entirely, and all instances labeled by the semi-supervised algorithm are then used as “core supports,” i.e., the most representative instances for the future time-steps. We call this modified version of the algorithm FAST COMPOSE [66].

The pseudocode and implementation details of FAST COMPOSE are shown in Algorithm 5. FAST COMPOSE only requires an SSL algorithm with its relevant free parameters as an input. The algorithm begins by receiving $M$ initially labeled instances, $L^0$, and corresponding labels $Y^0$, of $C$ classes in step 1. The algorithm then receives a new set of $N$ unlabeled instances $U^t$. The SSL algorithm is then executed given the current unlabeled and labeled instances to receive the hypothesis $h^t$ of the current time-step in step 4. The hypothesis is then used to label the data for the next time-step as shown in steps 5 - 8 of Algorithm 5.

Algorithm 5. **FAST COMPOSE**

```
Input: SSL algorithm - SSL with relevant free parameters
 1: Receive labeled data
    $L^0 = \{x_l^t \in X\}$ ,
    $Y^0 = \{y_l^t \in Y = \{1, \ldots, C\}, l = 1, \ldots, M\}$
 2: for $t = 0, 1, ....$ do
 3:     Receive unlabeled data $U^t = \{x_u^t \in X , u = 1, ..., N\}$
 4:     Run SSL with $L^t$ , $Y^t$, and $U^t$
        to obtain hypothesis, $h^t : X \rightarrow Y$
 5:     Let $D^t = \{(x_u^t, h^t(x_u^t)) : x \in U^t \forall u\}$
 6:     Set $L^{t+1} = \emptyset, Y^{t+1} = \emptyset$
 7:     for each class $c = 1, 2, ...., C$ do
 8:         $CS_c = \{x : x \in D_c^t\}$ , and add to labeled data for next time-step
            $L^{t+1} = L^{t+1} \cup CS_c$
            $Y^{t+1} = Y^{t+1} \cup \{y_u : u \in [|CS_c|], y = c\}$
 9:     end for
10: end for
```

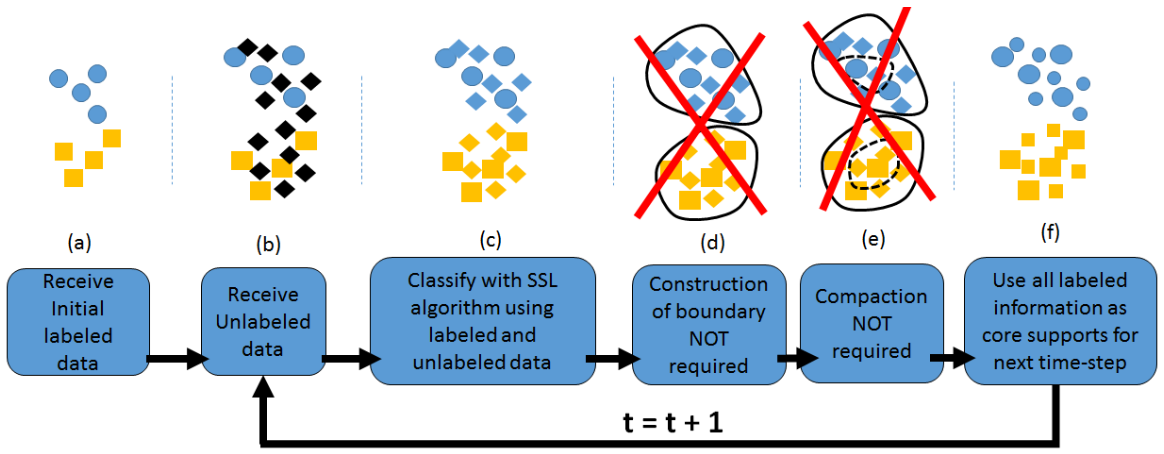

(a)
(b)
(c)
(d)
(e)
(f)
Receive Initial labeled data
Receive Unlabeled data
Classify with SSL algorithm using labeled and unlabeled data
Construction of boundary NOT required
Compaction NOT required
Use all labeled information as core supports for next time-step
t = t + 1

*Figure* 9. Block diagram and graphical illustration of FAST COMPOSE; (a) Receive initial labeled data represented as colored squares and circles, (b) start receiving unlabeled data (represented as black diamonds), (c) classify unlabeled data using SSL, (d) construction of boundary object is no longer required; (e) nor is compaction of that object; (f) use all the labeled data as core supports to be used for next time-step.

The graphical representation of FAST COMPOSE illustrating its different stages can be seen in Figure 9. Initial labeled data (indicated in yellow and blue) are received in Figure 9(a), and the algorithm starts receiving unlabeled data (indicated in black diamonds) in 9(b). SSL algorithm is used to classify unlabeled data in 9(c). The construction of boundary object and compaction of that object, as done in two previous versions of COMPOSE are no longer needed by FAST COMPOSE as shown in 9(d) and 9(e), respectively. Finally in 9(f) the algorithm uses all data just labeled by SSL algorithm as core supports for the next time step and the process is repeated.

## 4.3 LEVEL$_{IW}$: Learning Extreme VErification Latency With Importance Weighting

The primary shortcoming of the COMPOSE algorithm, and hence the goal in developing its third version was well known and motivated: the density estimation based core support extraction is computationally expensive, and we were investigating alternative approaches to reduce the algorithm's computational burden by removing the computational bottleneck. As shown in Chapter 5 and demonstrated by large number of experiments, the new version of COMPOSE did indeed met our goals, and is currently the fastest running and hence computationally most efficient algorithm for learning in nonstationary environments under extreme verification latency, hence earning its name FAST COMPOSE.

The second approach we explored was motivated more by academic curiosity than having a specific improvement on a particular metric, and was inspired by a simple observation: there is a fundamental similarity between the domain adaptation problem and the problem of learning in a nonstationary environment. In both cases, there is a change in distribution between the two consecutive time steps where the environment is provided with additional data. In case of domain adaptation, the problem is a mismatch of distributions between the source (training) and target(testing) domains, where there is often little or no labeled data from the target domain. While there is no streaming data in a domain adaptation problem, this setting is similar to any two consecutive iterations of the learning in a nonstationary environment with extreme verification latency (EVL). Therefore, a logical question to ask is whether an algorithm used for domain adaptation can also be used in EVL setting, by iteratively repeating the domain adaptation algorithm every time there is new data.

We observe that importance weighting based domain adaptation used for covariate shift and concept drift problems are related, though algorithms for each make different assumptions. Concept drift problems typically assume at least a gradual (or at least limited) drift assumption, but do not require stationary posteriors or shared support. We explore whether and when well-established, computationally efficient importance weighting based domain adaptation approaches can be used for concept drift problems associated with extreme verification latency. We show that the answer to this question is affirmative, when indeed the original importance weighting (covariate shift) assumptions are satisfied, i.e., the class conditional distributions at consecutive time steps share support, and posterior distributions do not change.

More specifically, recall that COMPOSE originally assumed a significant distribution overlap at consecutive time steps, allowing instances lying in the center of the feature space to be used as the most representative labeled instances from current time step to help label the new data at the next time step. Such an assumption is also inherent in importance weighting based domain adaptation, but only for a single time step with mismatched train and test data distributions. We therefore explore importance weighting not for a single time step matching training / test distributions, but rather matching distributions between two consecutive time steps, and estimate the posterior distribution of the unlabeled data using importance weighted least squares probabilistic classifier (IWLSPC), as explained in detail later in this chapter. The estimated labels are then iteratively used as the training data for the next time step. We call this algorithm LEVEL$_{\text{IW}}$: $\underline{\text{L}}$earning $\underline{\text{E}}$xtreme $\underline{\text{VE}}$rification $\underline{\text{L}}$atency with $\underline{\text{I}}$mportance $\underline{\text{W}}$eighting.

To explain how we use importance sampling based domain adaptation in the EVL setting, we first describe importance weighted least square probabilistic classifier, as first proposed in [67].

### 4.3.1 Importance weighted least-squares probabilistic classifier.

One of the most important components of the LEVEL$_{\text{IW}}$ algorithm is the importance weighted least-squares probabilistic classifier (**IWLSPC**), which combines a probabilistic classification method, called least-squares probabilistic classifier, with the covariate shift adaptation technique. As described in more detail in [67], where this approach was first proposed, probabilistic classification is used to estimate the true class-posterior probability $p(y|\mathbf{x})$, modeled through the following linear model

$$p(y|\mathbf{x}, \boldsymbol{\theta}_y) = \sum_n \theta_{y,n} K(\mathbf{x}, \mathbf{x}_{te,n}) \tag{4.1}$$

where $n$ is an index on number of instances, $\mathbf{x}_{te,n}$ is the $n^{th}$ test instance, $\boldsymbol{\theta}_y = (\theta_{y,1}, \ldots, \theta_{y,n})$ is the parameter vector of linear model, and $K(\mathbf{x}, \mathbf{x}_{te})$ is a Kernel function, typically the Gaussian kernel

$$K(\mathbf{x}, \mathbf{x}_{te,n}) = \exp\,(-\frac{||\mathbf{x} - \mathbf{x}_{te,n}||^2}{2\sigma^2}) \tag{4.2}$$

with kernel width $\sigma$ serving as the first free parameter for IWLSPC. The parameter vector $\boldsymbol{\theta}_y$ is determined by minimizing the squared error $J_y(\boldsymbol{\theta}_y)$ through quadratic programming

$$\begin{aligned} J_y(\boldsymbol{\theta}_y) &= \frac{1}{2}\int (p(y|\mathbf{x};\boldsymbol{\theta}_y) - p(y|\mathbf{x}))^2 p_{te}(\mathbf{x})d\mathbf{x} \\ &= \frac{1}{2}\boldsymbol{\theta}_y^T \mathbf{Q} \boldsymbol{\theta}_y - \mathbf{q}_y^T \boldsymbol{\theta}_y + \frac{\lambda}{2}\boldsymbol{\theta}_y^T \boldsymbol{\theta}_y \end{aligned} \tag{4.3}$$

where the last term is a regularization term to minimize over-fitting through the algorithms's second free parameter $\lambda$, and where $\mathbf{Q}$ – an $n_{te} \times n_{te}$ matrix – and $\mathbf{q}_y = (q_{y,1}, \ldots, q_{y,n_{te}})$ are approximated using the adaptive importance sampling technique, through the *importance weight* defined as

$$w(\mathbf{x}) = \frac{p_{te}(\mathbf{x})}{p_{tr}(\mathbf{x})} \tag{4.4}$$

The quantities $\mathbf{Q}$ and $\mathbf{q}_y$ are then obtained as follows

$$Q_{n,n'} = \int K(\mathbf{x}, \mathbf{x}_{te,n}) K(\mathbf{x}, \mathbf{x}_{te,n'}) p_{tr}(\mathbf{x}) w(\mathbf{x}) d\mathbf{x} \tag{4.5}$$

$$q_{y,n} = p(y) \int K(\mathbf{x}, \mathbf{x}_{te,n}) p_{tr}(\mathbf{x}|y) w(\mathbf{x}) d\mathbf{x} \tag{4.6}$$

where $p_{tr}(\mathbf{x}|y)$ denotes the training input density for class $y$. Based on the above expressions, $Q$ and $q_y$ are approximated using the training samples $\{\boldsymbol{x_{tr,n}}, \boldsymbol{y_{tr,n}}\}_{n=1}^{N_{tr}}$ as follows

$$\hat{Q}_{n,n'} = \frac{1}{N_{tr}} \sum_{n''=1}^{N_{tr}} K(\mathbf{x_{tr,n''}}, \mathbf{x_{te,n'}}) K(\mathbf{x_{tr,n''}}, \mathbf{x_{te,n'}}) w(\mathbf{x_{tr,n''}}) \tag{4.7}$$

$$\hat{q}_{y,n} = \frac{1}{N_{tr}} \sum_{n';y_{tr,n'}=y} K(\mathbf{x_{tr,n'}}, \mathbf{x_{te,n}}) w(\mathbf{x_{tr,n'}}) \tag{4.8}$$

where, the class prior probability $p(y)$ was estimated by $N_{tr,y}/N_{tr}$, and $N_{tr,y}$ denotes the number of training samples with label $y$. The following optimization problem is consequently obtained to solve for $\boldsymbol{\theta}_y$, which in turn is used to determine the class-posterior probability $p(y|x;\boldsymbol{\theta}_y)$ through Equation 4.1.

$$\hat{\boldsymbol{\theta}}_y = \underset{\theta_y}{\operatorname{argmin}} [\frac{1}{2} \boldsymbol{\theta}_y^T \hat{\mathbf{Q}} \boldsymbol{\theta}_y - \hat{\mathbf{q}}_y^T \boldsymbol{\theta}_y + \frac{\lambda}{2} \boldsymbol{\theta}_y^T \boldsymbol{\theta}_y] \tag{4.9}$$

Given a test instance $\mathbf{x}_{te}$, the class label $y_{te}$ is finally estimated as

$$\hat{y}_{te} = \underset{y}{\operatorname{argmax}}\, p(y|\mathbf{x}_{te}; \boldsymbol{\theta}_y). \tag{4.10}$$

The critical parameter in model selection for IWLSPC is kernel width $\sigma$, which is obtained through importance weighted cross validation (IWCV) [68] (as described in IWLSPC's original description in [67]) and it is updated each time step separately. Cross-validation (CV) is a standard procedure for model (or parameter) selection, but under covariate shift, ordinary CV is highly biased due to differing distributions. Therefore modified version of the ordinary CV is proposed in [68] known as *importance weighted cross validation (IWCV)*, which basically weighs the validation error in the ordinary CV procedure according to the importance. The pseudocode of the original IWCV is given in Algorithm 6.

