10-21-2015

COMPOSE: Compacted object sample extraction a framework for semi-supervised learning in nonstationary environments

Karl Dyer

COMPOSE: COMPACTED OBJECT SAMPLE EXTRACTION FOR SEMI-SUPERVISED LEARNING IN NONSTATIONARY ENVIRONMENTS EXPERIENCING GRADUAL DRIFT

by
Karl Bachman Dyer

A Thesis

Submitted to the
Department of Electrical and Computer Engineering
College of Engineering
In partial fulfillment of the requirement
For the degree of
Master of Science in Electrical and Computer Engineering
at
Rowan University
August 26, 2015

Thesis Chair: Robi Polikar, Ph.D.

Acknowledgements

I would like to thank Dr. Robi Polikar for taking a chance on a mechanical engineering and welcoming me into his Signal Processing & Pattern Recognition Lab. He has been a constant source of encouragement and has been extremely helpful in my professional and life choices. I would like to thank my committee members, Dr. Nidhal Bouaynaya and Dr. Krishan Bhatia, for their help through thesis process. Special thanks to Greg Ditzler and Rob Capo who have been extremely helpful in code development, paper reviews, and a sounding board for some "off the wall" ideas. Finally, I would like to thank my family and friends who have supported and encouraged me to continue pressing forward throughout my academic journey.

Abstract

Karl B. Dyer
COMPOSE: COMPACTED OBJECT SAMPLE EXTRACTION
A FRAMEWORK FOR SEMI-SUPERVISED LEARNING
IN NONSTATIONARY ENVIRONMENTS
2015-2016
Robi Polikar, Ph.D.
Master of Science in Electrical and Computer Engineering

An increasing number of real-world applications are associated with streaming data drawn from drifting and nonstationary distributions. These applications demand new algorithms that can learn and adapt to such changes, also known as concept drift. Proper characterization of such data with existing approaches typically requires substantial amount of labeled instances, which may be difficult, expensive, or even impractical to obtain. In this thesis, compacted object sample extraction (COMPOSE) is introduced - a computational geometry-based framework to learn from nonstationary streaming data - where labels are unavailable (or presented very sporadically) after initialization. The feasibility and performance of the algorithm are evaluated on several synthetic and real-world data sets, which present various different scenarios of initially labeled streaming environments. On carefully designed synthetic data sets, we also compare the performance of COMPOSE against the optimal Bayes classifier, as well as the arbitrary subpopulation tracker algorithm, which addresses a similar environment referred to as extreme verification latency. Furthermore, using the real-world National Oceanic and Atmospheric Administration weather data set, we demonstrate that COMPOSE is competitive even with a well-established and fully supervised nonstationary learning algorithm that receives labeled data in every batch.

Table of Contents

Table of Contents (Continued)

List of Figures

List of Figures (Continued)

Figure Page

List of Tables

Table Page

Chapter 1

Introduction

The fundamental goal of machine learning is to emulate (albeit at a limited scale) the decision making capabilities of the brain, so it is not surprising to find topics in machine learning often parallel human learning methodology. The cognitive development of humans from infancy through adolescence then into adulthood can be likened to three broad categories of machine learning – unsupervised, supervised, and semi-supervised learning, respectively.

The following section draws parallels between human cognitive development and the aforementioned three broad divisions of machine learning. Once an understanding of general machine learning concepts has been established, nonstationary learning – a task humans accomplish innately - is presented as a challenging twist to traditional machine learning paradigms. Throughout this next section machine learning terms are gradually introduced in (parenthetical italics) and by the end of the chapter we will be using only machine learning terms.

The remainder of the chapter presents a global picture of the problem this thesis addresses before narrowing the scope and identifying the specific contributions of this manuscript. An organizational overview of the remainder of this thesis can be found at the end of this chapter.

1.1 Human Cognition and Machine Learning

1.1.1 Three broad divisions of machine learning. At infancy, we observe defining characteristics (features) – such as color, shape, size, etc. – of objects (instances) all around us. However, at this stage of cognitive development we do not necessarily know the names (classes or labels) of all the objects. For example, a toddler playing with blocks may form groups (clusters) of like featured objects, but is unable to follow instruction to sort them by color since he has not learned colors at this stage of development. This scenario is very similar to unsupervised learning algorithms which try to group data into "natural" clusters - where "natural" is defined by the similarity measure used by the clustering algorithm [1] - based solely on analysis of their features. The resulting clusters are assigned cluster identifiers using non-descript roman numerals or alpha-numeric characters, but these identifiers do not contain any information about true class membership.

At youth, we rely heavily on parents and school teachers to provide connections between an object and its accepted name (training). Through repetition and a multitude of examples we are eventually able to make predictions about an object's correct label (classification) even though we have not been formally taught the information prior. For example, after being told that roses, daffodils, and tulips are all flowers we are likely to assume anything with green leaves and brightly colored petals can be referred to as a flower. This scenario draws a strong correlation to supervised learning algorithms which use a set of labeled data to train a classifier - a mathematical model that maps features to corresponding labels – which is to provide class labels for other unknown instances.

By the time we reach adulthood we generally require fewer and fewer labeled examples in order to make an educated guess in unfamiliar situations. In machine learning, this concept is the foundation of semi-supervised learning. Combining the ability of unsupervised learning to form logical clusters with the ability of supervised learning to assign class labels, semi-supervised learning algorithms use a relatively small number of labeled instances to assign class information to the cluster identifiers, and therefore the unlabeled instances contained within that cluster. Providing an explicit example is rather difficult; however, studies, such as [2] and [3], have been conducted to determine if humans actually utilize semi-supervised learning presented in the machine learning context. In [3], the more rigorously executed study, Zhu et. al. presented 22 subjects with a two class categorization task of visually complex unrecognizable supershapes of which a select subset is presented in *Figure 1.1*. Each shape presented in *Figure 1.1* is produced using the same function evaluated using the value displayed below the shape. Supershapes, defined by the Superformula proposed by Geilis in [4], are continuously flowing shapes (i.e. they gradually morph from one state to another) and can be governed by one variable.

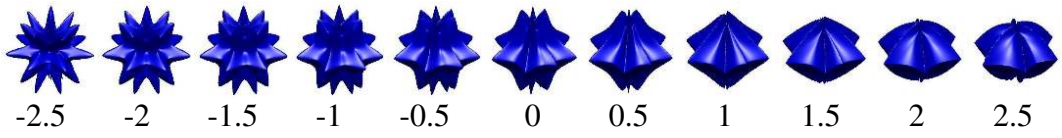

Figure 1.1. Example of supershapes
Supershapes morph from one state to another gradually. This transition can be parameterized by a single variable. Each shape pictured is produced by the same function evaluated using the value below the image. (figure obtained from [4])

The data were characterized by a bimodal Gaussian – each mode representing an opposing class. Subjects were given one sample from the center of each mode as training data, and were then asked to categorize a large set of additional instances. The subjects were divided into two groups: one received unlabeled data sampled from a similar bimodal distribution shifted to the left of the original Gaussian, and the other group was presented unlabeled data from a similar bimodal distribution shifted to the right. Results showed both groups developed initial decision boundaries near the middle of the two training instances until they were exposed to the shifted unlabeled data. Subjects from the left shifted distributions moved their decision boundary to the left while subjects from the right shift altered the decision boundary to the right. This experiment demonstrates that humans do in fact utilize a semi-supervised learning methodology.

1.1.2 Nonstationary environments. To make learning in any of these three categories more realistic to human cognition, we must add one of the most challenging aspects of the human brain to emulate – adapting to an environment that is constantly changing. Infants learn to distinguish their family members' faces in different lighting conditions even though they may not know their names; children under the age of ten are able to identify a speaker over the phone even with a poor connection or voice alterations due to illness; and adults make thousands of decisions daily while driving in various weather conditions or deciding to buy/sell shares in an ever fluctuating financial market.

In machine learning, the challenge of making decisions in a changing environment is referred to as nonstationary learning. Nonstationary learning is extremely challenging since it requires algorithms to maintain a delicate balance of retaining relevant knowledge and forgetting concepts that are no longer applicable. In machine

learning, this challenging balance is known as the stability vs. plasticity dilemma [5]: stability refers to the ability to retain previously acquired knowledge making a stable learning environment; whereas plasticity refers to the ability of the classifier to adapt to new concepts, and acquire new knowledge.

Once again machine learning approaches emulate humans' decision making processes of i) using pooled experiences; or in some cases ii) recalling only their most recent experience. For example, when deciding whether a particular meal is enjoyable, a person relying on pooled experiences may recall several (or sometimes all) occasions they have tasted that dish before making a decision. The collection of experiences may include positive and negative feelings toward the meal, but in the end an overall decision is made to either like or dislike the dish. In machine learning, ensemble systems use this same decision making construct. Ensembles used in nonstationary environments are a collection of classifiers that are typically constructed at different periods in time; each classifier containing information about the state of the environment at the time it was constructed. Combining the classifiers' knowledge produces a final collective decision of the ensemble. Each classifier's vote in the final decision can be weighted, giving more influence to recent classifiers, as they are most likely to represent the current state of knowledge on the environment. Returning to our meal example, a person's taste buds change every few years, so an experience in recent months should have more impact than a meal seven years prior.