Algorithm 6. **IWCV**

```
Inputs: Training data $D = \{x_i, y_i\}; i = 1, .., N_{tr}$ ; number of folds $K$ in $K$-fold loss
1: Divide the training data into $K$ disjoint non-empty subsets $\{D_k\}_{k=1}^{K}$
2: Build a hypothesis $h(x_i)$ from $D$ - $D_k$ (i.e. without $D_k$)
3: Compute the mean discrepancy between the true output value $y_i$ and its estimate obtained using hypothesis $h(x_i)$ i.e. $loss(h(x_i), y_i)$
4: Compute $K$-fold IWCV estimate of the generalization error as $\hat{G}_{IWCV} = \frac{1}{K}\sum_{k=1}^{K}\frac{1}{|D_k|}\sum_{(x,y)\in D_k} w(x)loss(h(x), y)$
```

The importance weights in Equation 4.4 are estimated through *unconstrained least squares importance fitting* (uLSIF) [69] as done in [67]. uLSIF formulates the direct importance estimation problem as a least-squares function fitting problem. A linear model is used to model the importance ratio as given below

$$\hat{w}(x) = \sum_{n} \alpha_n K(x, x_{te,n}) \tag{4.11}$$

where, $\alpha = (\alpha_1, \dots, \alpha_n)$ is the parameter vector of linear model to be learned from data samples, and $K(x, x_{te,n})$ is a Gaussian kernel function given in Equation 4.2. The parameters $\alpha_n$ for the linear model are determined by minimizing the squared error $J_0$ between the actual importance $w(x)$ and modeled importance $\hat{w}(x)$ through quadratic programming.

$$\begin{aligned} J_0(\alpha) &= \frac{1}{2}\int (\hat{w}(x)) - w(x))^2 p_{tr}(x)dx \\ &= \frac{1}{2}\int \hat{w}(x)^2 p_{tr}(x)dx - \int \hat{w}(x)w(x)p_{tr}(x)dx + \frac{1}{2}\int w(x)^2 p_{tr}(x)dx \\ &= \frac{1}{2}\int \hat{w}(x)^2 p_{tr}(x)dx - \int \hat{w}(x)p_{te}(x)dx + \frac{1}{2}\int w(x)^2 p_{tr}(x)dx \end{aligned} \tag{4.12}$$

The last term in Equation 4.12 is constant and therefore can be safely ignored for the purposes of minimizing the squared error objective function. Let us denote the first two terms by $J$ as given below

$$\begin{aligned} J(\alpha) &= \frac{1}{2}\int \hat{w}(x)^2 p_{tr}(x)dx - \int \hat{w}(x)p_{te}(x)dx \\ &= \frac{1}{2}\sum_{n,n'} \alpha_n\alpha_{n'}(\int K(x, x_{te,n})K(x, x_{te,n'})p_{tr}(x)dx) - \sum_{n} \alpha_n(\int K(x, x_{te,n})p_{te}(x)dx) \\ &= \frac{1}{2}\alpha^T H\alpha - h^T\alpha \end{aligned} \tag{4.13}$$

where, $T$ denotes the transpose, $H$ is the $n \times n$ matrix with the $(n, n')^{th}$ element denoted as

$$H = \int K(x, x_{te,n})K(x, x_{te,n'})p_{tr}(x)dx \tag{4.14}$$

and $h$ is the $n$-dimensional vector with the $n^{th}$ element denoted as

$$h = \int K(x, x_{te,n})p_{te}(x)dx \tag{4.15}$$

The equations 4.14 and 4.15 can be approximated using simple mean across the samples in both source and target distributions as given below.

$$\hat{H} = \frac{1}{N_{tr}}\sum_{i=1}^{N_{tr}} K(x_{tr,i}, x_{te,n})K(x_{tr,i}, x_{te,n'}) \tag{4.16}$$

and $h$ is the $n$-dimensional vector with the $n^{th}$ element denoted as

$$\hat{h} = \frac{1}{N_{te}}K(x_{te,j}, x_{te,n}) \tag{4.17}$$

where $N_{tr}$ and $N_{te}$ represent the total number of training and test examples respectively. The solution to an optimization problem is obtained in order to find the values of the parameter vector $\alpha$. These parameter vector values are ultimately used in Equation 4.11 to estimate importance values. The pseudocode of uLSIF algorithm is given in Algorithm 7.

Algorithm 7. **uLSIF**

```
Inputs: Training data $x_{tr}$; Test data $x_{te}$; Gaussian Kernel with suitable bandwidth
1: Compute value of $H$ using equation 4.14
2: Compute value of $h$ using equation 4.15
3: Estimate parameter $\alpha$ by minimizing squared error $J(\alpha)$ as defined in Equation 4.13
4: Use $\alpha$, and the Gaussian Kernel function to compute importance ratio as defined in
   Equation 4.11.
```

Here, we suitably modified IWLSPC – originally proposed for only single time step problems, where it was used to match the divergence in the training (source) and test (target) distributions on a non-streaming data application – and extended its use to problems in which i) data arrive in a continuous streaming fashion, where concept drift is occurring possibly at every time step, and perhaps more importantly ii) data arrive with extreme verification latency. The pseudocode of the original IWLSPC is given in Algorithm 8

Algorithm 8. **IWLSPC**

```
Inputs: Subroutines for unconstrained least squares importance fitting uLSIF - Importance weighted cross validation IWCV
1: Receive training data $x_{tr}$
2: Receive test data $x_{te}$
3: Run uLSIF to estimate importance weights by minimizing the squared error between actual importance and modeled importance using equation 4.13
4: Run IWCV to weigh the validation error in estimating Gaussian kernel width $\sigma$ according to the importance
5: Compute Gaussian Kernel Function using $\sigma$ as defined in Equation 4.2
6: Estimate parameter $\boldsymbol{\theta}_y$ by minimizing squared error $J_y(\boldsymbol{\theta}_y)$ as defined in Equation 4.3
7: Use $\boldsymbol{\theta}_y$, and the Gaussian Kernel function to compute posterior probability as defined in Equation 4.1.
```

### 4.3.2 LEVEL$_{\mathbf{IW}}$.

The common assumption made by most concept drift algorithms is that the data drift gradually between two time steps that allows class-conditional distributions of any class to possess *significant overlap* at each consecutive time steps; this *significant overlap* is also the motivation behind using the importance weighting approach as such overlap is likely to result in satisfying the two important assumptions of importance weighting approaches: i) shared support of class-conditional distributions at two consecutive time steps; and ii) posterior distribution for each class remains the same (or at least, changes very little). Recall that importance weighting, as used in domain adaptation, is intended for a single time step scenario with mismatched training and test datasets, whereas we need an algorithm that is intended to be used in streaming datasets with nonstationary distributions. Therefore, iteratively applying importance weighting, where each consecutive time step serve as the traditional source and target datasets, allows us to cast the importance weighting in a streaming data environment, with the caveat that we are in fact

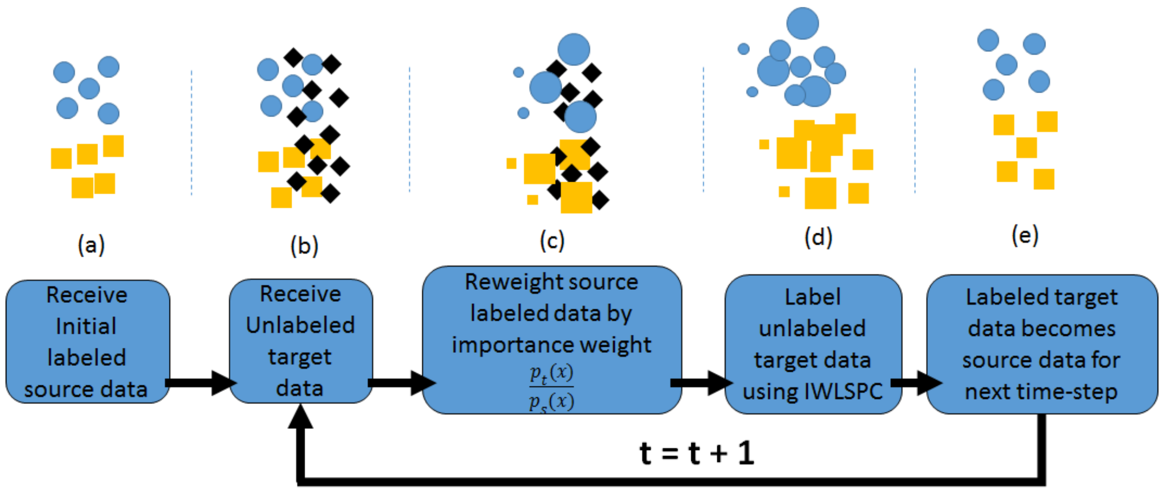

(a)
(b)
(c)
(d)
(e)
Receive Initial labeled source data
Receive Unlabeled target data
Reweight source labeled data by importance weight $\frac{p_t(x)}{p_s(x)}$
Label unlabeled target data using IWLSPC
Labeled target data becomes source data for next time-step
t = t + 1

*Figure* 10. Block diagram and graphical representation of of LEVEL$_{\text{IW}}$; (a) Receive initially labeled source data, (b) receive unlabeled target data, (c) re-weigh source data using importance weighting, (d) use IWLSPC to label unlabeled target data, (e) labeled target data becomes source data for next time step

working in an extreme verification latency environment. Hence, we name our approach Learning Extreme VErification Latency with Importance Weighting: LEVEL$_{\text{IW}}$.

The pseudocode and implementation details of this approach are described below and summarized in Algorithm 9, whereas the graphical representation of this approach showing its different stages is shown in Figure 10, where importance weight $p_t(x)/p_s(x)$ is used to re-weigh the labeled source instances as shown in Figure 10(c), while this re-weighted instances are used to label the unlabeled target instances in Figure 10(d) using IWLSPC. Finally the labeled target data become the source data for the next time step and the process is repeated as shown in Figure 10(e).

LEVEL$_{\text{IW}}$ takes advantage of the importance weighted least squares probabilistic classifier (IWLSPC) as a subroutine [67], and hence serves as a wrapper approach.

Algorithm 9. **LEVEL$_{\mathbf{IW}}$**

```
Inputs: Importance weighted least squares probabilistic classifier - IWLSPC; Kernel
    bandwidth value $\sigma$
1: At $t = 0$, receive initial data $\mathbf{x} \in X$ and the corresponding labels $y \in Y = 1, \ldots, C$.
   Set $\mathbf{x}_{te}^{t=0} = \mathbf{x}$
   Set $y_{te}^{t=0} = y$
2: for $t = 1, ....,$ do
3:     Receive new unlabeled test data $\mathbf{x}_{te}^{t} \in X$
4:     Set $\mathbf{x}_{tr}^{t} = \mathbf{x}_{te}^{t-1}$
5:     Set $y_{tr}^{t} = y_{te}^{t-1}$
6:     Call IWLSPC with $\mathbf{x}_{tr}^{t}, \mathbf{x}_{te}^{t}, y_{tr}^{t}$, and $\sigma$ to estimate $y_{te}^{t}$
7: end for
```

Initially, at $t = 0$, LEVEL$_{\text{IW}}$ receives data $\mathbf{x}$ with their corresponding labels $y$, initializes the test data $\mathbf{x}_{te}^{t=0}$ to initial data $\mathbf{x}$ received, and sets their corresponding labels $y_{te}^{t=0}$ equal to the initial labels $y$. Then, the algorithm iteratively processes the data, such that at each time step $t$, a new unlabeled test dataset $\mathbf{x}_{te}^{t}$ is first received, the previously unlabeled test data from previous time step $\mathbf{x}_{te}^{t-1}$, which is now labeled by the IWLSPC subroutine, becomes the labeled training data $\mathbf{x}_{tr}^{t}$ for the current time step, and similarly the labels $y_{te}^{t-1}$ obtained by IWLSPC during the previous time step become the labels of the current training data $\mathbf{x}_{tr}^{t}$. The training data at the current time step $\mathbf{x}_{tr}^{t}$, the corresponding label information at the current time step $y_{tr}^{t}$, the kernel bandwidth value $\sigma$ and the unlabeled test data at the current time step $\mathbf{x}_{te}^{t}$ are then passed onto the IWLSPC algorithm, which predicts the labels $y_{te}^{t}$ for the test unlabeled data. The entire process is then iteratively repeated.

# Chapter 5

# Experiments, Results and Comprehensive Analysis of Algorithms for Learning Under Extreme Verification Latency

Fifteen synthetic datasets and one real dataset constituted the primary test bench used in the evaluation and comparison of the algorithms that are designed to handle extreme verification latency. These datasets were selected because they are also used as a benchmark by the authors of the other algorithms. Some of these datasets (indicated by an asterix in Table 1) were originally provided by us in our prior works of [5] and [13], and others are provided by the authors of SCARGC in [15], and then provided at one convenient web site (https://sites.google.com/site/nonstationaryarchive/) for machine learning community.

Brief descriptions of these datasets are provided below, whose important characteristics are listed in Table 1.