Conversely, another person may allow only the most recent food encounter, good or bad, to sway their opinion of the meal. Eating a dish that causes gastrointestinal discomfort may prevent one from eating that dish in the future. In machine learning, this

is similar to a single classifier system; they are updated to incorporate new information reflecting the change in the environment. Single classifier systems are managed in through incremental updates, adding the most recent experience to a single classifiers decision making ability, or by completely reconstructing a new classifier each time a change is detected.

Ensemble systems and incremental learners have both advantages and disadvantages and selecting the appropriate style of learner is largely application dependent. Examples of each variety are discussed in more depth in *Section 0*.

1.2 Problem Statement

A fundamental assumption made by most learning algorithms is that data are drawn from a fixed but unknown distribution. This assumption implies that future unlabeled instances the model is expected to classify come from the same distribution as the data on which the model was developed in the first place. The previous section presented a few scenarios that contradict this static distribution scenario; in fact, many real world machine learning applications involve evolving surroundings (e.g. cancer detection, weather predictions, web ad placement, etc.).

Nonstationary environments present a challenging problem for all machine learning algorithms. However, the benefit gained from tracking environments using unsupervised methods is limited – most applications require explicit class information be related rather than a cluster identifier. Therefore, most nonstationary learning research utilizes supervised or semi-supervised algorithms. A majority of research conducted has used supervised learners and has produced methods proven to be very effective at

learning in and adapting to changing environments [6]–[17]. However, supervised learning algorithms' dependence on large sets of labeled examples for training has two drawbacks – labeled data are expensive and time consuming to obtain, as they require human annotation. When working in a nonstationary environment, where data often arrive as a stream, taking time to gather large sets of labeled examples is often impractical. For this reason, semi-supervised learning algorithms have been gaining increasing attention for nonstationary learning applications. The reliance of semi-supervised learners on relatively small sets of labeled data paired with their ability to utilize cluster information available from abundant, inexpensive, readily available unlabeled instances makes semi-supervised learning very attractive for nonstationary applications.

Most semi-supervised approaches to learning in non-stationary environments, for which a summary of relevant work is provided in *Chapter 3*, often assume that labeled data are available with every batch of incoming data. However, more recent research, typically referenced as verification latency, has added an important and practical constraint: labeled data are not available at every time step, nor even in regular intervals, which significantly complicates the learning process. Verification latency, as denoted by Marrs et. al. [18], 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.

This thesis searches for a solution to the problem of learning concepts from nonstationary environments in a cost effective and time efficient manner. The next

section narrows the scope of the thesis providing the constraints considered when implementing a solution to this problem.

1.3 Scope of Thesis

This thesis explores non-stationary data in an extreme verification latency scenario, where the lag duration is set to infinity – meaning no labeled data is ever received after initialization. We refer to this scenario as initially labeled streaming environment (ILSE), and propose a framework for learning in such an environment. A theoretically justified solution to this extreme learning environment can then provide effective algorithms for learning from environments that do not receive labeled data for extended periods of time, whether that period is finite or otherwise. Real-world examples of such an extreme learning setting are perhaps few today, but are rapidly growing due to massive automated and autonomous acquisition of sensor, web user, weather, financial transaction, energy usage, and other data. Furthermore, such applications can be extremely important: 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, because the data – which arrive continuously (i.e., streaming) – need to be isolated on a virtual machine, features need to be extracted from the header data, and then evaluated by a human expert. Many automation applications provide other examples, such as robots, drones, and autonomous vehicles encountering surrounding environment changing at a pace too quick for a human to verify all actions.

1.4 Organization of Thesis

Chapter 2 provides background of topics that have motivated this research – primarily semi-supervised learning, nonstationary learning, and verification latency. *Chapter 3* outlines the current state of knowledge in the field through a literature review on those topics that motive this research. *Chapter 4* introduces and explains the methodology of the COMPOSE algorithm developed for this thesis. *Chapter 5* presents the experimental setup and results of experiments on synthetic and real world data, followed by a discussion of the results. *Chapter 6* presents a summary of conclusions and suggestions for future work. Finally, the contributions this thesis has made to machine learning are summarized in *Chapter 7*.

Chapter 2

Background

This chapter provides background on the individual topics that motivated this research. A general overview of semi-supervised learning methodology, nonstationary learning approaches, and concerns with verification latency are presented.

2.1 Semi-Supervised Learning

Semi-supervised learning is a combination of unsupervised and supervised learning methods. It offers an advantage of reduced cost through limited use of labeled data, as obtaining labeled data is often costly and time consuming. Semi-supervised learning is rationalized in two ways: unsupervised learning with additional constraints (i.e., labeled data); or conversely, supervised learning with additional information provided (i.e., unlabeled data) [19]. These differing views ultimately achieve the same result; however, considering both perspectives can be helpful when considering the fundamental assumptions of semi-supervised learning and reviewing semi-supervised algorithms.

One or more of the four general assumptions listed below are utilized by semi-supervised learning algorithms [19], [20]:

i) the smoothness or local consistency assumption - if instances in a high density region are close to each other with respect to some similarity or distance measure, their class labels should be similar, while instances in a low density region need not belong to the same class.

ii) the cluster or global consistency assumption - instances in the same cluster should belong to the same class.

iii) the low-density separation assumption - decision boundaries should lie in low-density regions.

iv) the manifold assumption - high dimensional data reside on a lower dimensional manifold.

The first three assumptions are often combined to produce a more general definition of semi supervised learning that assumes class boundaries to reside where data are least dense, and the transition between classes should be gradual. The manifold assumption addresses a well-known problem in all of machine learning and statistics – the curse of dimensionality. When dimensionality increases linearly, volume of the feature space increases exponentially; therefore, more instances are required to adequately populate the feature space. Many learning applications do not have enough data to populate a high dimensional space, making learning difficult. By projecting the high dimensional data onto a lower dimensional manifold, the remaining three assumptions can be enforced in the lower dimensions, thus making learning feasible. Illustrating the manifold assumption in high dimensionality is difficult; however, a reduction from a three dimensional to one dimensional feature space is shown in *Figure 2.1*. The two distributions, represented with red and blue labeled data and black unlabeled data, in (a) are projected "downward" onto the f_1 and f_2 plane to produce the distribution in (b); then this distribution is projected "downward" again onto the f_1 axis. The result is a lower dimensional feature set that can then be analyzed using the other three assumptions to determine a decision boundary. It is important to note that not every manifold is created using an orthogonal basis, nor each

manifold produces a learnable reduced dimensionality dataset. If the data from (b) had been projected onto the f_2 axis instead of f_1 the resultant dataset would have been substantially more difficult to learn, if not impossible. There have been several techniques proposed to produce "optimal" manifolds. The most well-known and commonly used approaches are principle component analysis, independent component analysis, canonical correlation analysis, and Fisher's linear discriminant. In some of these methods the original features are combined to produce a new representative feature set in a lower dimension.

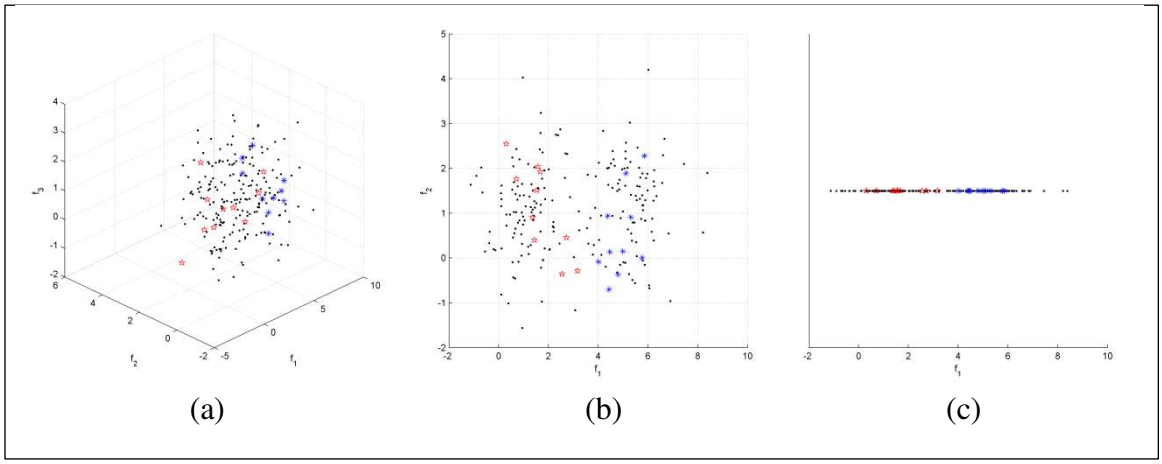

Figure 2.1. Example of manifold assumption
A manifold is a projection of higher dimensions in to lower dimensions – this method is common in semi-supervised learning where sufficient data may not be available to adequately populate the feature space. The two distributions in (a), represented by red and blue labeled data and black unlabeled data, are projected onto the $f_1 f_2$ plane producing (b). The dimensionality is reduced further by projecting the data in (b) onto the f_1 axis producing (c). The order and direction of the projections impact the end result greatly. If (b) had been projected onto the f_2 axis instead the data may not be separable.