1. $1CDT$ represents a 2-class, bi-dimensional dataset, where one class ($1C$) is diagonally translating ($DT$) over the other class;

2. $2CDT$ represents a 2-class, bi-dimensional dataset, where two classes ($2C$) are diagonally translating ($DT$) through each other;

3. $1CHT$ represents a 2-class, bi-dimensional dataset, where one class ($1C$) is horizontally translating ($HT$);

4. $2CHT$ represents a 2-class, bi-dimensional dataset, where two classes ($2C$) are horizontally translating ($HT$);

5. $4CR$ represents a 4-class, bi-dimensional dataset, where four classes $(4C)$ are rotating $(R)$ with complete separation among them;

6. $4CRE - V2$ represents a 4-class, bi-dimensional dataset, where four classes $(4C)$ are rotating $(R)$ with expansion $(E)$ causing the classes to overlap at some points;

7. $5CVT$ represents a 5-class, bi-dimensional dataset, where five classes $(5C)$ are vertically translating $(VT)$;

8. $1Csurr$ represents a 2-class, bi-dimensional dataset where one class $(1C)$ circumnavigates (surrounds) $(surr)$ the other class;

9. $4CE1CF$ represents a 5-class, bi-dimensional dataset, where four classes $(4C)$ are expanding $(E)$ while the remaining one class $(1C)$ stays fixed $(F)$;

10. $UG\_2C\_2D$ represents a 2-class, bi-dimensional unimodal Gaussian dataset;

11. $MG\_2C\_2D$ represents a 2-class, bi-dimensional Multi-modal Gaussian dataset;

12. $GEARS\_2C\_2D$ represents a 2-class, bi-dimensional dataset, where two gears representing two different classes are rotating;

13. $FG\_2C\_2D$ represents a two bi-dimensional classes as four Gaussians;

14. $UG\_2C\_3D$ represents a 2-class, three dimensional unimodal Gaussian dataset;

15. $UG\_2C\_5D$ represents a 2-class, five dimensional unimodal Gaussian dataset; and finally the real-world dataset

Table 1

*Dataset descriptions*

| Datasets | # of classes | # of features | Cardinality | Drift interval | Class Overlap |
|---|---|---|---|---|---|
| 1CDT | 2 | 2 | 16000 | 400 | no |
| 1CHT | 2 | 2 | 16000 | 400 | no |
| 1CSurr | 2 | 2 | 55283 | 600 | yes |
| 2CDT | 2 | 2 | 16000 | 400 | yes |
| 2CHT | 2 | 2 | 16000 | 400 | yes |
| 4CE1CF | 5 | 2 | 173250 | 750 | no |
| 4CR | 4 | 2 | 144400 | 400 | no |
| 4CRE-V2 | 4 | 2 | 183000 | 1000 | yes |
| 5CVT | 5 | 2 | 40000 | 1000 | yes |
| FG_2C_2D* | 2 | 2 | 200000 | 2000 | yes |
| GEARS_2C_2D | 2 | 2 | 200000 | 2000 | no |
| MG_2C_2D* | 2 | 2 | 200000 | 2000 | yes |
| UG_2C_2D* | 2 | 2 | 100000 | 1000 | yes |
| UG_2C_3D* | 2 | 3 | 200000 | 2000 | yes |
| UG_2C_5D* | 2 | 5 | 200000 | 2000 | yes |
| keystroke | 4 | 10 | 1600 | 200 | DNK |

16. *keystroke* dataset represents a 10-dimensional, four-class dataset with a complex drift scenario, contains information from the keystrokes dynamics obtained from the users who type a fixed password, *.tie5Roan1*, followed by the *Enter* key 400 times in 8 sessions performed on different days. The task of the classifier is to classify each one of four different users over time according to their typing profile.

The synthetic datasets are deliberately chosen to be two dimentional so that their drift can be visualized. Evolving behavior of different class distributions showing the progress of drift for datasets $1CDT$, $1CHT$, $2CDT$, and $2CHT$ are illustrated in Figure 11 a~d using three snapshots in time. The black arrows indicate the direction in which each distribution drift.

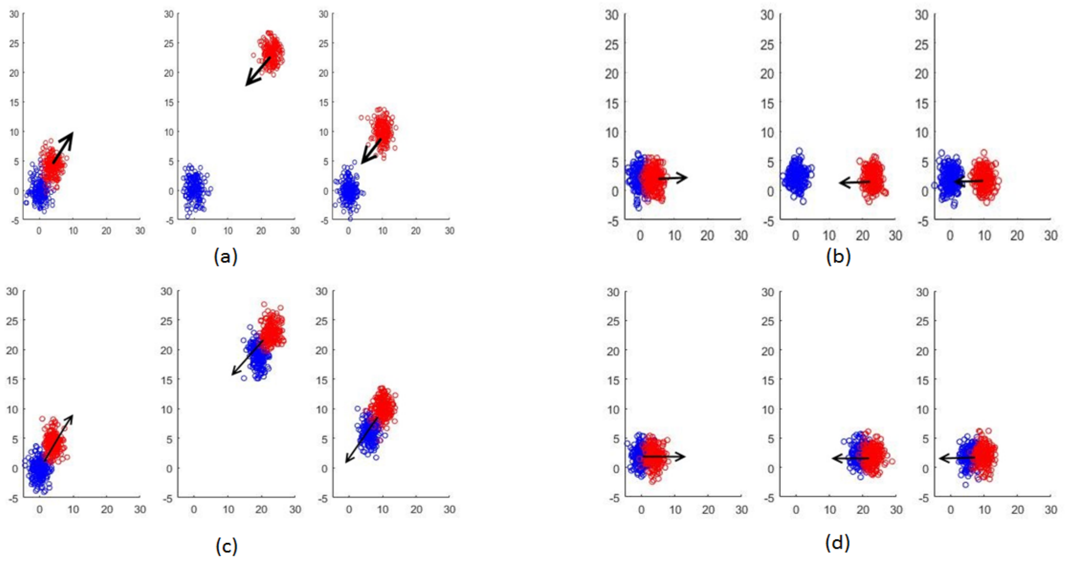


*Figure* 11. Progress of drift for 1CDT, 1CHT, 2CDT, and 2CHT datasets; (a) 1CDT data for three different snapshots with black arrow representing the drift direction of red class throughout the experiment, (b) 1CHT data for three different snapshots, (c) 2CDT data for three different snapshots, (d) 2CHT data for three different snapshots.

Figure 14 a~d illustrate the similar behavior for datasets $4CR$, $4CRE-V2$, $1Csurr$ and $4CE1CF$, respectively. Figure 12, Figure 13, Figure 15, Figure 18, Figure 20, and Figure 22 show the drift progress for datasets $GEARS\_2C\_2D$, $UG\_2C\_3D$, $5CVT$, $FG\_2C\_2D$, $MG\_2C\_2D$ and $UG\_2C\_2D$ respectively.

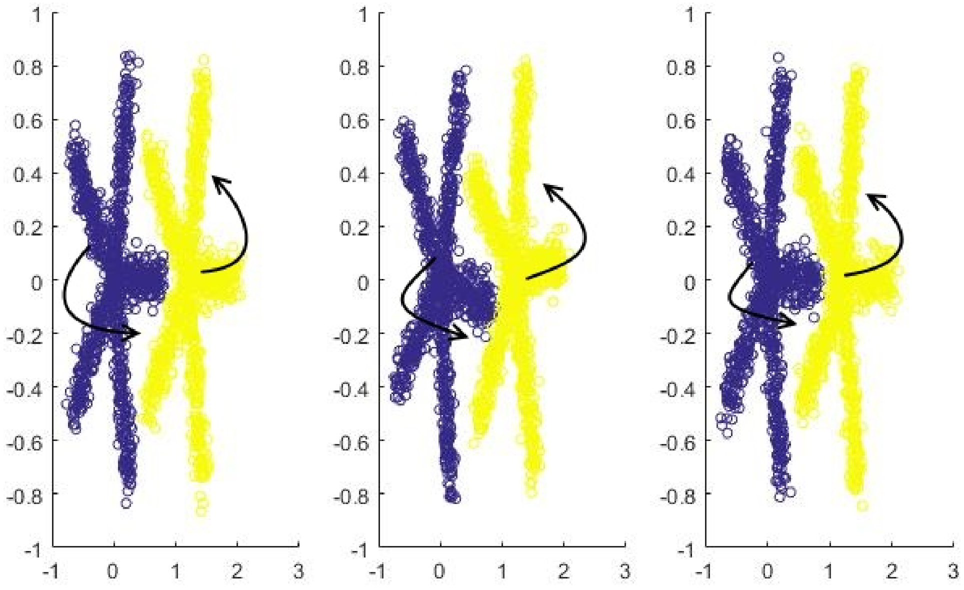


*Figure* 12. Three different snapshots of GEARS_2C_2D data

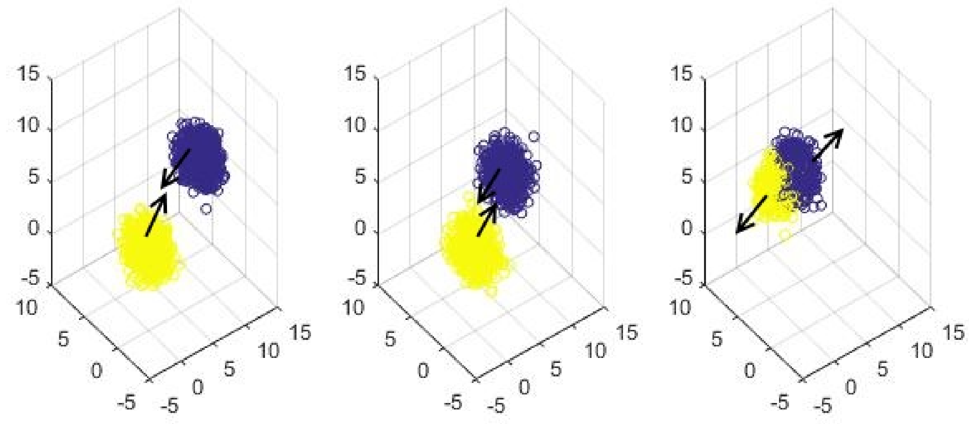


*Figure* 13. Three different snapshots of UG_2C_3D data

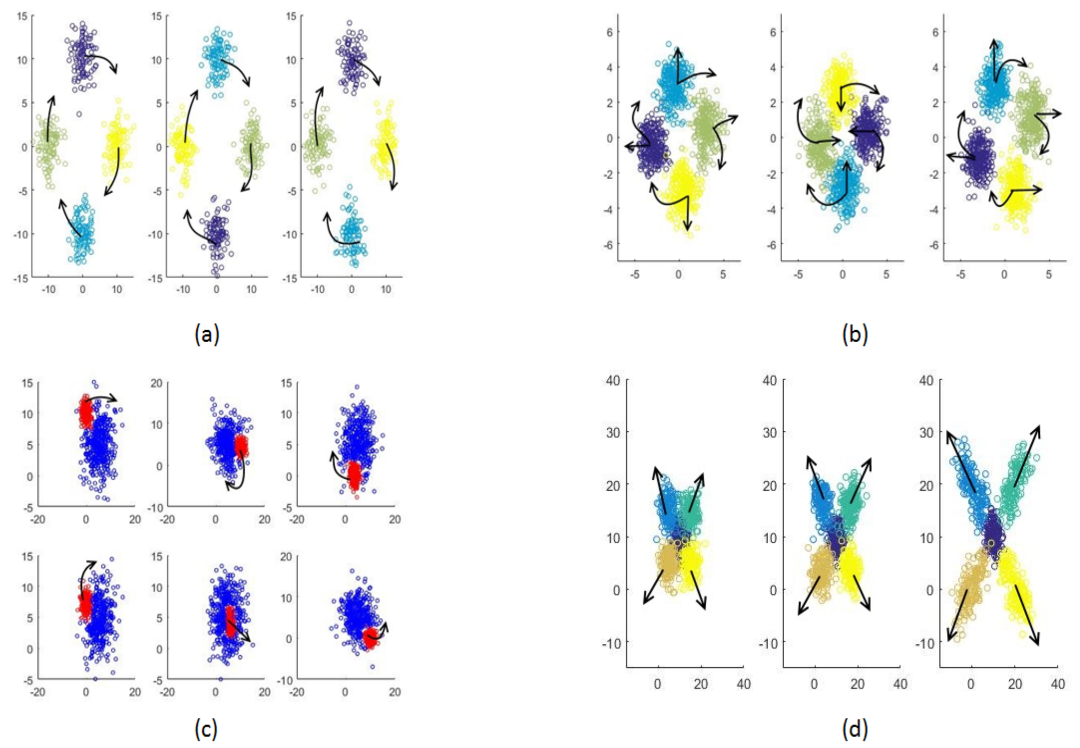


*Figure* 14. Progress of drift for 4CR, 4CRE-V2, 1Csurr, and 4CE1CF datasets;(a) 4CR data for three different snapshots with black arrows representing the drift direction of each class throughout the experiment, (b) 4CRE-V2 data for three different snapshots, (c) 1Csurr data for six different snapshots with black arrow representing the drift direction of red class throughout the experiment, (d) 4CE1CF data for three different snapshots with black arrows representing the drift direction of 4 classes with the middle class stay stationary throughout the experiment.

We analyze the algorithms' behavior from three different perspectives: the average classification accuracy shown in Table 2, computational complexity of these algorithms as measured in runtime on a fixed system shown in Table 3, and a more detailed parameter sensitivity based analysis shown in Tables 6, 7, 8, and 9. Our analyses here include SCARGC, MClassification, COMPOSE and LEVEL$_{IW}$. Arbitrary sub-population tracker (APT) was not included in the analyses, as this algorithm's steep computational complexity was prohibitive on running of some of the larger datasets. This behavior of APT was also previously reported in [13], even on a simple bi-dimensional problem.

The analysis of this algorithm in [13], when originally compared to COMPOSE also revealed another significant shortcoming – that APT requires all modes of the data distribution to be present at the initialization, and hence can not accommodate scenarios where a distribution splits into multiple modes or vice versa over time. Taken together, then, these two concerns rendered APT to be less competitive compared to other algorithms in real world scenarios and hence was not included in further analysis.

In order complete proper statistical analysis of the algorithms' performance to determine whether there are statistically significant differences with respect to the aforementioned figures of merit (accuracy, runtime, parameter sensitivity), we ran all algorithms on all datasets multiple times, and then used the Friedman test along with its corresponding Nemenyi post-hoc test. The post-hoc test results comparing the statistical significance ($p \leq 0.05$) for accuracy and execution time are found in Tables 4 and 5, respectively. Empty cells at the intersection of any two algorithms indicate that the difference in accuracy or execution time between those two algorithms was not statistically significant. A left arrow ($\leftarrow$) or an up arrow ($\uparrow$) at the intersection of any two algorithms represents a statistically significant difference, with the direction of the arrow indicating the classifier that performed significantly better.

The results in this chapter are organized by algorithm, discussing the observations made for each algorithm under evaluation in comparison to others. All evaluations are based on the Tables mentioned above, which are collectively provided at the end of the chapter, all in one place.

## 5.1 Analysis of Three Versions of COMPOSE

Average accuracy results comparing all three versions of COMPOSE (COMPOSE with $\alpha$-shapes, with GMM and FAST COMPOSE), do not show any significant difference among them, or among any of the other algorithms as shown in Table 4. We do observe, however, that FAST COMPOSE – while not quite with statistical significance at 0.05 level – does perform consistently better on most datasets compared to all other algorithms, and provides the lowest overall average rank (lower rank is better in performance, rank 1 is the best algorithm and rank 7 is the worst algorithm). While all three versions of COMPOSE provide similar classification performance when averaged across all datasets (at least in terms of lack of statistical significance), looking at a particular dataset where the difference is significant may provide additional insights. Table 2, which lists the average accuracy of the algorithms, shows that performance is similar for all datasets except $5CVT$. $5CVT$ is a dataset in which five classes ($5C$) are subject to vertical translation ($VT$) in one direction. Four different snapshots of this dataset are illustrated in Figure 15, where black arrows indicate the (common) direction of drift for all classes. This dataset, when used in its original form as provided in the repository, seems not particularly challenging at first look (class overlap is relatively modest, so is the drift rate), but a closer inspection of this dataset reveals class imbalance, with one class having twice as many instances as other four classes.