Regardless of the assumptions utilized, all semi-supervised algorithms rely on some variation of a common iterative recipe: 1) train a classifier from available labeled data, 2) classify the remaining unlabeled data, 3) add instances whose confidence exceeds a threshold to the permanently labeled training set, and 4) remove instances that do not meet this threshold. This process has produced several well-established semi-supervised algorithms, primarily for use in static environments, which typically fall into one of three general categories:

i) generative algorithms, such as [21], [22], which assume that the data are provided by a fixed yet unknown distribution, and that the decision boundaries can be represented based on class posteriors;

ii) low-density separation algorithms, such as [23], [24], which use density information from unlabeled instances to modify a decision boundary created by using only labeled data;

iii) graph-based algorithms, such as [25], [26], which construct a graph, $G = (V, E)$ with vertices, V, representing instances and edges, E, representing relationships between vertices. Class information is transferred from labeled instances to neighboring unlabeled instances based on the relationship defined by the connecting edges.

Some semi-supervised algorithms developed for static environments have recently been modified or and are included a wrapper-based approach enabling them to work in nonstationary environments; these approaches are discussed in the literature review featured in *Chapter 3*.

2.2 Nonstationary Environments

Environments that provide data with changing distributions over time, such that $p^t(\boldsymbol{x}, y) \neq p^{t+1}(\boldsymbol{x}, y)$, are referred to as nonstationary environments. Here $\boldsymbol{x} \in X$ is an instance from the feature space X, belonging to the class (concept) $y \in Y$ from the class space Y, at time stamp t. The components of the distribution that differ between each time step can be categorized into four scenarios, listed below and depicted in *Figure 2.2*, all of which may occur independently or simultaneously:

i) the number of instances per class – class priors, $p(y)$

ii) the shape of the distribution – class-conditional, $p(\boldsymbol{x}|y)$, or sample distribution, $p(\boldsymbol{x})$

iii) the class assignment – posterior distributions of class membership, $p(y|\boldsymbol{x})$

iv) the addition/subtraction of a class – number of target concepts, $|Y|$

A significant body of research has focused on various combinations of the first three scenarios – known as concept drift – limiting the environment to fixed number classes (concepts). In this thesis, the fourth scenario is also addressed so the all-encompassing term nonstationary environment is used throughout.

Early work on learning in nonstationary environments has primarily been on defining the problem, and identifying types of nonstationary environments that may be learned [16], [27]–[29]. This is not trivial, as each of the aforementioned 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 [16], [30], [31].

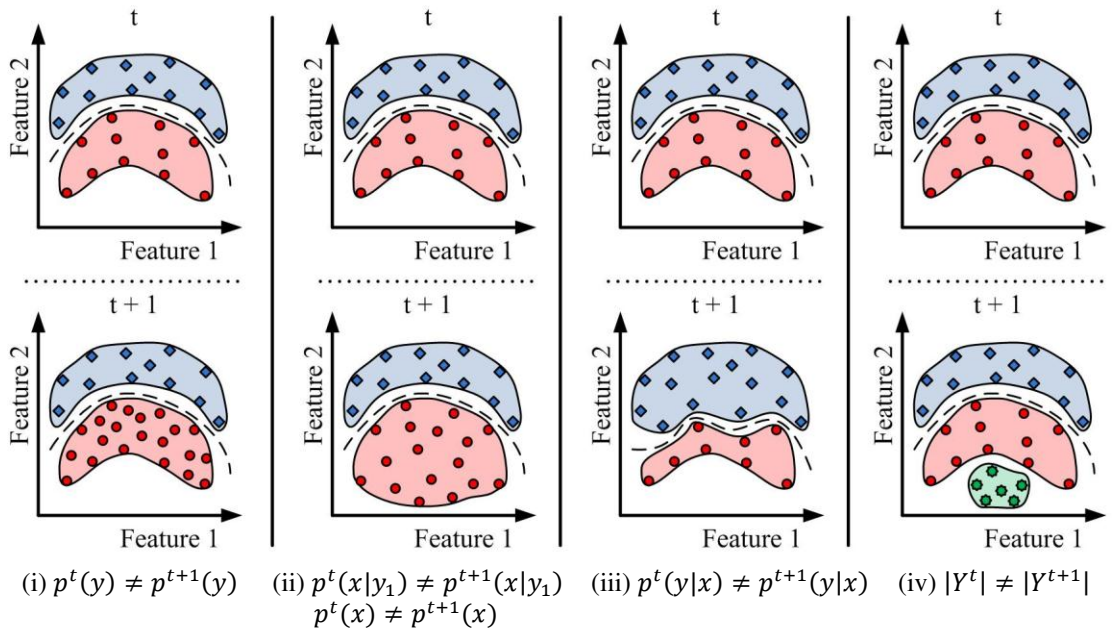

Figure 2.2. Types of change in nonstationary environments
(i) the class priors change between time steps; (ii) the class-conditional or sample distributions change between time steps; (iii) the posterior distributions of class membership change between time steps; (iv) the number of target classes (concepts) is changed through addition or deletion of a class (concept)

Nonstationary learning algorithms can be characterized in several ways, such as online vs. batch approaches; single classifier vs. ensemble-based approaches; or active approaches (explicitly seeking to determine when a change/drift has occurred before taking corrective action) vs. passive approaches [9] (assuming drift may occur at any time, and update a model every time new data arrive).

2.2.1 Online vs. batch approaches. Nonstationary data are presented in a stream – a time controlled progression of data – usually in one of two formats: online, where a single instance is available at each time step requiring a learner to adapt as each instance is acquired; or batch, where several instances are accumulated from the stream then presented to the learner. Both formats are depicted in *Figure 2.3* with periods in time

annotated for discussion. In an online setting each instance (star) is received and processed as acquired; however, the batch method waits until a block of instances (four, in the example illustrated in *Figure 2.3*) are received before these instances are presented to the learner (batches are divided by vertical dashed lines). At discussion point (a) we see what is clearly an outlier from the batch view; however when viewed from the online perspective it is exceedingly difficult to determine if this is an outlier or change in concept. At discussion point (b), we see a similar case from the batch perspective; one instance appears to be an outlier even though it is truly the start of a change in concept. An often made assumption, although rarely true, is concept change does not occur within a batch. As a result, batch learners often lag in reacting to changing concepts whereas online learners are able to react much faster to a change. At discussion point (c) we find the rare occurrence where batch learning does not lag behind an online learner and instead has a distinct advantage; the concept change occurs between batches instead of within a batch as in (b). These three discussion points illustrate why online learning is considered to be substantially more difficult than batch learning - less data make concept generalization more difficult. Sometimes an incremental learning constraint is imposed making nonstationary learning even more difficult. Incremental learning dictates previously seen data are not accessible after the learner has initially seen the data. This additional assumption is shown in *Figure 2.3* as gray shading over the previously encountered instances.

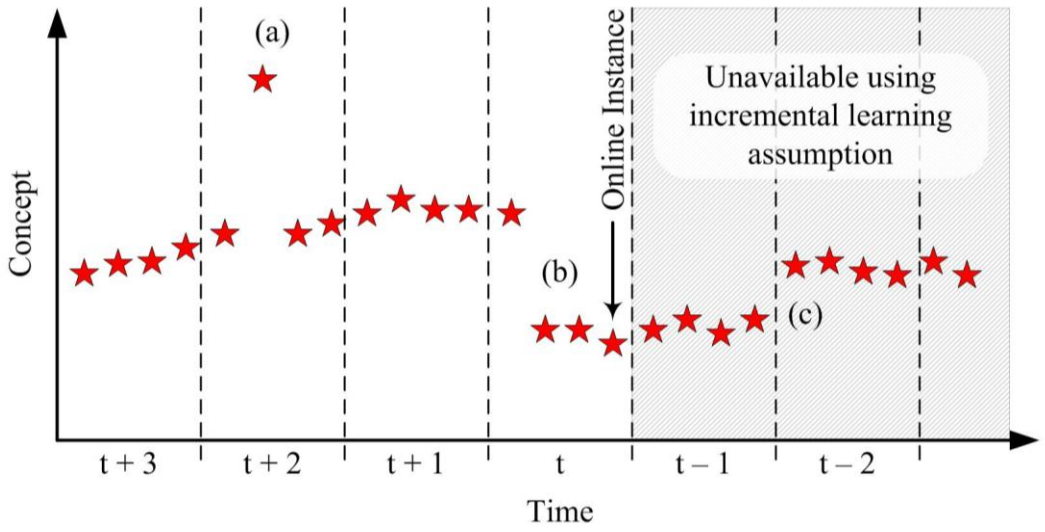

Figure 2.3. Online vs. batch nonstationary streaming data
Depicts special discussion points for the comparison of online and batch data formats. In discussion point: (a) an outlier that would be easily recognized by batch learner may be considered a concept change to an online learner; (b) the batch assumption "change does not occur within a batch" delays the batch learner from realizing the concept change until the next batch; (c) the batch learner has a clear advantage over online learners.

2.2.2 Active vs. passive approaches. Active approaches determine when a change has occurred before taking corrective action to update the learner, whereas passive approaches assume drift may occur at any time, and update the model every time new data arrive. Active nonstationary learning algorithms include window based approaches, such as STAGGER [27] and FLORA [16], and their variants [32]–[37], which use a sliding window to choose a block of new data to train a new classifier when change is detected. Other approaches use control charts to detect drift, including Alippi and Roveri's just-in-time (JIT) classifiers [6], [38], [39], and the more recent intersection of confidence intervals (ICI) rule [40] are examples of such approaches. Information theoretic measures [41]–[43], Hoeffding bounds or Hellinger distance [44], [45] of

individual features have also been used for detecting drift and updating a classifier [41], [42], [46].

2.2.3 Single vs. ensemble approaches. Many nonstationary learning algorithms are single-classifier approaches, which typically adapt to change by either: i) updating the adjustable parameters of the classifier to reflect changes present in newly received data [34], [47], [48]; or ii) replacing the current classifier with a new classifier trained on newly received data. Both suffer from the stability-plasticity dilemma [5]. Stability is required to retain previous knowledge but too much stability hinders learning new concepts. Plasticity, on the other hand, allows new information to be readily learned but too much plasticity results in previously acquired knowledge being forgotten too quickly. Algorithms strive to balance stability and plasticity. A learner that is entirely stable would not adapt to changes in the environment and a learner that is entirely plastic is plagued with catastrophic forgetting [49] – no previous knowledge is ever retained. While non-stationary learning is possible with fully plastic learners, adding in stability often increases performance.

Ensemble based approaches use a combination of several classifiers to make a decision, hence avoiding stability-plasticity problems, albeit at increased computational cost. Combining decisions of several classifiers, often created at different time steps, provides a natural mechanism to update the collective knowledge of the ensemble. Classifiers are added, removed, or updated to provide a better balance of stability vs. plasticity. Ensemble approaches track the environment by adding new (and possibly removing old) classifiers to build an ensemble of classifiers with each incoming dataset. These approaches typically use a passive drift detection and a fixed ensemble size, where

the oldest member (as in Street's Streaming Ensemble Algorithm [14], and Bifet's adaptive Hoeffding tree bagging [50]) or the least contributing ensemble member (as in Kolter's Dynamic Weighted Majority (DWM) [51]) is replaced with a new one. Voting is the most common approach for combining the classifiers, though there is disagreement on whether a weighted [15] or simple majority voting should be used [52]. Hybrid approaches that combine active detection, sliding window and ensembles have also been proposed, such as in Abdulsalam et al.'s random forests with entropy [43], Masud et al.'s concept drift with time constraints [53], He et al.'s IMORL and ADAIN [10], [54], and Bifet's integration of a Kalman filter with Adaptive Sliding Window (ADWIN) [7], [55], part of his Massive Online Analysis (MOA) suite [56], which also includes Learn^{++}.NSE [9], [57], [58] for mining data streams with concept drift.

2.3 Verification Latency

Verification latency, as first defined by Marrs et. al. [18], describes a scenario where true class labels are not available until sometime after the classifier has made a prediction on the current 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.

Verification latency is a problem that plagues an increasing number of real-world nonstationary learning environments (e.g. credit card fraud, autonomous drone navigation, medical diagnosis, etc.), but is often disregarded in research due to its complexity. In most nonstationary learning problems, drift is assumed to be limited or gradual, and labeled data are assumed to arrive with every batch of incoming data.

Regular availability of labeled data and assumptions of relatively small shifts in the underlying concepts allows verification latency effects to be ignored in most research. However, when underlying distributions change rapidly, or access to labeled data is restricted, latency in model verification becomes drastically more important.

To illustrate this importance, let us consider a slowly evolving cancer and compare it to a credit card fraud situation. Cancer detection often relies on several markers to indicate the presence of cancer. In a slowly evolving cancer the thresholds that indicate cancer will slowly fluctuate, and these changes can be documented as each new possible cancer detected is evaluated and biopsied. The time taken to biopsy and denote changes in the markers introduces latency but since the system is slowly changing the delay is not devastating to classifier performance. In the case of credit card fraud, most transactions are normal and the classifiers monitoring the credit accounts learn our purchasing habits. When a fraudulent transaction occurs, it can go unnoticed for up to a month when the billing cycle closes and the balance is sent to the user. In this case a rapid change in purchasing may go undetected for several days, during which extensive damage can be done. Latency in identifying the difference between a fraudulent and normal transaction has had detrimental impact on the overall system.

The consequences of verification latency have been circumvented by applying (often) unrealistic assumptions to the environment (i.e. the regular availability of labeled data and small shifts in concepts as mentioned above). However, there have been several attempts to start relaxing some of these assumptions [59 - 65] which are discussed more thoroughly in the literature review in *Chapter 3*.

Chapter 3

Literature Review

This chapter highlights algorithms utilizing semi-supervised learning in non-stationary environments relevant in the development of this work. A brief summary of each algorithm is presented with a focus on the following criteria:

- Types of learners utilized
- Limitations of tracking different types of non-stationary environments
- Required frequency of labeled data

3.1 Recurring Concept Drifts From Limited Labeled Streaming Data (REDLLA)

Li et. al [59] propose REDLLA to explore REcurring concept Drifts from Limited LAbeled streaming data. Recurring concepts are difficult to address due to the stability plasticity dilemma [5] – one must retain old knowledge that is still relevant, yet replace obsolete knowledge to adapt to new concepts. To address this recurring concept problem, REDLLA maintains a decision tree along with a table of previously seen concepts. The algorithm assumes data arrive in batches of mixed labeled and unlabeled instances at every time step. The algorithm has been shown effective with 10% of the instances arriving with labels.

REDLLA constructs a decision tree on receipt of the first batch of data (step 1 in *Figure 3.1*). Each instance in every subsequent batch is sorted through the tree and grouped at the appropriate leaf (step 2 in *Figure 3.1*). Each instance grouped at a specific leaf increases the instance count of that leaf, n_L, while the instance features are added to an attribute array, $attrArray$, and if labeled, its class is recorded into a class

array, $classArray$ (steps 3-5 in *Figure 3.1*). In every leaf, if the instance count, n_L, exceeds a user defined threshold, n_{min}, the unlabeled instances are labeled using a k-means clustering algorithm with simple majority voting of labeled instances placed in the same cluster, where k is set equal to the number of different classes present in $classArray$ (step 6 in *Figure 3.1*). The k-means clustering algorithm uses a distance metric to partition all instances, both labeled and unlabeled, into k different clusters around the closest mean, where the number of means is a predetermined value, k. The means can be provided by the user or can be determined through an iterative process. When simple majority voting is utilized, the labeled instances found within a cluster are tallied and the class with most instances is then assigned to all instances in the cluster, even if previously labeled. The results of the simple majority vote are used to update the $classArray$ (step 7 in *Figure 3.1*). After all instances have been labeled, split tests are conducted using information gain criteria, an impurity based method using entropy measures explained in [34], and new leaves are grown while the current leave becomes a decision node (step 8 in *Figure 3.1*).

To determine if the tree has encountered any recurring concepts, the algorithm performs a check at a user defined detection period interval, DP (in *Figure 3.1*) this check is expressed as $|E| \,\%\, DP = 0$ where $\%$ is the modulus operator and $|E|$ is the number of instances received in the data stream). For every leaf in the current tree, the radius, $r_{c,new}$, of each cluster, c is calculated as the averaged Euclidean distance of every instance to its cluster mean, m_c. The cluster mean and radius are recorded into a temporary array, $M_{new} = \{r_c, m_c\}$. If the leaf was newly created during the last split test, array $M_{last} = M_{new}$ and this concept is added to table, $conceptList$ (steps 9-11 in *Figure*

3.1). If the leaf existed during the previous detection period, the Euclidean distance, d_c, between the means of similar classes in M_{new} and M_{last} are calculated, where M_{last} is the cluster information from the last detection period. The clusters in M_{new} are then evaluated and placed into one of three categories: i) potential drift, where $0 \leq d_c \leq max(r_{c,last}, r_{c,new})$, and the previous concepts are updated to reflect the minute drift, $M_{last} = M_{new}$; ii) noise artifacts, where $max(r_{c,last}, r_{c,new}) < d_c < r_{c,last} + r_{c,new}$, and M_{new} is discarded so that the next detection period uses the current M_{last} for comparison purposes; or iii) true drift, where $d_c \geq r_{c,last} + r_{c,new}$. When true drift is detected, $M_{last} = M_{new}$ and M_{new} is added to $concepList$ only if there is no other "similar" concepts in $conceptList$. "Similarity" is determined using the same Euclidean distance metric described above (steps 12-15 in *Figure 3.1*).

To ensure the tree does not over fit, pruning is conducted at a user defined pruning period interval, PP. Pruning is conducted using a bottom up error based approach to remove leaves with an error rate greater than 50%. The tree performance is calculated at a predefined user incremental output period, OP, where the performance is calculated using accumulated sum of a user defined loss function between predicted and observed values.

Inputs: Stream of data: E; Minimum number of split-examples: n_{min} ; Detection Period: DP; Pruning Period: PP; Incremental Output Period: OP.