We observe that applying core support extraction process to extract the useful instances as done by COMPOSE.V1 (with $\alpha$-shapes) and COMPOSE.V2 (with GMM) on this imbalanced dataset introduces a bias in the decision boundary away from the unlabeled data at the next time-step. This bias makes it difficult for the semi-supervised learning

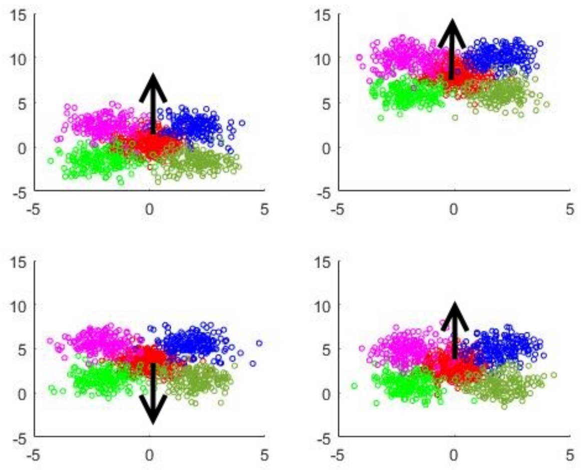


*Figure* 15. Four different snapshots of *5CVT* dataset

(SSL) algorithm to cluster the data properly. When using all instances from the previous time-step as the labeled information – as done in FAST COMPOSE – the labeled instances are closer to the unlabeled data, reducing or eliminating the aforementioned bias, and allows FAST COMPOSE to perform better than either of the first two versions [66]. That said, FAST COMPOSE is not the best performing algorithm on this dataset; that honor goes to MClassification.

Since the dataset is synthetic, we recreated a balanced version of this dataset, and reevaluated all versions of COMPOSE. Figure 16 shows the accuracy of all three versions of COMPOSE. We now see that all versions perform equally well giving an average accuracy around 89%. Furthermore, the class balance helps FAST COMPOSE to perform even better and closer to MClassification than with the imbalanced version of the data.

Computational complexity (as measured in seconds for runtime) among the three versions of COMPOSE as well as other algorithms also provide some useful and interest-

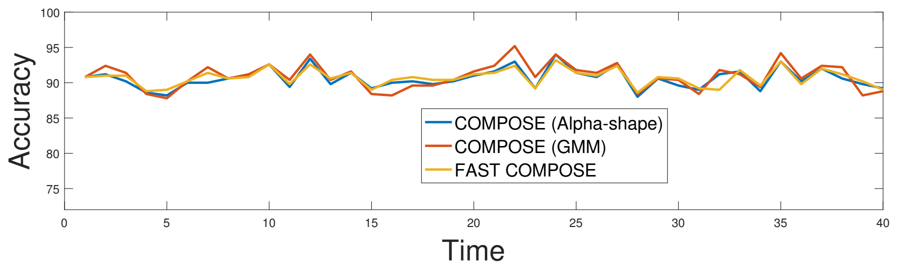


*Figure* 16. Accuracy Comparison of COMPOSE on balanced 5CVT dataset

ing results. As shown in Table 5, COMPOSE ($\alpha$-shape) is found to be the second worst algorithm in terms of computational complexity after MClassification, and performs significantly worse than all other algorithms except SCARGC (1-NN), MClassification and LEVEL$_{\mathrm{IW}}$ (with no significant difference among the last four). For COMPOSE ($\alpha$-shape), the curse of dimensionality is the biggest bottleneck as can be seen from the significantly large computation time it takes for two datasets with even modestly high dimensionality: a 5-dimensional dataset $UG\_2C\_5D$ and the 10-dimensional real world dataset $keystroke$. We can easily see that the computational complexity of COMPOSE ($\alpha$-shape) increases exponentially with dimensionality, and therefore is impractical to use for large dimensional datasets. With respect to execution time, COMPOSE (GMM) shows significant improvement over COMPOSE ($\alpha$-shape), SCARGC (1-NN), and MClassification as seen in Table 5. FAST COMPOSE shows significant improvement over all other algorithms except COMPOSE (GMM) and SCARGC (SVM). We observe that FAST COMPOSE also performs consistently better on most datasets with respect to computation time, providing the lowest rank as shown in Table 3. FAST COMPOSE thus becomes the fastest algorithm known in the literature to handle extreme verification latency leaving behind SCARGC (SVM) which was previously known to be the fastest algorithm in literature per claims made in [15].

In addition to classification accuracy and runtime based computational complexity, we also investigated the parameter sensitivity of these algorithms. Parameter sensitivity analysis measures the robustness of a given algorithm's performance in response to changes in the algorithm's most influential free parameters. In general, we prefer *stable algorithms*, whose performances do not change wildly for modest changes in their free parameters.

COMPOSE.V1 and COMPOSE.V2 employ two modules, namely core support extraction and semi-supervised learning (SSL), each requiring their own free-parameters. The primary free parameters for COMPOSE-V1 are $\alpha$-shape detail level $\alpha$, $\alpha$-shape compaction percentage $CP$, and the number of clusters $k$ for cluster and label SSL algorithm. COMPOSE.V2 requires the number of Gaussian mixtures components $K$, compaction percentage parameter $CP$, and number of clusters parameter $k$ for cluster and label SSL algorithm. All these parameters normally require fine tuning in order to give good results. COMPOSE.V3, i.e. FAST COMPOSE, is introduced primarily to reduce the computation complexity of the algorithm, but it also reduces the number of free-parameters by removing the core support extraction module, and hence requires only the number of clusters parameter $k$. Therefore we perform the sensitivity analysis of COMPOSE with respect to this parameter common to all three versions of COMPOSE. Table 8 shows the results obtained by COMPOSE using cluster-and-label, where for each dataset, we provide the COMPOSE performance with the optimal $k$ value, as well as $k$ incorrectly chosen by just "1." This $\pm 1$ represents the smallest possible change in $k$ around its optimal value. For example, if the optimal value is $k = 4$, the three values of $k$ used for comparison are $k = 3, k = 4$, and $k = 5$. When optimal $k$ is two, the selection of $k = 1$ is, of course, meaningless, as $k = 1$

would result in all instances being classified into the same class. Hence, such cases are indicated as N/A in Table 8. We observe that the cluster-and-label is able to identify the structure in the data from few labeled instances, and it does so reasonably well even when there is overlap among the clusters. However, this performance is subject to correct choice of the number of clusters $k$ in the data, to which it tends to be rather sensitive, and in most datasets changing the value of $k$ from the optimal value even just by 1, significantly and catastrophically reduces the average accuracy for that dataset.

In summary, then, there is no statistically significance difference among any of the algorithms with respect to classification accuracy (though FAST COMPOSE consistently perform better). FAST COMPOSE and COMPOSE with GMM are significantly better in terms of runtime, and LEVEL$_{\text{IW}}$ appears to be more robust with respect to parameter variations among other algorithms.

## 5.2 Analysis of SCARGC

We included two versions of SCARGC, one using nearest neighborhood (1NN) and the other using support vector machines (SVM), neither of which provided any significant difference over any of the other algorithms in terms of classification accuracy, as shown in table 4. However the dataset $5CVT$ is also a useful case to explain the behavior of this algorithm in detail. Recall that the original version of this dataset as provided in the repository contains class imbalance, which causes both versions of the SCARGC algorithm to catastrophically fail on this dataset as shown in Figure 17. Again we reevaluated the performance of both versions of SCARGC on the recreated balanced version of the data as shown in Figure 17. We observe that class balance helps both versions of SCARGC to

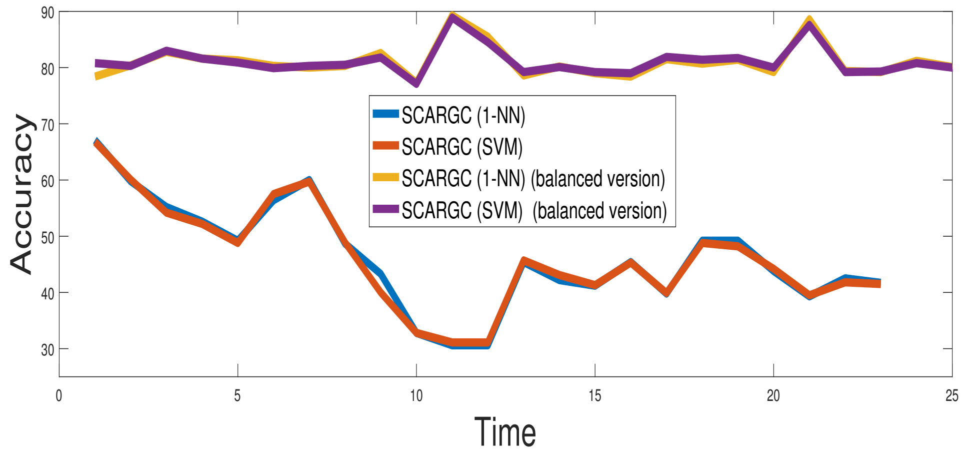


*Figure* 17. Classification accuracy comparison of SCARGC on 5CVT dataset

perform well, the same behavior we already observed with all versions of COMPOSE on this dataset.

With respect to the execution time, SCARGC (1-NN) does not show significant improvement over any algorithm, while SCARGC (SVM) shows significant improvement over COMPOSE ($\alpha$-shape), and MClassification as shown in table 5. FAST COMPOSE showed a significant improvement over SCARGC (1-NN) as previously discussed, and as can also be seen in table 5: the computational performance of FAST COMPOSE is not significantly better than SCARGC (SVM), however, FAST COMPOSE does take less computation time on almost every dataset as compared to SCARGC (SVM).

SCARGC has three input parameters, initial labeled data, pool size and the number of clusters. The authors in the paper [15] show that SCARGC is robust to the change in the values of the initial labeled data and the pool size (the number of instances in each batch evaluated by the algorithm at any given time). Therefore, we fixed and set the pool size equal to the batch size (drift interval shown in Table 1) used in all versions of COMPOSE and LEVEL$_{\mathrm{IW}}$ to ensure the fairness in comparison. As with all algorithms, we also assume that the entire initial batch of the data is labeled, followed by all unlabaled data. This allows all algorithm to see the exact same data in each batch.

The third parameter, the number of clusters $k$, is the more useful one to test with respect to the parameter sensitivity. For the sensitivity analysis, we followed a similar procedure as we did for COMPOSE, and we evaluated SCARGC using the optimal $k$ value, as well as $k$ incorrectly chosen by just ”1”, as shown in Table 6. We observed that performance shown by SCARGC is also quite sensitive to correct choice of this parameter, as the performance drops dramatically and significantly for incorrect choices of $k$, particularly for the cases with class overlap. Overestimating the value of $k$ from its optimal value does not hurt the performance much for those datasets that do not have class overlap, though - perhaps not surprisingly - underestimating this value does negatively impact the classification accuracy.

## 5.3 Analysis of MClassification

MClassification behaves similarly to other algorithms in terms of the classification accuracy when averaged across all datasets, and does not provide any significant difference as shown in Table 4. Looking in detail into its behavior for three specific datasets, namely $FG\_2C\_2D$, $MG\_2C\_2D$, and $5CVT$ is however more useful. $FG\_2C\_2D$ is the dataset where two bi-dimensional classes are represented as four Gaussians. Figure 18 illustrates the behavior of this dataset over time, which shows that one class traverses the perimeter of the other class before diagonally criss-crossing the other class resulting in a complete class overlap.

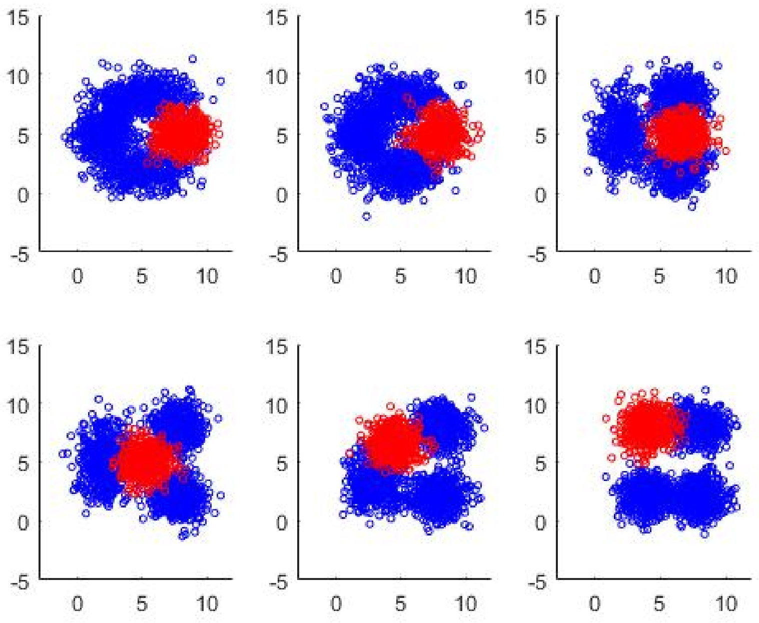


*Figure* 18. Six Snapshots of $FG\_2C\_2D$ data

Figure 19 shows the accuracy obtained on this dataset by the four algorithms under consideration. All versions of COMPOSE and both versions of SCARGC show similar accuracy on this dataset. We only include FAST COMPOSE and SCARGC (1-NN) in the comparison result for simplicity. We note that after time-step 30, the average accuracy for MClassification drops down to 40%, a significant drop in the average accuracy compared to other algorithms. We find that the three Gaussians which are representing one class in the data have their means close to each other initially, and evolve without changing their mean positions too much, but after timestep 30 these three Gaussians start drifting in different directions forming three different clusters. More importantly, the fourth Gaussian representing the other class also starts drifting towards the opposite direction traversing the opposing class diagonally, resulting in a complete class overlap. This sudden change

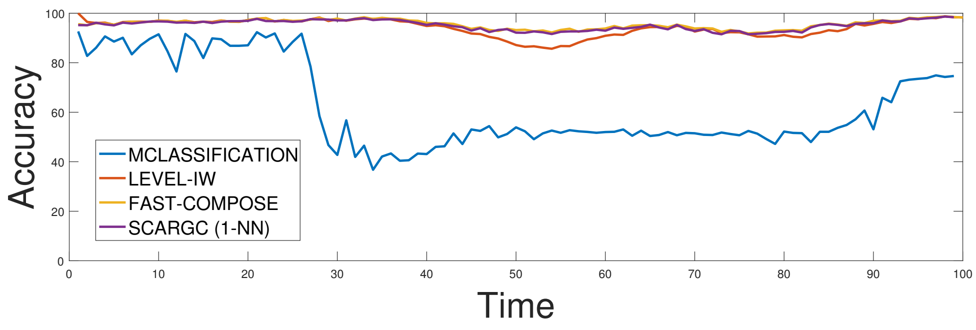


*Figure* 19. Accuracy of Algorithms on $FG\_2C\_2D$ data

in the positions and parameters of the distributions representing two different classes co-occurring with the overlap of these classes appears to be the reason for the significant drop in the average accuracy for MClassification. The other algorithms do see a small drop in their performance when the overlap occurs (as expected), but they do not lose track of the clusters with the sudden change in the positions and parameters of the distributions representing classes.