1. Create a leaf for tree, T

Do for each instance $e \in E$

| 2. Sort e into available leaf, L.
| 3. Increase count of instances sorted to leaf, n_L
| 4. Add features of e to $attrArray$
| **If** e is labeled
| | 5. Add class of e to $classArray$
| **End If**

End

Do for each leaf $L \in T$

| **If** $n_L \geq n_{min}$
| | 6. Label the unlabeled instances in leaf using k-Means clustering and simple majority voting of labeled instances contained within the cluster. k is set equal to number of classes present in $classArray$.
| | 7. Update $classArray$ to reflect results of k-Means cluster and label for previously unlabeled instances.
| | 8. Conduct split-test and grow new children leaves
| **End If**
| **If** $|E| \,\%\, DP = 0$
| | 9. Calculate radius, r_c, of each cluster, c, where the radius is the averaged Euclidean distance of each instance to the cluster mean, m_c.
| | 10. Create array $M_{new} = \{r_c, m_c\}$
| | **If** L is a new leaf (i.e. created in 8.)
| | | 11. Create array $M_{last} = M_{new}$ and place M_{last} into array $conceptList$
| | **Else**
| | | 12. Calculate Euclidean distance, d_c, between means of similar classes in M_{new} and M_{last}, where M_{last} is the an array containing the radius and mean of each cluster from the previous detection period.
| | | **Select**
| | | Case 1: $0 \leq d_c \leq \max(r_{c,last}, r_{c,new})$ → Potential Drift
| | | | 13. Update model to reflect minor changes, $M_{last} = M_{new}$
| | | Case 2: $\max(r_{c,last}, r_{c,new}) < d_c < r_{c,last} + r_{c,new}$ → Noise Artifact
| | | | 14. Discard M_{new} (i.e. do nothing so M_{new} will be overwritten)
| | | Case 3: $d_c \geq r_{c,last} + r_{c,new}$ → True Drift
| | | | 15. Compare M_{new} to M_{hist} using the same distance metric, d_c, where M_{hist} is a list of all the previous concepts in $conceptList$. If no matching concept is found M_{new} is added to $conceptList$ as a new concept and $M_{last} = M_{new}$
| | | **End Select**
| | **End If**
| **End If**
| **If** $|E| \,\%\, PP = 0$
| | 16. Conduct bottom up error based pruning of branches with error rate greater than 50%
| **End If**
| **If** $|E| \,\%\, OP = 0$
| | 17. The performance for current model using an accumulated sum of loss function between predicted and observed values.
| **End If**

End

Figure 3.1. REDLLA pseudocode

3.2 Weight Estimation Algorithm (WEA)

Diztler et. al [60] propose WEA, a Weight Estimation Algorithm to learn nonstationary concepts in streaming data using any supervised learning algorithm as the base classifier for a learning ensemble. WEA assumes data arrive in a batch format of labeled data followed by unlabeled data. The unlabeled data are assumed to (possibly) originate from a drifted distribution (i.e. labeled and unlabeled data are from different distributions).

WEA, psuedocode presented in *Figure 3.2*, works iteratively; adding to the ensemble, as new data arrive. At each time step, t, WEA trains a fully supervised **BaseClassifier** on the available labeled data, and then constructs a Gaussian Mixture Model (GMM), $\mathcal{M}_c^t$, with a user defined number of components, K_c, for each class, c, in the labeled data (steps 1 and 2 in *Figure 3.2*). When unlabeled data are received, possibly from a drifted distribution, a second GMM, $\mathcal{N}^t$, is constructed with its number of components totaling the sum of the all components in $\mathcal{M}_c^t$ (step 3 in *Figure 3.2*). The Bhattacharyya distance between each component in $\mathcal{N}^t$ and each component in $\mathcal{M}_c^t$ is calculated, and the label of the closest component in $\mathcal{M}_c^t$ is assigned producing a labeled GMM of the unlabeled data, $\mathcal{N}_c^t$ (step 4 in *Figure 3.2*). The Bhattacharyya distance is used as the distance metric since this paper defines its limited drift assumption to be the Bhattacharyya distance between a known component and its future position must be less than the Bhattacharyya distance between the known component and any other future component of a differing class. A user defined number, q^t, of synthetic samples are drawn from the now labeled GMM, $\mathcal{N}_c^t$. These synthetic instances are used to compute the error of each classifier in the ensemble. If the error exceeds 50% incorrect

classification the error is set to 50% (steps 5 and 6 in *Figure 3.2*). The classifier weights, which are proportional to the calculated error, are determined and used to produce a weighted majority ensemble hypothesis on the unlabeled data (steps 7 and 8 in *Figure 3.2*).

WEA was tested on synthetic data and compared to a similar ensemble algorithm Learn^{++}.NSE, which only utilizes labeled data. The results demonstrated comparable performance between the two algorithms when the labeled and unlabeled data were drawn from a slowly drifting distribution. However, as the drift increased, WEA performed significantly better than Learn^{++}.NSE. When drift became too great and violated the Bhattacharyya distance limited drift assumption, WEA's performance dropped significantly.

Inputs: Labeled training data $\mathcal{L}^t = \{\boldsymbol{x}_i \in \mathcal{X}; y_i \in \mathcal{Y}\}$ where $i = 1, \dots, m^t$;

Unlabeled data $\mathcal{U}^t = \{\boldsymbol{x}_i \in \widehat{\mathcal{X}}\}$ where $i = 1, \dots, n^t$;

K_c – number of centers for the cth class in a GMM;

q^t – number of instances generated to estimate classifier error;

BaseClassifier learning algorithm

Do for $t = 1,2, \dots$

1. Call **BaseClassifier** on $\mathcal{L}^t$ to generate hypothesis $h_t: \mathcal{X} \to \mathcal{Y}$
2. Generate GMM with K_c centers for each class present in $\mathcal{L}^t$, $\mathcal{M}_c^t = GMM(K_c, \mathcal{L}^t)$
3. Generate GMM with K_c centers from unlabeled data $\mathcal{U}^t$, $\mathcal{N}^t = GMM(\Sigma K_c, \mathcal{U}^t)$
4. Assign each component in $\mathcal{N}^t$ the label of the closest component in $\mathcal{M}_c^t$, where distance metric is the Bhattacharyya distance, $\mathcal{N}_c^t = Bhattacharyya(\mathcal{N}^t, \mathcal{M}_c^t)$
5. Generate q^t synthetic instances from $\mathcal{N}_c^t$ used to compute error, $\hat{\varepsilon}$, of each classifier.

$$\hat{\varepsilon}_k^t = \frac{1}{q^t}\sum_{l=1}^{q^t}[h_k(\boldsymbol{x}_l) = y_l] \text{ where } k = 1,2, \dots, t$$

If $\hat{\varepsilon}_k^t > 1/2$

6. Limit the error, $\hat{\varepsilon}_k^t = 1/2$

End If

7. Compute the classifier voting weights for the unlabeled data,

$$W_k^t \propto \log\frac{1 - \hat{\varepsilon}_k^t}{\hat{\varepsilon}_k^t}$$

8. Classify the unlabeled data in $\mathcal{U}^t$ using weights,

$$H^t(\boldsymbol{x}_j \in \mathcal{U}^t) = \arg\max_{c \in \Omega} \sum_{k=1}^{t} W_k^t[h_k(\boldsymbol{x}_j) = y_j]$$

where Ω is the set consisting of all classes in the problem

End

Figure 3.2. WEA pseudocode

3.3 Semisupervised Stream Clustering (SmSCluster)

Masud et. al. [61] propose an ensemble of clusters to track nonstationary concepts in streaming data when limited labeled data are available. This algorithm assumes both labeled and unlabeled data are available in every batch, and updates the ensemble to select the best preforming clusters to classify the most recent data. The SmSCluster process is outlined in the pseudocode presented in *Figure 3.3*.

At each timestamp, data are clustered into K user defined clusters by minimizing an impurity cost function through the Expectation-Maximization Algorithm [21] (step 1

in *Figure 3.3*). The impurity cost function is the sum of i) the Euclidean distance between each instance and the cluster centroid and ii) the Euclidean distance between labeled data and the cluster centroid scaled by an impurity measurement. The impurity measurement is the product of an aggregated dissimilarity count (ADC) and the entropy (Ent) of the cluster. The author defines the aggregated dissimilarity count as a tally of all labeled instances not belonging to the majority class of a cluster and uses a standard definition of entropy,

$$Ent_i = \sum_{c=1}^{C} -\frac{|\mathcal{L}_i(c)|}{|\mathcal{L}_i|} * log\left(\frac{|\mathcal{L}_i(c)|}{|\mathcal{L}_i|}\right),$$

where C is the number of classes and $\mathcal{L}_i$ is the labeled data in cluster i.

Once the clusters have been created, a model, M^t, for that timestamp is created containing a statistical summary of the K clusters formed (step 2 in *Figure 3.3*). The statistics recorded for each cluster are:

- the total number of instances: N
- the total number of labeled instances: Lt
- a vector with the total number of labeled instances in each class: $Lp[c]_{c=1}^{C}$
- the cluster centroid: $\boldsymbol{u}$
- a vector containing the sum of each dimension $r \in d$ of all cluster data: $Sum[r]_{r=1}^{d}$

These statistics must be recorded in order for future clusters to be merged when a new class is experienced. When a new class is introduced at the current time step, previous models have no knowledge of this new class, skewing the ensemble voting process. In order to overcome this problem the new class information is injected into previous models using the following process. In each model, the closest two classes

having the same majority class are determined and then combined to generate new statistics from their previous values (step 3 in *Figure 3.3*). A subset of data from the newly experienced class is then injected with a user defined probability ρ (step 4 in *Figure 3.3*). After each model has been injected with a random subset (if needed), the performance of each model is acquired by testing that model's classification rate on the labeled data from the current timestamp (step 5 in *Figure 3.3*). The m highest performing models are selected to the ensemble and then used to label the unlabeled data from the current timestamp.

The ensemble voting is executed in the following manner: for each unlabeled instance, the Q closest clusters are identified using a distance metric between unlabeled instance and cluster centroid. The normalized frequency, $\frac{Lp[c]}{Lt}$, obtained from the summary statistics are calculated and summed across the Q clusters. The unlabeled instance in then assigned the class of the highest cumulative normalized frequency.

Inputs: Data arriving at time t – $\mathcal{D}^t = \{x_i \in \mathcal{X}; y_i \in \mathcal{Y}\}$ and $\mathcal{Y} = \{\phi, 1, \ldots, C\}$ where $\phi = unlabeled$ and C is the total number of classes;

K – number of clusters to be created;
Q – number of nearest neighbors for kNN classification;
ρ – probability of injection;
m – number of models in ensemble

Do for $t = 1,2, \ldots$

1. Create K clusters using the E-M algorithm on the K-means with Minimization of Cluster Impurity cost function

$$Minimize: \sum_{i=1}^{K}\left(\sum_{\boldsymbol{x}\in\mathcal{X}_i}\|\boldsymbol{x}-\boldsymbol{u_i}\|^2 + \sum_{\boldsymbol{x}\in\mathcal{L}_i}\|\boldsymbol{x}-\boldsymbol{u_i}\|^2 * ADC_i * Ent_i\right)$$

where

u_i is the centroid of cluster i
$\mathcal{L}_i$ is the set of all labeled point in cluster i
ADC_i is the aggregated dissimilarity count of cluster i
Ent_i is the entropy of cluster i

2. Create a model $M^t \in M$ which contains summary of statistics for each created cluster in 1.

The statistics for each cluster M_i^t include:
N : the total number of points
Lt : the total number of labeled points
$Lp[c]_{c=1}^{C}$: a vector with the total number labeled points in each class
$\boldsymbol{u}$: the centroid of the cluster
$Sum[r]_{r=1}^{d}$: a vector containing the sum of each dimension $r \in d$ of all cluster data

If clusters in M^t contain a new class not in clusters in $M^j \in M$ where $j = 1, \ldots, m$

Do for $j = 1, \ldots, m$

3. Merge the closest two clusters having the same majority class in M^j
4. Injecting cluster M_i^t containing new class $\hat{c}$ into M^j with probability ρ.

End

End If

5. Test each model $M^j \in M$ and M^t on the labeled data in D^t and obtain its accuracy
6. $M \leftarrow$ best m models in $M \cup \{M^t\}$ based on accuracy
7. For all unlabeled data, $D^t\{\boldsymbol{x}; y = \phi\}$, find the Q nearest labeled clusters in M by computing the distance between the point and the centroid of the cluster.
8. Calculate the normalized frequency of each of the Q nearest clusters, $\frac{Lp[c]}{Lt}$
9. Sum the normalized frequencies of the Q nearest clusters and assign the data point the class label of the highest cumulative normalized frequency.

End

Figure 3.3. SmSCluster psuedocode

3.4 Relational K-means Transfer Semi-Supervised Support Vector Machine

Zhang et. al. [62] identify four types of data in nonstationary streams involving mixed labeled and unlabeled data: labeled data (Type I) and unlabeled data (Type III) from the same distributions as the next-to-arrive batch of data; and labeled (Type II) and unlabeled (Type IV) data from similar distributions as the next-to-arrive data batch. Zhang et. al. propose a Transfer Semi-Supervised Support Vector Machine (TS^{3}VM) model to learn data types I, II, and III and a relational k-means (RK) based model to learn Type IV data. They proceed to combine the two models together producing RK-TS^{3}VM for learning from nonstationary streaming data with labeled and unlabeled instances.

The TS^{3}VM model is formulated by incrementally incorporating type I, II, and III data. Learning from type I data, $T_1 = \left\{(\boldsymbol{x}_i, y_i)|\boldsymbol{x}_i \in \mathbb{R}^d, y_i \in \{-1,1\}\right\}_{i=1}^{L_1}$, where $\boldsymbol{x}_i$ and y_i are the feature vector and class label, respectively, of the i^{th}instance in a d dimensional set of L_1 instances, is achieved by training a generic semi-supervised support vector machine (SVM) model where the margin is maximized between classes and the misclassification rates are minimized given in Equation 3.1:

$$min_\theta \ \frac{1}{2}\|\boldsymbol{w}\|^2 + C\sum_{i=1}^{L_1} H\left(y_i f_\theta(\boldsymbol{x}_i)\right) \tag{3.1}$$

where w is the projection direction, C is the penalty of instances inside the margin, $H(t) = max(0, 1-t)$ is the hinge loss function, the function $f_\theta(x) = (wx + b)$, $\theta = (w, b)$ is the classification boundary.

To incorporate type II data, $T_2 = \left\{(\boldsymbol{x}_i, y_i)|\boldsymbol{x}_i \in \mathbb{R}^d, y_i \in \{-1,1\}\right\}_{i=L_1+1}^{L_2}$, where L_2 indicates the number of type two instances, into the SVM model a multitask learning approach is taken. In multi-task learning two objectives are optimized simultaneously,

but are controlled by weights; a greater weight indicates preference in task optimization. For this two task problem, labeled data from both same and similar distributions, the multi-task learning objective is given in Equation 3.2:

$$min_\theta \ \frac{1}{2}\|\boldsymbol{w}\|^2 + C_1\|\boldsymbol{v}_1\|^2 + C_2\|\boldsymbol{v}_2\|^2 + C\sum_{i=1}^{L_2} H\big(y_i f_\theta(\boldsymbol{x}_i)\big) \quad (3.2)$$

Where C_1 and C_2 are the weights controlling task preference, v_1 and v_2 are discrepancies between the global optimal decision boundary w and the decision boundary for each local task, $f_\theta(x) = (\boldsymbol{w} + \boldsymbol{v}_1)\boldsymbol{x}_i + \boldsymbol{b}$ for $1 \le i \le L_1$ and $f_\theta(x) = (\boldsymbol{w} + \boldsymbol{v}_2)\boldsymbol{x}_i + \boldsymbol{b}$ for $L_1 + 1 \le i \le L_2$, and $\theta = (\boldsymbol{w}, \boldsymbol{v}_1, \boldsymbol{v}_2, \boldsymbol{b})$.

When incorporating type III data, $T_3 = \{(\boldsymbol{x}_i)|\boldsymbol{x}_i \in \mathbb{R}^d\}_{i=L_2+1}^{U}$, the SVM must consider unlabeled data, which is accomplished by modifying the hinge loss function to be a symmetric hinge loss function [63]. A symmetric hinge loss function simply requires the absolute value of the data be taken since no class information is available and instances be penalized for residing inside the margin. The updated semi-supervised SVM is shown in Equation 3.3:

$$min_\theta \begin{array}{c} \frac{1}{2}\|\boldsymbol{w}\|^2 + C_1\|v_1\|^2 + C_2\|v_2\|^2 + \\ C\sum_{i=1}^{L_2} H\big(y_i f_\theta(\boldsymbol{x}_i)\big) + C^*\sum_{i=L_2+1}^{U} H(|f_\theta(\boldsymbol{x}_i)|) \end{array} \quad (3.3)$$

Where C^* is the penalty of unlabeled instance residing inside the margin, $\theta = (\boldsymbol{w}, \boldsymbol{v}_1, \boldsymbol{v}_2, \boldsymbol{b})$, and $f_\theta(x_i) = (\boldsymbol{w} + \boldsymbol{v}_1)\boldsymbol{x}_i + \boldsymbol{b}$ for $1 \le i \le L_1$, $f_\theta(x_i) = (\boldsymbol{w} + \boldsymbol{v}_2)\boldsymbol{x}_i + \boldsymbol{b}$ for $L_1 + 1 \le i \le L_2$, $f_\theta(x) = \boldsymbol{w}\boldsymbol{x}_i + \boldsymbol{b}$ for $L_2 + 1 \le i \le L + U$, $L = L_1 + L_2$ and U is the number of unlabeled instances. When working with the unlabeled data, there is a possibility that all unlabeled instances are assigned to one class with a very large margin, often this is an error and leads to poor performance. To rectify this potential problem, the author adds a balance constraint to Equation 3.3 stating the objective function is to be

minimized such that $\frac{1}{U}\sum_{i=L_2+1}^{U} f_\theta(\boldsymbol{x}_i) = \frac{1}{L_2}\sum_{i=1}^{L_2} y_i$. This additional balance constraint estimates the class ratios from the labeled data in T_1 and T_2.

The author has incorporated data types I – III into the TS^{3}VM model, however type IV data become much more difficult since they are unlabeled data from a different distribution than the target domain. To overcome this difficulty a relational k-means clustering model (RK) is devised. The Type IV data are grouped into clusters using k-means clustering algorithm then the similarity between each cluster center and Type I data is calculated using a Euclidean distance.

The combined models result in the RK-TS^{3}VM algorithm which works as follows. When a new batch of data arrives identify the four types of data according to the labeled rate and the concept drift probability (both provided by user). The author assumes Type I and III data are the calculated percentage of instances located at the tail of the batch (most recent data generated) and Type II and IV are at the remaining instances at the head of the batch (oldest data in batch). Using the RK model, cluster centers of Type IV data are obtained. For each Type I, II, and III instances cluster center attributes are added by taking the inner product of the cluster centroid's features with the instances features. The new instances generated from the inner product are then used to construct the TS^{3}VM model - this model is used for prediction.