$MG\_2C\_2D$ dataset also contains two classes in two dimensions, where the distribution of one class (indicated in blue) starts with two modes, whereas that of the other class has a unimodal distribution. The positions and the parameters of the distribution of the classes change over time. The evolving behavior of this dataset is shown in Figure 20 for six different time steps, which indicates that the red class splits into a bi-modal distribution, followed later by merging back into a unimodal distribution, while traversing the blue class perimeter (recall that APT cannot handle such cases).

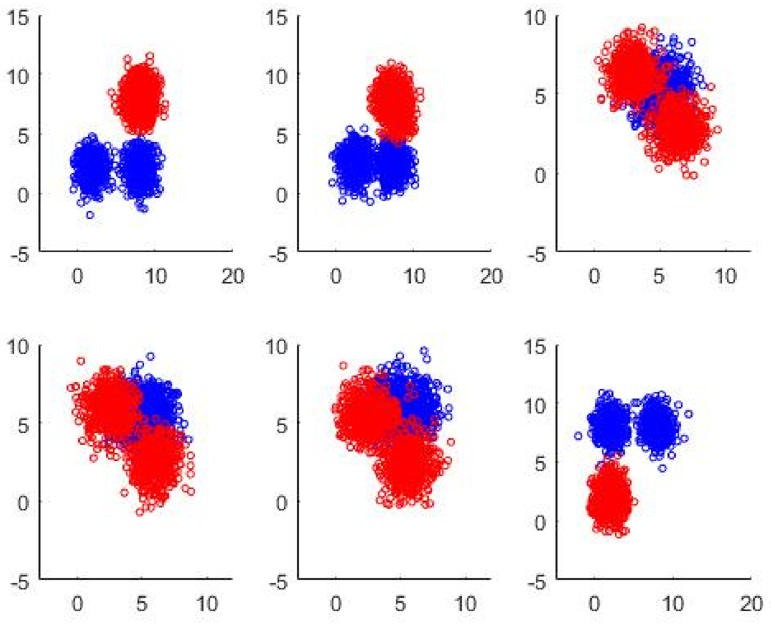


*Figure* 20. Six Snapshots of $MG\_2C\_2D$ data

Figure 21 illustrates the accuracy obtained from different algorithms, which shows that between time-steps 30 to 70, the accuracy for MClassification drops down from 100% to an average of 65%. The reason for this drop is again attributed to the sudden change in the modes or clusters representing two classes of the data initially, co-occurring with the overlap of classes. The other algorithms albeit seeing a drop in their performance because of the significant overlap, do not lose track of the distributions even those distributions diverge into multiple modes. The reason MClassification recovers after time step 70 is that now the initial modes or clusters (two modes for blue class and the other class with one mode) start representing the classes again, and more importantly these modes also start to separate.

$5CVT$ is the 5-class vertical translation data mentioned before and shown in Fig.

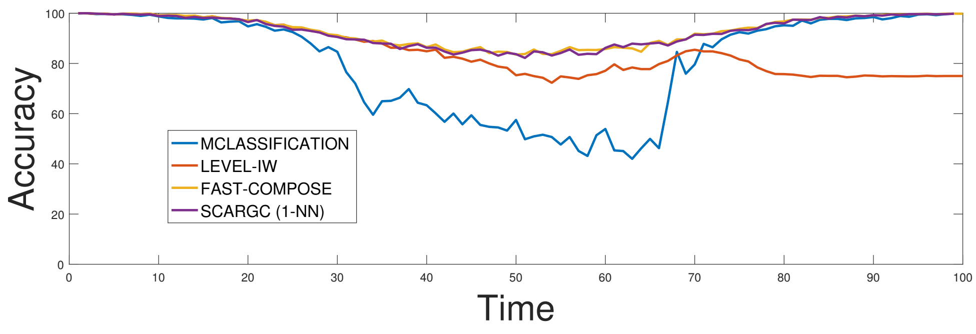


*Figure* 21. Accuracy of Algorithms on $MG\_2C\_2D$ data

15. Curiously, Table 2 shows that MClassification is the only algorithm that works well on the original imbalanced version of this dataset, showing the usefulness and practical importance of this algorithm for the scenarios possessing slight imbalance among classes.

With respect to the execution time, this algorithm appears to be the worst algorithm (other than APT), providing the highest rank (highest being the worst and lowest being the best) as shown in Table 3. As shown in Table 5, MClassification takes significantly longer to run than all other algorithms, except COMPOSE with $\alpha$-shape (and perhaps APT) with which the difference is not significant.

From the parameter sensitivity perspective, we first note that MClassification is introduced by the same authors of SCARGC as an alternative that is claimed to use a parameter that is less sensitive and requires no prior knowledge to tune. The only parameter this algorithms uses is the maximum micro-cluster radius threshold $r$, a user-defined parameter that the authors claim is quite robust. The authors further argue that the value $r$ = 0.1 works generally well in all cases. In order to test this claim, we evaluated this algorithm on 8 different values of the parameter $r$, i.e., 0.01, 0.05, 0.1, 0.2, 0.5, 1, 1.5 and 2, whose results are given in the Table 7. For each of the datasets, three different values of $r$ were used, representing the smallest possible value of 0.01 and largest value of $r$ among all values on which the algorithm starts seeing a drop in its performance, and the claimed default value of $r$ = 0.1. We observed that for all datasets except $MG\_2C\_2D$, the lower values of 0.01 and 0.05 do not make any difference to the performance from the optimal value. However, the performance does not remain consistent when the values greater than the optimal value are used: increasing the threshold value decreases the performance. The performance decreases more dramatically for the datasets that possess significant class overlap.

## 5.4 Analysis of LEVEL$_{\text{IW}}$

The average classification accuracy shown by LEVEL$_{\text{IW}}$ for all datasets was, as previously mentioned, not statistically significantly different from the remaining algorithms as shown in Table 4. With respect to the execution time, LEVEL$_{\text{IW}}$ is significantly slower compared to FAST COMPOSE only, as shown in Table 5. As with other algorithms, additional insights can be obtained by further investigating the classification accuracy of LEVEL$_{\text{IW}}$ on certain datasets. More specifically, we observe that LEVEL$_{\text{IW}}$ performs rather poorly for datasets with significant between-class overlap, as can be seen from Table 2. The reason for this relatively poor performance can be traced to the assumptions made by domain adaptation algorithms: the significant between-class overlap coupled with a drifting environment ultimately leads to a significant change in the posterior probability distribution $p(y|x)$ of classes, violating one of the covariance shift assumptions behind domain adaptation algorithms in general, and LEVEL$_{\text{IW}}$ in particular. We note that the ability of other algorithms to perform well even under significant between-class overlap is in fact due to a crucial piece of information provided to them, through one of their free-parameters.

To better understand the underlying behavior of $\text{LEVEL}_{\text{IW}}$, let's consider a specific dataset as a case study, *UG_2C_2D*, where two unimodal bi-dimensional Gaussians circle around each other. This dataset features class overlap at all time steps, but with varying degree, some near complete overlap. Figure 22 illustrates the behavior of this dataset at six different time snapshots.

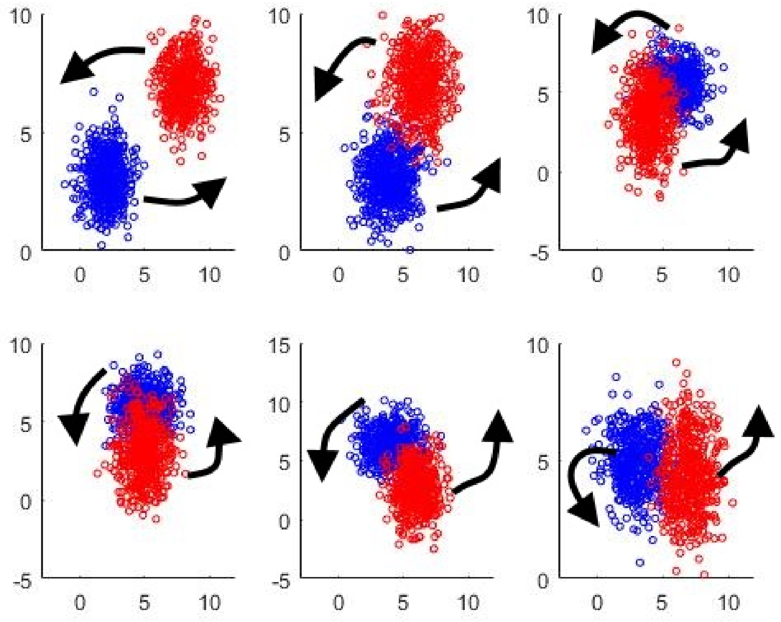


*Figure* 22. UG_2C_2D data for six different snapshots

This is an interesting dataset and explains the performance and behavior of $\text{LEVEL}_{\text{IW}}$ very clearly, as shown in Figure 23. The two classes cross over each other with significant overlap at around time step 60. At this time step, all algorithms, including $\text{LEVEL}_{\text{IW}}$, suffer a steep drop in their classification performance, which is expected. After time step 70,

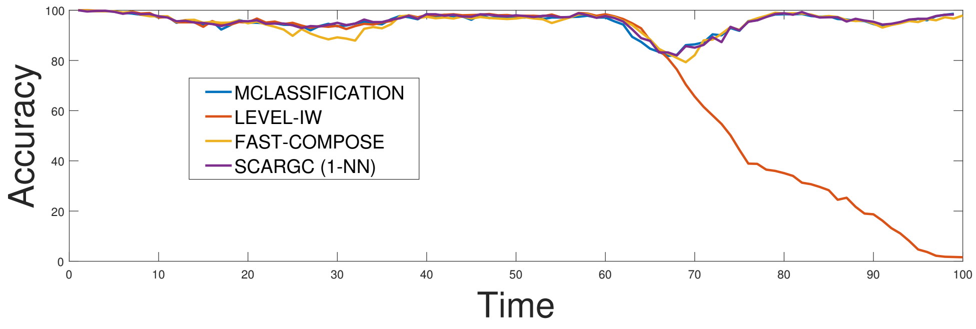


*Figure* 23. Accuracy of algorithms on $UG\_2C\_2D$ data

the two class distributions start pulling away from each other. Unlike other algorithms, LEVEL$_{\text{IW}}$ is unable to recover when the class distributions separate from other. COMPOSE, SCARGC and MClassification can all recover, but only when the correct or optimal value of their primary free parameter is provided. For all three algorithms, the primary free parameter ultimately controls the number of classes / clusters within the data. In this particular example, knowing that the data includes two clusters allow the underlying cluster analysis based procedures for all three algorithms to correctly detect the two classes based on cluster separation when the clusters start moving apart from each other. As Tables 6,7, and 8 show, none of these algorithms can recover from such a severe overlap, however, unless their primary free parameter is correct. This outcome, of course, is not surprising, nor interesting given the complete class overlap (even the supervised Bayes classifier would fail catastrophically in this scenario). What is perhaps more interesting to explore is why LEVEL$_{\text{IW}}$ cannot recover even when its free parameter is chosen correctly, yet it is surprisingly more stable and consistent for various values of its free parameter.

The free parameter for LEVEL$_{\text{IW}}$ is the value of the kernel width $\sigma$ as used in

Gaussian kernel. The kernel width does not provide any direct information on the number of clusters, but rather on the overall smoothness of the decision boundaries. Such information, while not terribly useful after a complete overlap, provides more protection and less sensitivity to minor or even moderate changes in its value. To see this effect, a parameter sweep range was chosen to cover a range commonly known to work well in other algorithms that use Gaussian kernels, and include the values of 0.01, 0.1, 0.2, 0.5, 1, 1.2, 1.5, 2, and 5. In Table 9, we show the performance of LEVEL$_{\text{IW}}$ for each of the datasets with three different values of $\sigma$, representing the smallest and largest values of $\sigma$ on which the algorithm performs well, as well as an additional value in the middle of the two. We observe that LEVEL$_{\text{IW}}$ is surprisingly robust to such wide fluctuations of $\sigma$ values of typically five fold, and sometimes as wide as an order of magnitude difference. This outcome shows the consistent and stable performance of LEVEL$_{\text{IW}}$, its most prominent advantage over remaining algorithms.

## 5.5 Analysis on two Additional Real World Datasets

The *Keystroke* dataset that was included in all aforementioned experiments is the only real world dataset in the original benchmark. That benchmark was used in part because it was used by other algorithms, allowing a fair comparison of our results to those reported in their respective publications [13–16, 66]. We had access to two additional datasets which we used separately, on which we evaluated all four main groups of algorithms. In this section we discuss the behavior of these algorithms on these two additional real world datasets, namely *Weather* and *Traffic* datasets. For this analysis, among three versions of COMPOSE, we use COMPOSE.V3 (FAST COMPOSE), because of its fewer parameter requirements and reduced computational complexity, and in general we now know that it works as well or better than the previous two versions.