3.5 The Ensemble Classifier and Clusters Model

Zhang et. al. [64] propose The Ensemble Classifier and Clusters Model, which is able to learn nonstationary streaming concepts from batches that do not provide labeled data in all time steps; however, the algorithm does require labeled data periodically to

properly update the ensemble. The ensemble maintains n base models for which the model may be a classifier or a cluster depending whether the most recent batch contains labeled data. The ensemble contains a classifier models, $\lambda^1, \ldots, \lambda^a$, and b cluster models, $\lambda^{a+1}, \ldots, \lambda^n$; $a + b = n$. The objective of the ensemble E is to provide a class label, $y \in Y = \{c1, \ldots c_r\}$, to a yet-to-arrive instance, $x \in \mathbb{R}^d$, were d is dimensionality of data and r is the total number of classes. The ensemble objective is simply defined as

$$y^* = argmax_{y \in Y} P(y|x, E) \tag{3.4}$$

When working with ensembles, the classification of an instance is often the weighted vote of all models in the ensemble, so the posterior probability would usually be defined as:

$$P(y|x, E) = \sum_{i=1}^{n} w_i P\left(y\middle|x, \lambda^i\right) \tag{3.5}$$

where w_i is the weight of the i^{th} model in the ensemble. However, when relying on clusters as some ensemble models, there is no true class information available – only group (cluster) identifiers, g. To incorporate the clusters into the ensemble model, the posterior probability is estimated by integrating the class mappings together for each cluster such that the weighted ensemble posterior probability is better defined by:

$$P(y|x, E) = \begin{matrix} \sum_{i=1}^{a} w_i P\left(y\middle|x, \lambda^i\right) + \\ \sum_{j=a+1}^{n} \sum_{k=1}^{r} w_j P\left(y\middle|g_k^j\right) P\left(g_k^j\middle|x, \lambda^j\right) \end{matrix} \tag{3.6}$$

The difficulty with calculating this ensemble posterior probability is $P\left(g_k^j\middle|x, \lambda^j\right)$ must be estimated and the weights cannot be determined through common performance metrics on the most recent batch of data since often the data are unlabeled. To overcome these problems the authors use a graph, $G = (V, E)$, in which the vertices, V, represent the cluster center of each model (classifier and cluster models alike), and the edges, E,

represent the similarity between the vertexes. The graph is used to propagate labels (and therefore estimate cluster posteriors), and define ensemble weights using the edges to develop a similarity metric.

When the incoming batch of data is received, the algorithm treats it as if the batch is unlabeled and creates v groups $\{g_1^{n+1}, \ldots, g_v^{n+1}\}$. If the batch of data is labeled, the class labels are assigned to the recently constructed v groups and a classifier, λ^{n+1} is constructed. The graph is then updated, adding the v new groups as vertexes, and removing old vertexes $\{g_1^1, \ldots, g_v^1\}$. The class label of each unlabeled group in the model is estimated using label propagation from the labeled groups to the unlabeled groups and the weights are determined using the following similarity metric:

$$w_i = \frac{1}{Z} * \frac{1}{\|\lambda^i - \lambda^n\|^2} \tag{3.7}$$

where $Z = \sum_{i=1}^{n} \frac{1}{\|\lambda^i - \lambda^n\|^2}$ serves as a regularizing term.

The ensemble can then be constructed and the weighted average of all the models is used to classify the incoming data.

3.6 Arbitrary Sub-Population Tracker Algorithm (APT)

Krempl proposes APT, the Arbitrary Sub-Population Tracker algorithm [65], which is the only algorithm we have discovered that attempts to address the same extreme verification latency issues as our COMPOSE Framework. Before discussing the mechanics of APT, let us outline the assumptions the APT model makes about the environment, where $P(X)$ represents the feature distribution, $P(Z)$ represents the

component prior distribution (i.e. mixing proportions), and $P(Y)$ represents the distribution of class labels:

a) the underlying population of the feature space consists of several sub-populations that evolve differently over time;
b) the data generated from this feature space can be represented with a mixture model of several components that drift over time;
c) each sub-population of the feature space must be represented by labeled data at initialization, where a sub-population is defined as a mode in the class conditional distribution $P(Y|X)$ (i.e., a bimodal class distribution would consist of two separate subpopulations to be tracked within a single class);
d) furthermore every instance must be labeled at initialization;
e) the drift must be gradual and "systematic", meaning it can be represented as a piecewise linear function;
f) the drift only affects the conditional feature distributions $P(X|Z)$;
g) so the conditional posterior distributions, $P(Y|Z)$, remains fixed (i.e. a component's class label cannot change);
h) and the prior distribution of components, $P(Z)$, is static (or changes very gradually if model is relaxed as discussed below);
i) the posterior distribution is independent of the (latent) component membership, $P(Y|Z) = P(Y|Z, X)$
j) covariance of each component remains constant

Since the author does not assume the conditional feature distributions of the components, $P(X|Z)$, to be Gaussian or any other parametric distribution, he uses a kernel estimator, a

non-parametric approach, to represent density distributions. A kernel estimator uses M samples, $X = \{x_1, \ldots, x_M\}$, to model the density distribution, $\hat{f}(x)$, underlying a sample, x. The standard kernel estimator is given in Equation 3.8 and works with several different kernel functions, $K_X(x - x_m)$ (e.g. radial basis, polynomial, Gaussian, etc.). Krempl presents his paper using the common choice of the Gaussian kernel; however, any kernel function will work. The generic D-dimensional Gaussian Kernel is given in Equation 3.9.

$$\hat{f}(x) = \frac{1}{M}\sum_{m=1}^{M} K_X(x - x_m) \tag{3.8}$$

$$K_X(x - x_m) = (2\pi)^{-\frac{D}{2}}|\Sigma^{-1}|^{\frac{1}{2}} exp\left\{-\frac{1}{2}(x - x_m)^T\Sigma^{-1}(x - x_m)\right\} \tag{3.9}$$

where Σ, is the covariance or generally referred to as the bandwidth of a kernel function. Krempl takes this standard kernel estimator and makes some minor changes to better fit the APT model to the nonstationary learning environment. Equation 3.10 shows a the adjusted kernel estimator function accounting for different time steps and a modified Gaussian kernel, $g_m(x, t)$, is presented in equation 3.11.

$$\hat{f}(x, t) = \frac{1}{M}\sum_{m=1}^{M} g_m(x, t) \tag{3.10}$$

$$g_m(x, t) = (2\pi)^{-\frac{D}{2}}\left|\Sigma_{z_m}^{-1}\right|^{\frac{1}{2}} exp\left\{-\frac{1}{2}d_m^T\Sigma_{z_m}^{-1}d_m\right\} \tag{3.11}$$

Where Σ_{z_m} allows there to be a different bandwidth matrix (covariance) for each component z, and $d_m = x - \tilde{x}_m(t)$ is the difference between position x where the density is being evaluated and the estimated position $\tilde{x}_m$ of the m^{th} component and time t. The estimated position is calculated in Equation 3.12 as

$$\tilde{x}_m(t) = x_m + (t - t_m) * \mu_{z_m}^{\Delta} \tag{3.12}$$

where $\mu_{z_m}^{\Delta}$ defines the component movement vector of the m^{th} component center. At initialization the initial cluster position is indicated by $\mu_{z_m}^{0}$.

The mechanics of APT are simple: incoming data are classified through a two-step procedure: i) use of expectation maximization to determine the optimal one-to-one assignment between the most recent batch of unlabeled data and the previous batch, now considered drift-adjusted labeled data; then ii) update the classifier to reflect the population parameters of newly received data and the drift parameters relating the previous time step to the current one.

Following assumption h, stating $P(Z)$ remains static, we are faced with a problem of creating a one-to-one mapping of an instance in time step t to an instance in time step $t+1$. When given a set of M known examples (exemplars), and a set of N new observations at positions $X = \{x_1, x_2, \dots, x_N\}$ and at times $T = \{t_1, t_2, \dots, t_N\}$, this problem corresponds to the following likelihood maximization problem

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

where $\Theta = \{\mu_1^0, \dots, \mu_K^0, \mu_1^\Delta, \dots, \mu_K^\Delta\}$ and z_{nm} is the observation-exemplar correspondence:

$$z_{nm} = \begin{cases} 1 \; if \; observation \; n \; corresponds \; to \; exemplar \; m \\ 0 \; otherwise \end{cases}$$

The bandwidth matrices Σ_{z_m} used are determined at initialization and assumed to remain constant.

To solve this likelihood maximization problem Krempl turns to a very standard approach of expectation maximization [21] which is formulated as:

$$max \quad l(\Theta; X, T) = \sum_{n=1}^{N} \sum_{m=1}^{M} z_{nm}\left(-2 d_{nm} \Sigma_{z_m} d_{nm}\right)$$

subject to

$$\sum_{n=1}^{N} z_{nm} = 1 \quad \forall m \quad \in 1,2,\dots,M$$

$$\sum_{m=1}^{M} z_{nm} = 1 \quad \forall n \quad \in 1,2,\dots,N$$

$$M = N$$

$$z_{nm} \in \{0,1\}$$

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. Krempl suggests a bootstrap method that can make the one-to-one assignments more robust, but at additional computational cost.

Chapter 4

The COMPOSE Framework

This chapter introduces COMPOSE – a framework utilizing semi-supervised learning to track data in nonstationary environments experiencing verification latency. The term framework is used since COMPOSE accomplishes its objectives using a combination of two modular components: any semi-supervised learning algorithm; and a class boundary estimator paired with its compaction technique. The chapter presents the algorithm's evolution through each revision accompanied by pseudocode, and detailed descriptions of each stage of the algorithm – constructing class boundaries, compacting these boundaries, and extracting relevant samples.