The *Weather* dataset is created by our group in one of our prior work [13], and is based on the raw data obtained from the National Oceanic and Atmospheric Administration (NOAA) department. The raw data was collected over a 50-year span from Offutt Air Force Base in Bellevue, Nebraska. Eight features (temperature, dew point, sea-level pressure, visibility, average wind speed, max sustained wind speed, and minimum and maximum temperature) are used to determine whether each day experienced precipitation (rain) or not. The data set contains 18,159 daily readings of which 5,698 are *rain* and the remaining 12,461 are *no rain*. Hence this data has moderate class imbalanced with 68.62% of the instances belonging to class 1 while 31.38% of the instances belonging to the other class. Data were grouped into 49 batches of one-year intervals, each containing 365 instances (days); the remaining data were placed into the fiftieth batch as a partial year. The imbalance inherent in the overall data, combined with consistent significant class overlap caused all algorithms to classify all data to one class, giving (a false sense of) accuracy of 69% on this dataset as shown in Figure 24. Therefore, the results on this dataset are inconclusive.

The second real dataset we use in our analysis is the *Traffic* dataset, which was first introduced in [70]. This dataset consists of 5,412 instances, 512 real attributes and 2 classes, representing whether a traffic intersection is busy (has cars in the intersection) or empty. The images in this dataset are captured from a fixed traffic camera continuously observing an intersection over a two-week period. Some sample images of this dataset are shown in Figure 25.

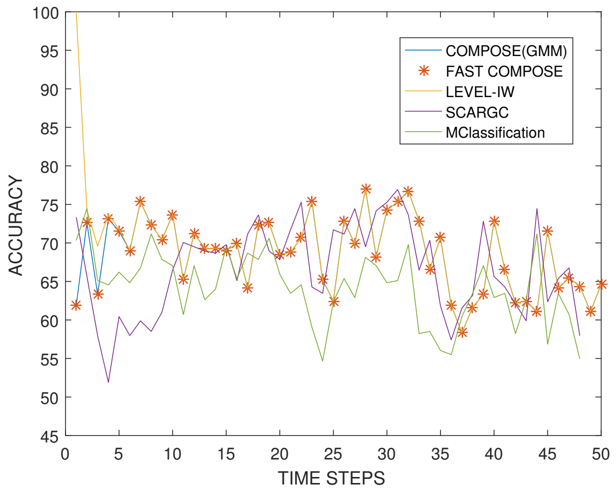


*Figure* 24. Accuracy of algorithms on real world *weather* data

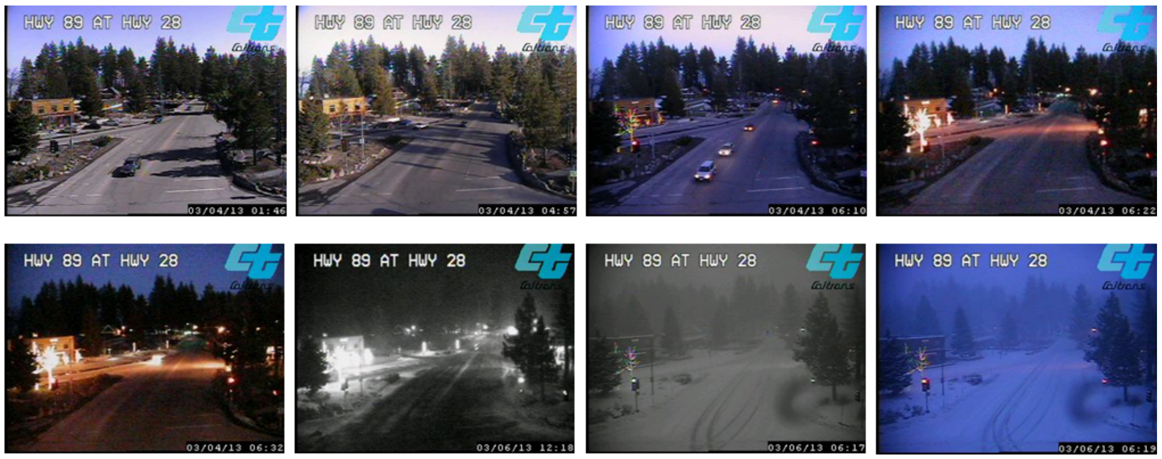
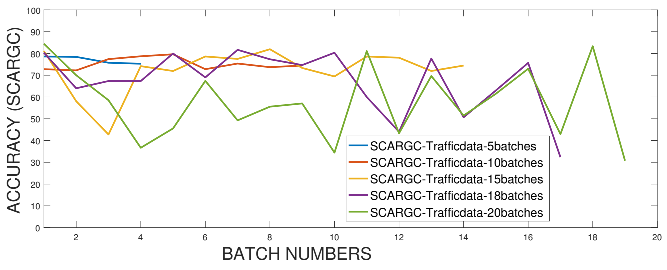


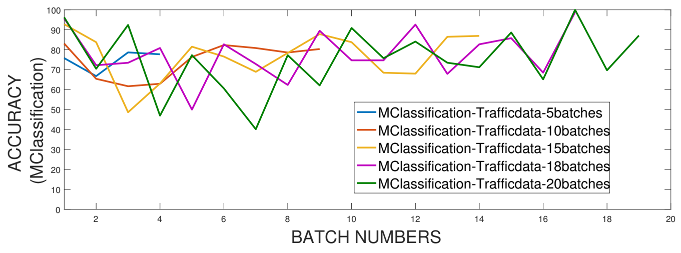
*Figure* 25. Sample images of traffic scenes streaming from a traffic camera

The concept drift in this dataset is due to the ambient changes in the scene that occur because of the variations in illumination, shadows, fog, snow, or even light saturation from oncoming cars, etc. We observe that this dataset also possesses imbalance but not as significant as the *Weather* dataset: out of 5,412 instances, 3,168 instances (58.54%) belong to class 1, while 2,244 instances (41.46%) belong to the other class. While the overall data does not have significant imbalance inherent in it, dividing the data into batches does add significant imbalance to certain batches of data. The imbalance becomes increasingly more significant with the number of batches.

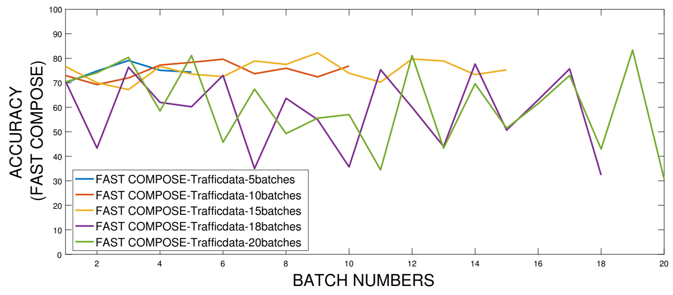
Figure 26 shows the performance of each algorithm on this dataset with different number of batches, where Figure 26(a) represents the classification accuracy of SCARGC for 5, 10, 15, 18, and 20 batches. Figure 26(b), Figure 26(c), and Figure 26(d) show the same information for MClassification, FAST COMPOSE and LEVEL$_{\text{IW}}$, respectively. We observe that all algorithms show around 76% classification accuracy, so long as the number of batches is less than or equal to 18.



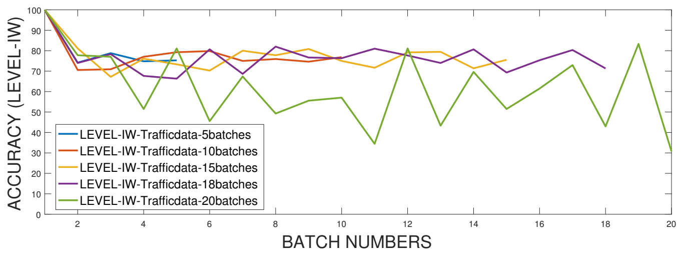
(a) SCARGC performance



(b) MClassification performance



(c) FAST COMPOSE performance



(d) LEVEL$_{\text{IW}}$ performance

*Figure* 26. Accuracy of algorithms on *Traffic* dataset using various batch sizes

Table 2

*Average classification accuracy*

| DATASETS | COMPOSE ($\alpha$-shape) | COMPOSE (GMM) | FAST COMPOSE | SCARGC (1-NN) | SCARGC (SVM) | MClassification | $\text{LEVEL}_{\text{IW}}$ |
|---|---|---|---|---|---|---|---|
| 1CDT | 99.96(2) | 99.85(5) | 99.97(1) | 99.69(7) | 99.72(6) | 99.89(4) | 99.92(3) |
| 1CHT | 99.60(2) | 99.34(6) | 99.57(3) | 99.69(1) | 99.27(7) | 99.38(5) | 99.52(4) |
| 1CSurr | 90.95(5) | 89.72(6) | 95.64(1) | 94.53(3) | 94.99(2) | 85.15(7) | 91.30(4) |
| 2CDT | 96.58(1) | 95.92(2) | 95.17(4) | 87.71(6) | 87.82(5) | 95.23(3) | 58.32(7) |
| 2CHT | 90.39(1) | 89.63(2) | 89.41(3) | 83.62(5) | 83.39(6) | 87.93(4) | 52.15(7) |
| 4CE1CF | 93.92(5) | 93.90(6) | 93.95(4) | 94.04(3) | 92.79(7) | 94.38(2) | 97.74(1) |
| 4CR | 99.99(2.5) | 99.99(2.5) | 99.99(2.5) | 99.96(6) | 98.94(7) | 99.98(5) | 99.99(2.5) |
| 4CRE-V2 | 92.59(1) | 92.30(3) | 92.46(2) | 91.34(6) | 91.46(5) | 91.59(4) | 24.10(7) |
| 5CVT | 57.97(3) | 45.10(6) | 81.33(2) | 46.26(4) | 46.19(5) | 88.30(1) | 33.10(7) |
| FG_2C_2D | 87.90(6) | 95.50(5) | 95.58(3) | 95.51(4) | 95.60(2) | 62.48(7) | 95.71(1) |
| GEARS_2C_2D | 90.98(7) | 95.83(3) | 91.26(6) | 95.99(2) | 95.81(4) | 94.73(5) | 97.74(1) |
| MG_2C_2D | 93.12(2) | 93.20(1) | 93.02(3) | 92.92(5) | 92.94(4) | 80.58(7) | 85.44(6) |
| UG_2C_2D | 95.63(3) | 95.71(1) | 95.61(5) | 95.65(2) | 95.62(4) | 95.28(6) | 74.34(7) |
| UG_2C_3D | 94.92(3) | 95.20(1) | 95.12(2) | 94.83(5) | 94.91(4) | 94.72(6) | 64.69(7) |
| UG_2C_5D | 92.07(2) | 92.13(1) | 91.99(3) | 91.38(4) | 90.94(6) | 91.25(5) | 80.17(7) |
| keystroke | 84.31(7) | 87.21(5) | 85.92(6) | 88.07(3.5) | 88.07(3.5) | 90.62(1) | 90.56(2) |
| Average Rank (lower is better) | 3.2813 | 3.4688 | 3.1563 | 4.1563 | 4.8438 | 4.5000 | 4.5938 |

The only minor exception is MClassification algorithm, which can perform equally well even if the data is divided into more than 18 batches (for instance 20 batches as seen in Figure 26(b)). We attribute this behavior to the online nature of this algorithm, as it can process data one example or instance at a time, and hence the algorithm is not bothered by the batch size. Similar behavior was also observed when MClassification was evaluated on the original version of $5CVT$ benchmark data. With all other algorithms, the problem with batch size can be linked to the class imbalance: If the data is split into 20 batches, ten batches contain on average 68% and 32% imbalance among classes, while the other ten batches contain imbalance on average equal to the imbalance of the overall data i.e. 58.54% and 41.46%. These results further confirm a mutual shortcoming of concept drift algorithms that are asked to work under extreme verification latency that they are sensitive to class imbalance.

Table 3

*Average execution time (in seconds)*

| DATASETS | COMPOSE ($\alpha$-shape) | COMPOSE (GMM) | FAST COMPOSE | SCARGC (1-NN) | SCARGC (SVM) | MClassification | LEVEL$_{IW}$ |
|---|---|---|---|---|---|---|---|
| 1CDT | 19.18(6) | 4.21(3) | 1.15(1) | 10.20(4) | 2.50(2) | 64.75(7) | 15.02(5) |
| 1CHT | 19.76(6) | 4.04(3) | 1.17(1) | 10.76(4) | 3.29(2) | 62.36(7) | 15.34(5) |
| 1CSurr | 72.84(6) | 7.32(2) | 2.53(1) | 51.78(5) | 16.40(3) | 220.49(7) | 43.83(4) |
| 2CDT | 20.21(6) | 2.89(2) | 1.46(1) | 10.00(4) | 3.34(3) | 62.48(7) | 15.71(5) |
| 2CHT | 19.59(6) | 3.55(3) | 1.41(1) | 10.09(4) | 2.89(2) | 60.77(7) | 15.79(5) |
| 4CE1CF | 241.16(6) | 44.14(2) | 8.41(1) | 210.97(5) | 134.56(3) | 775.59(7) | 137.82(4) |
| 4CR | 213.51(6) | 55.90(2) | 12.04(1) | 91.22(4) | 56.22(3) | 608.00(7) | 148.32(5) |
| 4CRE-V2 | 216.55(5) | 34.82(2) | 6.44(1) | 280.27(6) | 41.51(3) | 641.46(7) | 147.81(4) |
| 5CVT | 29.09(5) | 6.16(3) | 2.52(1) | 40.08(6) | 6.08(2) | 89.02(7) | 19.21(4) |
| FG_2C_2D | 229.34(5) | 16.04(2) | 3.80(1) | 587.19(6) | 54.58(3) | 870.12(7) | 185.77(4) |
| GEARS_2C_2D | 237.24(5) | 14.45(2) | 2.50(1) | 609.95(7) | 26.91(3) | 497.87(6) | 186.42(4) |
| MG_2C_2D | 228.96(5) | 15.38(2) | 4.26(1) | 583.76(6) | 53.44(3) | 740.75(7) | 190.81(4) |
| UG_2C_2D | 115.30(5) | 16.92(2) | 3.45(1) | 152.24(6) | 23.27(3) | 362.48(7) | 72.69(4) |
| UG_2C_3D | 936.18(7) | 15.64(2) | 2.60(1) | 747.96(5) | 62.28(3) | 881.07(6) | 176.53(4) |
| UG_2C_5D | 2138.39(7) | 15.97(2) | 2.65(1) | 849.03(5) | 265.92(4) | 977.53(6) | 176.84(3) |
| keystroke | 31761.70(7) | 2.02(4) | 1.16(3) | 0.82(2) | 0.68(1) | 6.62(6) | 2.30(5) |
| Average Rank (lower is better) | 5.8125 | 2.3750 | 1.1250 | 4.9375 | 2.6875 | 6.7500 | 4.3125 |

Table 4

*Statistical significance at $\alpha = 0.05$ for classification accuracy*

| | COMPOSE($\alpha$-shape) | COMPOSE(GMM) | FAST COMPOSE | SCARGC(1-NN) | SCARGC(SVM) | MClassification | LEVEL$_{IW}$ |
|---|---|---|---|---|---|---|---|
| COMPOSE($\alpha$-shape) | n/a | | | | | | |
| COMPOSE(GMM) | | n/a | | | | | |
| FAST COMPOSE | | | n/a | | | | |
| SCARGC(1-NN) | | | | n/a | | | |
| SCARGC(SVM) | | | | | n/a | | |
| MClassification | | | | | | n/a | |
| LEVEL$_{IW}$ | | | | | | | n/a |

Table 5

*Statistical significance at $\alpha = 0.05$ for execution time*

| | COMPOSE($\alpha$-shape) | COMPOSE(GMM) | FAST COMPOSE | SCARGC(1-NN) | SCARGC(SVM) | MClassification | LEVEL$_{IW}$ |
|---|---|---|---|---|---|---|---|
| COMPOSE($\alpha$-shape) | n/a | ↑ | ↑ | | ↑ | | |
| COMPOSE(GMM) | ← | n/a | | ← | | ← | |
| FAST COMPOSE | ← | | n/a | ← | | ← | ← |
| SCARGC(1-NN) | | ↑ | ↑ | n/a | | | |
| SCARGC(SVM) | ← | | | | n/a | ← | |
| MClassification | | ↑ | ↑ | | ↑ | n/a | ↑ |
| LEVEL$_{IW}$ | | | ↑ | | | ← | n/a |

Table 6

*Accuracy with three different values of k (SCARGC)*

| DATASETS | Reduced k (Accuracy) | Optimal k (Accuracy) | Increased k (Accuracy) |
|---|---|---|---|
| 1CDT | N/A | k=2 (99.72) | k=3 (99.72) |
| 1CHT | N/A | k=2 (99.27) | k=3 (99.22) |
| 1CSurr | k=4 (91.68) | k=5 (94.99) | k=6 (91.66) |
| 2CDT | N/A | k=2 (87.82) | k=3 (51.99) |
| 2CHT | N/A | k=2 (83.39) | k=3 (67.48) |
| 4CE1CF | k=4 (2.15) | k=5 (92.79) | k=6 (49.67) |
| 4CR | k=3 (25.33) | k=4 (98.94) | k=5 (98.94) |
| 4CRE-V2 | k=3 (24.82) | k=4 (91.46) | k=5 (39.72) |
| FG_2C_2D | k=3 (68.49) | k=4 (95.60) | k=5 (94.91) |
| GEARS_2C_2D | N/A | k=2 (95.81) | k=3 (88.06) |
| MG_2C_2D | k=3 (64.87) | k=4 (92.94) | k=5 (82.76) |
| UG_2C_2D | N/A | k=2 (95.62) | k=3 (57.19) |
| UG_2C_3D | N/A | k=2 (94.91) | k=3 (80.20) |
| UG_2C_5D | N/A | k=2 (90.94) | k=3 (75.08) |
| keystroke | k=9 (57.43) | k=10 (88.07) | k=11 (58.07) |

Table 7

*Accuracy with three different values of $r$ (MClassification)*

| DATASETS | lowest $r$ (Accuracy) | Middle $r$ (Accuracy) | Highest $r$ (Accuracy) |
|---|---|---|---|
| 1CDT | $r$=0.01(99.85) | $r$=0.1(99.89) | $r$=2(97.85) |
| 1CHT | $r$=0.01(99.23) | $r$=0.1(99.38) | $r$=2(92.97) |
| 1CSurr | $r$=0.01(84.80) | $r$=0.1(85.15) | $r$=0.5(48.67) |
| 2CDT | $r$=0.01(94.76) | $r$=0.1(95.23) | $r$=0.5(55.84) |
| 2CHT | $r$=0.01(86.50) | $r$=0.1(87.93) | $r$=0.5(56.37) |
| 4CE1CF | $r$=0.01(94.59) | $r$=0.1(94.38) | $r$=2(96.21) |
| 4CR | $r$=0.01(99.98) | $r$=0.1(99.98) | $r$=1(23.02) |
| 4CRE-V2 | $r$=0.01(91.21) | $r$=0.1(91.59) | $r$=0.5(27.80) |
| FG_2C_2D | $r$=0.01(59.20) | $r$=0.1(62.48) | $r$=0.2(55.84) |
| GEARS_2C_2D | $r$=0.01(95.23) | $r$=0.1(94.73) | $r$=0.3(93.90) |
| MG_2C_2D | $r$=0.01(51.10) | $r$=0.1(80.58) | $r$=0.2(74.41) |
| UG_2C_2D | $r$=0.01(95.12) | $r$=0.1(95.28) | $r$=0.5(51.87) |
| UG_2C_3D | $r$=0.01(94.57) | $r$=0.1(94.72) | $r$=0.5(52.44) |
| UG_2C_5D | $r$=0.01(91.31) | $r$=0.1(91.25) | $r$=1(68.17) |
| keystroke | $r$=0.01(90.62) | $r$=0.1(76.90) | $r$=0.2(73.86) |

Table 8

*Accuracy with three different values of k (COMPOSE)*

| DATASETS | Reduced k (Accuracy) | Optimal k (Accuracy) | Increased k (Accuracy) |
|---|---|---|---|
| 1CDT | N/A | k=2 (99.85) | k=3 (99.76) |
| 1CHT | N/A | k=2 (99.34) | k=3 (98.72) |
| 1CSurr | k=3 (85.58) | k=4 (94.55) | k=5 (91.52) |
| 2CDT | N/A | k=2 (95.91) | k=3 (52.91) |
| 2CHT | N/A | k=2 (89.63) | k=3 (77.33) |
| 4CE1CF | k=4 (78.96) | k=5 (93.90) | k=6 (94.66) |
| 4CR | k=3 (74.88) | k=4 (99.98) | k=5 (99.98) |
| 4CRE-V2 | k=3 (25.13) | k=4 (92.30) | k=5 (22.78) |
| FG_2C_2D | k=3 (68.91) | k=4 (95.50) | k=5 (95.44) |
| GEARS_2C_2D | N/A | k=2 (95.82) | k=3 (87.99) |
| MG_2C_2D | k=3 (65.32) | k=4 (93.20) | k=5 (92.07) |
| UG_2C_2D | N/A | k=2 (95.71) | k=3 (56.28) |
| UG_2C_3D | N/A | k=2 (95.20) | k=3 (91.46) |
| UG_2C_5D | N/A | k=2 (92.12) | k=3 (88.03) |
| keystroke | k=9 (68.62) | k=10 (87.21) | k=11 (81.56) |

Table 9

*Accuracy with three different values of sigma (LEVEL$_{IW}$)*

| DATASETS | lowest sigma (Accuracy) | Middle sigma (Accuracy) | Highest sigma (Accuracy) |
|---|---|---|---|
| 1CDT | 0.2 (99.91) | 1 (99.91) | 2 (99.92) |
| 1CHT | 0.2 (99.40) | 1 (99.42) | 2 (99.51) |
| 1CSurr | 1 (91.30) | 1.5 (90.00) | 2 (87.79) |
| 2CDT | 0.2 (58.32) | 0.5 (50.32) | 1 (50.48) |
| 2CHT | 0.2 (50.10) | 0.5 (50.89) | 1 (52.15) |
| 4CE1CF | 0.2 (97.74) | 0.5 (97.12) | 1.5 (92.40) |
| 4CR | 0.2 (99.99) | 1 (99.99) | 2 (99.99) |
| 4CRE-V2 | 0.2 (20.96) | 0.5 (20.84) | 1 (24.10) |
| FG_2C_2D | 0.2 (95.71) | 0.5 (86.41) | 1 (94.28) |
| GEARS_2C_2D | 0.2 (97.73) | 1 (95.28) | 2 (95.36) |
| MG_2C_2D | 0.2 (78.03) | 0.5 (78.21) | 1.2 (85.44) |
| UG_2C_2D | 0.2 (70.61) | 0.5 (71.81) | 1 (74.33) |
| UG_2C_3D | 0.1 (61.21) | 1 (64.30) | 2 (64.68) |
| UG_2C_5D | 0.5 (77.67) | 1 (80.07) | 1.5 (80.17) |
| keystroke | 0.5 (88.12) | 1 (90.56) | 2 (89.43) |

# Chapter 6

# Conclusion and Future Work

This thesis introduces and describes two new approaches to learn from a nonstationary (drifting) environment experiencing extreme verification latency, and provides a comprehensive evaluation of existing approaches with respect to classification accuracy, computational complexity and parameter sensitivity.

In a nonstationary streaming environment, the nonstationary data, drawn from a drifting distribution, arrive in a streaming manner. The extreme verification latency places an additional constraint that beyond an initial batch, the entire data stream is assumed unlabeled. The first approach developed in this effort is a modification of the previously developed COMPOSE algorithm that significantly increases the execution speed of the algorithm. The modified version of COMPOSE, named FAST COMPOSE, is the original COMPOSE algorithm whose core support extraction step is replaced by using all of the instances labeled by the SSL in the previous time-step as the new labeled data to be used for the next time-step's SSL step. This simplification of the algorithm produces improved results in both classification accuracy (albeit not at a statistically significant level) and execution time (at a statistically significant level compared to original $\alpha$-shape based COMPOSE and other competing non-COMPOSE based algorithms). The second approach developed as part of this thesis is a modification of the importance weighted least squares probabilistic classifier so that it can work within a streaming data environment and when there is extreme verification latency. The proposed approach is called Learning Extreme VErification Latency with Importance Weighting (LEVEL$_{\text{IW}}$). This approach was developed to determine whether domain adaptation approaches that are intended for single time step training - test distribution mismatch problems can also be used in the streaming data setting. We found that while the algorithm does not provide any significant improvement on classification accuracy, it does provide improved stability - compared to other algorithms - over a range of values in the selection of its free-parameter.

One of most important contribution of this work is the comprehensive and comparative analysis of the available algorithms in the literature to handle extreme verification latency from three different perspectives: classification accuracy, computational complexity and parameter sensitivity. Our goal in this task has been to determine and describe the relative strengths and weaknesses of these algorithms, and point out different cases and scenarios where one algorithm is better suited over the others.

The original COMPOSE algorithms, COMPOSE with $\alpha$-shape (COMPOSE.V1), was a significant contribution to the field when it was first proposed, as it was the only algorithm capable at the time to address the problem of learning in nonstationary environments in the presence of extreme verification latency with no restrictions on the nature of the data distribution. However, that capability came at a steep price: the algorithm is computationally very expensive (though still significantly more efficient than the Arbitrary Population subTracker (APT) as well as the MClassification). The algorithm also provided to be quite sensitive to the choice of its primary free parameters. Despite these shortcomings, and despite several other competing algorithms developed since then, the original COMPOSE algorithm remains competitive with respect to classification accuracy. The second version of COMPOSE, COMPOSE with GMM (COMPOSE.V2), replaced the $\alpha$-shape based approach for determining the core supports with a Gaussian mixture model based density estimation module that dramatically increased its computational efficiency while retaining the classification accuracy of COMPOSE.V1 The latest version of COMPOSE, FAST COMPOSE proposed in this work, further improves the classification accuracy as well as the computational efficiency compared to all other algorithms. One remaining issue with FAST COMPOSE, however, is its sensitivity to the choice of its primary free parameter, the number of clusters in the cluster-and-label based SSL algorithm used in its core support computation.

SCARGC was developed as a competing algorithm to the original COMPOSE with the primary advantage of better computational efficiency. SCARGC with nearest neighbor (1NN) shows comparable accuracy to other algorithms and is less computationally expensive compared to COMPOSE ($\alpha$-shape), and MClassification (but not against COMPOSE with GMM or FAST COMPOSE), while it too is also sensitive to the choice of its primary free parameter – number of clusters $k$ in k-means clustering based subroutine it uses. SCARGC with SVM while perhaps reasonable with respect to computational burden, was found to be the worst (highest rank) in terms of classification accuracy. SCARGC with SVM retains the high parameter sensitivity as with SCARGC (1NN).

MClassification shows comparable accuracy performance to other algorithms but appears to be the worst algorithm in computationally complexity (other than APT), requiring more runtime than even COMPOSE with $\alpha$-shape on most datasets. This behavior is attributed to its online nature. In fact, MClassification is the only algorithm that is capable of processing the data in an online manner, a distinct advantage in a streaming environment, but that advantage appears to be unrealized or wasted due to the heavy computational burden. This algorithm is also quite sensitive to its primary free parameter.

LEVEL$_{\text{IW}}$, as with other algorithms, performed comparably similar with respect to classification accuracy, is less computationally expensive than COMPOSE.V1, SCARGC (1-NN), and MClassification (but more expensive than FAST COMPOSE, SCARGC (SVM) and COMPOSE.V2). While not the best performing algorithm either in terms of classification accuracy or computational efficiency, $\text{LEVEL}_{\text{IW}}$ has one advantage over other algorithms: greater robustness and stability compared to all of the remaining algorithms with respect to relatively wide fluctuations of the value of its primary free parameter.

## 6.1 Summary of Future Work

Further work is needed to generate or acquire more challenging datasets, as most algorithms perform similarly on the current synthetic benchmark datasets. Currently, there is a lack of datasets that contain abruptly changing distributions, datasets with recurring concepts or more severe class imbalances, datasets that have substantial feature or class noise, datasets with significant amount of outliers, datasets with very little or almost no shared support, and high dimensional datasets to name a few.

We already know from the analyses shown in this thesis that the algorithms described here will not work in all of the above-mentioned scenarios, such as abruptly changing distributions or severe class imbalance. Often in science, however, it is a challenging dataset, or a collection of datasets that provide the motivation for the development of specialized algorithms within a specific disciple. Additionally, future work is needed to provide machine learning community with an algorithm that can perform well with respect to classification accuracy, computationally complexity and parameter sensitivity as well as able to handle challenging datasets mentioned above under initially labeled non-stationary environments.

## References

[1] Stephen Grossberg. Nonlinear neural networks: Principles, mechanisms, and architectures. *Neural networks*, 1(1):17–61, 1988.

[2] Michael D Muhlbaier and Robi Polikar. Multiple classifiers based incremental learning algorithm for learning in nonstationary environments. In *Machine Learning and Cybernetics, 2007 International Conference on*, volume 6, pages 3618–3623. IEEE, 2007.

[3] Matthew Karnick, Metin Ahiskali, Michael D Muhlbaier, and Robi Polikar. Learning concept drift in nonstationary environments using an ensemble of classifiers based approach. In *Neural Networks, 2008. IJCNN 2008.(IEEE World Congress on Computational Intelligence). IEEE International Joint Conference on*, pages 3455–3462. IEEE, 2008.

[4] R. Elwell and R. Polikar. Incremental learning of concept drift in non- stationary environments. *IEEE Transactions Neural Networks*, 22(10):1517–1531, 2011.

[5] Gregory Ditzler and Robi Polikar. Incremental learning of concept drift from streaming imbalanced data. *IEEE Transactions on Knowledge and Data Engineering*, 25: 2283–2301, 2013.

[6] J. Kolter and M. Maloof. Dynamic weighted majority: An ensemble method for drifting concepts. *Journal of Machine Learning Research*, 8:2755–2790, July 2007.

[7] W Nick Street and YongSeog Kim. A streaming ensemble algorithm (sea) for large-scale classification. *ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 377–382, 2001.

[8] Hidetoshi Shimodaira. Improving predictive inference under covariate shift by weighting the log-likelihood function. *Journal of statistical planning and inference*, 90(2):227–244, 2000.

[9] Joaquin Quionero-Candela, Masashi Sugiyama, Anton Schwaighofer, and Neil D Lawrence. *Dataset shift in machine learning*. The MIT Press, 2009.

[10] Gregory Ditzler and Robi Polikar. Semi-supervised learning in nonstationary environments. *International Joint Conference on Neural Networks*, pages 2741–2748, 2011.

[11] Karl Dyer Robert Capo and Robi Polikar. Active learning in nonstationary environments. *International Joint Conference on Neural Networks*, pages 1–8, 2013.

[12] G. Marrs, R. Hickey, and M. Black. The impact of latency on online classication learning with concept drift. *Knowledge Science, Engineering and Management*, 6291.

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

[14] G. Krempl. The algorithm apt to classify in concurrence of latency and drift. *Intelligent Data Analysis*, pages 223–233, 2011.

[15] V. M. A. Souza, D. F. Silva, J. Gama, and G. E. A. P. A. Batista. Data stream classication guided by clustering on nonstationary envi- ronments and extreme verication latency. *SIAM International Conference on Data Mining*, pages 873–881, 2015.

[16] V. M. A. Souza, D. F. Silva, G. E. A. P. A. Batista, and J. Gama. Classification of evolving data streams with infinitely delayed labels. *IEEE International Conference on Machine Learning and Applications (ICMLA)*, pages 214–219, 2015.

[17] Gerhard Widmer and Miroslav Kubat. Learning in the presence of concept drift and hidden contexts. *Machine learning*, 23(1):69–101, 1996.

[18] Indrė Žliobaitė. Identifying hidden contexts in classification. *Advances in Knowledge Discovery and Data Mining*, pages 277–288, 2011.

[19] Geoff Hulten, Laurie Spencer, and Pedro Domingos. Mining time-changing data streams. In *Proceedings of the seventh ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 97–106. ACM, 2001.

[20] Lior Cohen, Gil Avrahami-Bakish, Mark Last, Abraham Kandel, and Oscar Kipersztok. Real-time data mining of non-stationary data streams from sensor networks. *Information Fusion*, 9(3):344–353, 2008.

[21] Lior Cohen, Gil Avrahami, Mark Last, and Abraham Kandel. Info-fuzzy algorithms for mining dynamic data streams. *Applied Soft Computing*, 8(4):1283–1294, 2008.

[22] Robert M French. Catastrophic forgetting in connectionist networks. *Encyclopedia of cognitive science*, 2003.

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

[24] Albert Bifet, Geoff Holmes, Bernhard Pfahringer, Richard Kirkby, and Ricard Gavaldà. New ensemble methods for evolving data streams. In *Proceedings of the 15th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 139–148. ACM, 2009.

[25] J Zico Kolter and Marcus A Maloof. Dynamic weighted majority: An ensemble method for drifting concepts. *Journal of Machine Learning Research*, 8(Dec):2755–2790, 2007.

[26] Alexey Tsymbal, Mykola Pechenizkiy, Pádraig Cunningham, and Seppo Puuronen. Dynamic integration of classifiers for handling concept drift. *Information fusion*, 9 (1):56–68, 2008.

[27] Jing Gao, Wei Fan, and Jiawei Han. On appropriate assumptions to mine data streams: Analysis and practice. In *Data Mining, 2007. ICDM 2007. Seventh IEEE International Conference on*, pages 143–152. IEEE, 2007.

[28] Hanady Abdulsalam, David B Skillicorn, and Patrick Martin. Classification using streaming random forests. *IEEE Transactions on Knowledge and Data Engineering*, 23(1):22–36, 2011.

[29] Mohammad Masud, Jing Gao, Latifur Khan, Jiawei Han, and Bhavani M Thuraisingham. Classification and novel class detection in concept-drifting data streams under time constraints. *IEEE Transactions on Knowledge and Data Engineering*, 23(6): 859–874, 2011.

[30] Albert Bifet. Adaptive learning and mining for data streams and frequent patterns. *ACM SIGKDD Explorations Newsletter*, 11(1):55–56, 2009.

[31] Albert Bifet, Geoff Holmes, Richard Kirkby, and Bernhard Pfahringer. Moa: Massive online analysis. *Journal of Machine Learning Research*, 11(May):1601–1604, 2010.

[32] Ryan Elwell and Robi Polikar. Incremental learning in nonstationary environments with controlled forgetting. In *Neural Networks, 2009. IJCNN 2009. International Joint Conference on*, pages 771–778. IEEE, 2009.

[33] Jeffrey C Schlimmer and Richard H Granger. Incremental learning from noisy data. *Machine learning*, 1(3):317–354, 1986.

[34] Cesare Alippi and Manuel Roveri. Just-in-time adaptive classifiers.

[35] Cesare Alippi, Giacomo Boracchi, and Manuel Roveri. Change detection tests using the ici rule. In *Neural Networks (IJCNN), The 2010 International Joint Conference on*, pages 1–7. IEEE, 2010.

[36] Stefan Hoeglinger and Russel Pears. Use of hoeffding trees in concept based data stream mining. In *Information and Automation for Sustainability, 2007. ICIAFS 2007. Third International Conference on*, pages 57–62. IEEE, 2007.

[37] Gregory Ditzler and Robi Polikar. Hellinger distance based drift detection for nonstationary environments. In *Computational Intelligence in Dynamic and Uncertain Environments (CIDUE), 2011 IEEE Symposium on*, pages 41–48. IEEE, 2011.

[38] T Ryan Hoens, Nitesh V Chawla, and Robi Polikar. Heuristic updatable weighted random subspaces for non-stationary environments. In *Data Mining (ICDM), 2011 IEEE 11th International Conference on*, pages 241–250. IEEE, 2011.

[39] Sadaoki Furui. Comparison of speaker recognition methods using statistical features and dynamic features. *IEEE Transactions on Acoustics, Speech, and Signal Processing*, 29(3):342–350, 1981.

[40] Guido Dornhege. *Toward brain-computer interfacing*. MIT press, 2007.

[41] Pradeep Shenoy, Matthias Krauledat, Benjamin Blankertz, Rajesh PN Rao, and Klaus-Robert Müller. Towards adaptive classification for bci. *Journal of neural engineering*, 3(1):R13, 2006.

[42] Jing Jiang and ChengXiang Zhai. Instance weighting for domain adaptation in nlp. In *ACL*, volume 7, pages 264–271, 2007.

[43] Masashi Sugiyama and Motoaki Kawanabe. *Machine learning in non-stationary environments: Introduction to covariate shift adaptation*. MIT press, 2012.

[44] Jing Jiang. A literature survey on domain adaptation of statistical classifiers. *URL: http://sifaka. cs. uiuc. edu/jiang4/domainadaptation/survey*, 3, 2008.

[45] Jing Jiang. *Domain adaptation in natural language processing*. University of Illinois at Urbana-Champaign, 2008.

[46] Wolfgang Härdle, Marlene Müller, Stefan Sperlich, and Axel Werwatz. *Nonparametric and semiparametric models*. Springer Science & Business Media, 2012.

[47] Steffen Bickel, Michael Brückner, and Tobias Scheffer. Discriminative learning under covariate shift. *Journal of Machine Learning Research*, 10(Sep):2137–2155, 2009.

[48] Jiayuan Huang, Arthur Gretton, Karsten M Borgwardt, Bernhard Schölkopf, and Alex J Smola. Correcting sample selection bias by unlabeled data. In *Advances in neural information processing systems*, pages 601–608, 2006.

[49] Nathalie Japkowicz and Shaju Stephen. The class imbalance problem: A systematic study. *Intelligent data analysis*, 6(5):429–449, 2002.

[50] Nitesh V Chawla, Kevin W Bowyer, Lawrence O Hall, and W Philip Kegelmeyer. Smote: synthetic minority over-sampling technique. *Journal of artificial intelligence research*, 16:321–357, 2002.

[51] Nuno Moniz, Paula Branco, and Luís Torgo. Resampling strategies for imbalanced time series. In *Data Science and Advanced Analytics (DSAA), 2016 IEEE International Conference on*, pages 282–291. IEEE, 2016.

[52] Ting Yao, Yingwei Pan, Chong-Wah Ngo, Houqiang Li, and Tao Mei. Semi-supervised domain adaptation with subspace learning for visual recognition. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 2142–2150, 2015.

[53] Wenyuan Dai, Gui-Rong Xue, Qiang Yang, and Yong Yu. Transferring naive bayes classifiers for text classification. In *AAAI*, volume 7, pages 540–545, 2007.

[54] Shuang Ao, Xiang Li, and Charles X Ling. Fast generalized distillation for semi-supervised domain adaptation. In *AAAI*, pages 1719–1725, 2017.

[55] Matthew E Taylor and Peter Stone. Transfer learning for reinforcement learning domains: A survey. *Journal of Machine Learning Research*, 10(Jul):1633–1685, 2009.

[56] Sinno Jialin Pan and Qiang Yang. A survey on transfer learning. *IEEE Transactions on knowledge and data engineering*, 22(10):1345–1359, 2010.

[57] Rich Caruana. Multitask learning. In *Learning to learn*, pages 95–133. Springer, 1998.

[58] Rajat Raina. *Self-taught learning*. Stanford University, 2009.

[59] James J Heckman. Sample selection bias as a specification error (with an application to the estimation of labor supply functions), 1977.

[60] Dyer karl. *COMPOSE: Compacted object sample extraction a framework for semi-supervised learning in nonstationary environments*. Rowan University, 2015.

[61] R. Capo, A. Sanchez, and R. Polikar. Core support extraction for learning from initially labeled nonstationary environments using compose. In *2014 International Joint Conference on Neural Networks (IJCNN)*, pages 602–608, July 2014. doi: 10.1109/IJCNN.2014.6889917.

[62] Xiaojin Zhu and Zoubin Ghahramani. Learning from labeled and unlabeled data with label propagation. Technical Report Technical Report CMU-CALD-02-107, Carnegie Mellon University, 2002.

[63] Kristin Bennett and Ayhan Demiriz. Semi-supervised support vector machines. *Advances in Neural Information processing systems*, pages 368–384, 2002.

[64] Robert Capo, Anthony Sanchez, and Robi Polikar. Core support extraction for learning from initially labeled nonstationary environments using compose. In *Neural Networks (IJCNN), 2014 International Joint Conference on*, pages 602–608. IEEE, 2014.

[65] J. Sarnelle, A. Sanchez, R. Capo, J. Haas, and R. Polikar. Quantifying the limited and gradual concept drift assumption. *International Joint Conference on Neural Networks*, 2015.

[66] M. Umer, C. Frederickson, and R. Polikar. Learning under extreme verification latency quickly: Fast compose. In *2016 IEEE Symposium Series on Computational Intelligence (SSCI)*, pages 1–8, Dec 2016. doi: 10.1109/SSCI.2016.7849962.

[67] Hirotaka Hachiya, Masashi Sugiyama, and Naonori Ueda. Importance-weighted least-squares probabilistic classifier for covariate shift adaptation with application to human activity recognition. *Neurocomputing*, 80:93–101, 2012.

[68] Masashi Sugiyama, Matthias Krauledat, and Klaus-Robert MÃžller. Covariate shift adaptation by importance weighted cross validation. *Journal of Machine Learning Research*, 8(May):985–1005, 2007.

[69] Takafumi Kanamori, Shohei Hido, and Masashi Sugiyama. A least-squares approach to direct importance estimation. *Journal of Machine Learning Research*, 10(Jul): 1391–1445, 2009.

[70] Judy Hoffman, Trevor Darrell, and Kate Saenko. Continuous manifold based adaptation for evolving visual domains. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 867–874, 2014.