4.1 Fundamental Premise of the COMPOSE Framework

COMPOSE is intended for learning from gradually drifting distributions generated by nonstationary environments that produce streaming data with no labels. Gradual drift is often considered more challenging to detect than abrupt change, as the data distribution $p^t(\boldsymbol{x})$ at time t and $p^{t+1}(\boldsymbol{x})$ at time $t+1$ may have significant overlap, which makes distinguishing (detecting change between) the two difficult. COMPOSE turns this difficulty into an opportunity and takes advantage of the overlapping nature of incrementally changing distributions at consecutive time steps. The entire COMPOSE process is presented in a block diagram with accompanying illustrations in *Figure 4.1*.

At $t = 0$, COMPOSE is provided with (possibly very few) labeled data, depicted by opposing classes of (red) squares and (blue) circles (*Figure 4.1a*), and relatively abundant unlabeled data, represented by (black) diamonds (*Figure 4.1b*). At all other

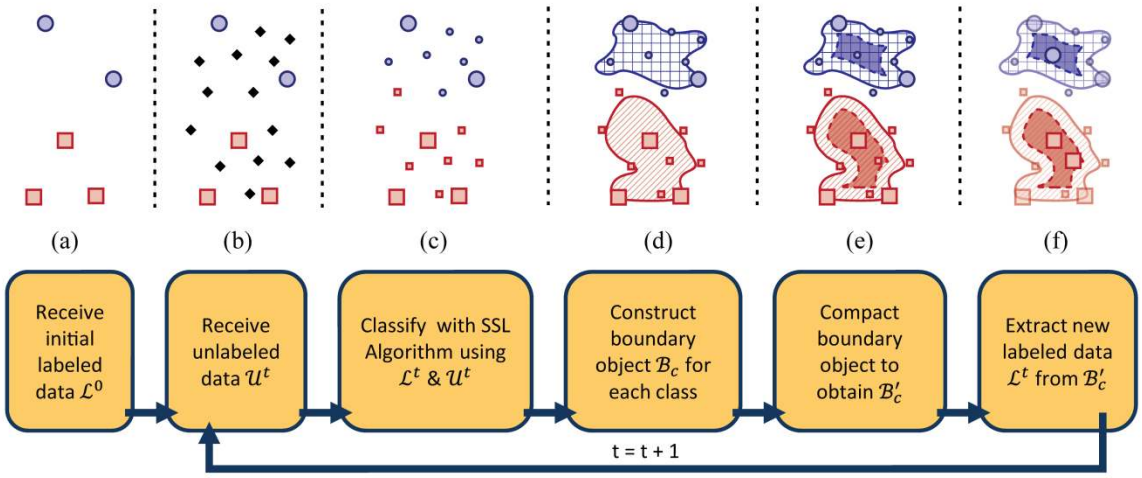

Figure 4.1. Graphical representation of COMPOSE stages

time steps $t > 0$, COMPOSE receives only unlabeled data. A semi-supervised learning algorithm is trained with the labeled and unlabeled data, to label the currently unlabeled instances, as indicated with change of color and shape in *Figure 4.1c*. COMPOSE creates a boundary object from the current data, defining a tight envelope representing the distribution of each class. Class boundaries are represented by solid outlines, enveloping shaded regions in *Figure 4.1d*. The boundary object of each class is compacted (i.e., shrunk) by a specified percentage, the compaction percentage, to determine the core support region of each distribution as shown by the darker shaded region with dashed outline in *Figure 4.1e*. Instances drawn from the core support region of the current distribution $p^t(\boldsymbol{x})$, shown as non-faded instances of *Figure 4.1f*, are the most likely candidates to represent data drawn from the next distribution $p^{t+1}(\boldsymbol{x})$ that may have experienced translational, rotational, or volumetric (i.e. expansion/contraction) drift. The final step of one iteration of COMPOSE extracts (now labeled) instances from the core support region(s) to be used as labeled data in the near future – these instances are referred to as the core supports of that class (*Figure 4.1f*). It is possible to have multiple

core support regions for any class. When new unlabeled data are received, they are combined with the core supports to retrain a semi-supervised learning algorithm to adapt to the drifting (nonstationary) environment, as COMPOSE iteratively updates itself. The progression of a single class distribution over a series of time steps is illustrated in *Figure 4.2*, experiencing translational (*Figure 4.2a*), rotational (*Figure 4.2b*), and volumetric (*Figure 4.2c*) drift. In each case, the core support region from the previous time step (boundaries indicated with dashed lines) indicate an area from which relevant instances can be extracted to label the next time step, t. It is important to emphasize that – unlike other semi-supervised learning algorithms used in nonstationary settings – all future labeled data are "earned" (generated) by COMPOSE through core support extraction, and not paid for, purchased or requested from the user.

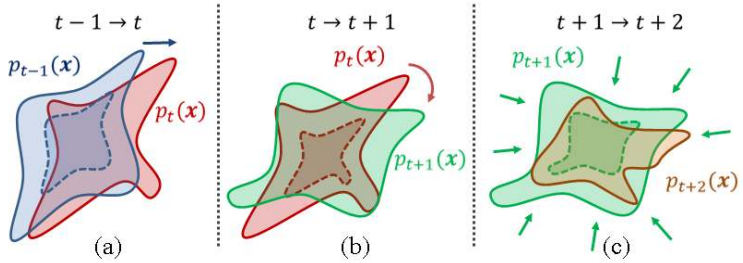

Figure 4.2. How COMPOSE accounts for various drift types Examples of (a) translational, (b) rotational, and (c) volumetric drift showing the core support region of previous time step provides an optimal area to draw instances from to train current data.

4.2 Evolution of the COMPOSE Framework

COMPOSE's fundamental principles, presented in the previous section, have remained consistent through its several minor revisions and one major revision presented

in *Table 4.1*. Each of the minor revisions has increased the accuracy of the framework while decreasing the computational complexity and the major revision involved a name change to include more class boundary estimation techniques.

Table 4.1.

Evolution of COMPOSE framework

Version 1.0	Any SSL algorithm, α-shape class boundary estimation, skeleton method compaction (limited to two dimensions)
Version 1.1	Any SSL algorithm, α-shape class boundary estimation, "FFT erosion" compaction of class boundary to relax
Version 1.2	Any SSL algorithm, α-shape class boundary estimation, layer lookup table compaction of α-shape
Version 2.0	Any SSL algorithm, any class boundary estimation, compaction matched to boundary estimation technique (framework renamed)

At conception, and throughout Version 1.x, COMPOSE stood for **COM**pacted **PO**lytope **S**ample **E**xtraction. The terms "sample extraction" and "compacted" are easily diagramed in *Figure 4.1* of the previous section; however, the term "polytope" is not adequately discussed. Quite simply, a polytope is a multi-dimensional geometric shape with flat sides (e.g., a polygon is a two dimensional polytope). This term is often used when discussing α-shapes, the class boundary estimation method used in Version 1.x. α-shapes are explained in detail in *Section 4.4*, and the progression of compaction methods for Version 1.x are presented in *Section 4.5*.

Version 2.x of the framework has changed the name to **COMP**acted **O**bject **S**ample **E**xtraction (but retaining the same acronym) to encompass alternative methods for generating class boundaries. Experiments have been conducted to prove alternative

methods exist to generate compactable class boundaries, but they are outside the scope of this thesis and are only described briefly as future work in *Chapter 6*.

4.3 Algorithm Description

Conventional semi-supervised algorithms, used in stationary environments, require sufficient amount of labeled as well as unlabeled data. In a nonstationary environment experiencing verification latency (as described in *Section 2.3*), not only are future labeled data rare or nonexistent, data also drift, preventing conventional semi-supervised algorithms from learning in such a setting. COMPOSE is designed to address this limitation by extracting relevant data, labeled by the semi-supervised learner in the current time step, to be combined with the next batch of unlabeled data. This important modification allows semi-supervised learning algorithms to be utilized in nonstationary environments.

The distribution $p^t(\boldsymbol{x})$ providing the unlabeled data at time t may have drifted from the distribution $p^{t-1}(\boldsymbol{x})$ at time $t-1$. Consistent with other nonstationary environment algorithms, we assume limited (gradual) drift, such that the extracted labeled data overlap the newly received unlabeled data. Therefore, the distribution $p^t(\boldsymbol{x})$ must overlap with the distribution $p^{t-1}(\boldsymbol{x})$. This minimum overlap requirement can be formally written as $\{\boldsymbol{x}: p^{t-1}(X = \boldsymbol{x}|Y) > 0 \cap p^t(X = \boldsymbol{x}|Y) > 0\} \neq \emptyset$. Of course, as the amount of overlap between distributions of subsequent time steps increase, the ability and performance of COMPOSE in tracking the nonstationary distribution is improved. The remainder of this section uses version 1.2 of the COMPOSE framework to explain in detail how COMPOSE i) creates α-shapes from the data; ii) compacts (shrinks) the α-

shapes to create core regions; and iii) extracts core supports from the compacted α-shapes to serve as labeled data for future time steps. The outline of the algorithm is listed in the pseudocode in *Figure 4.3*.

The algorithm has three inputs: i) **BaseClassifier**, which can be any semi-supervised learning algorithm, for classifying unlabeled data at each time step, t; ii) α, specifying the level of detail of the α-shape boundary object; and iii) CP, the compaction percentage. The algorithm is initialized at $t = 0$ with a set of labeled data, $\mathcal{L}^0 = \{\boldsymbol{x}_l^t \in X\}$, and corresponding labels, $\mathcal{Y}^0 = \{y_l^t \in Y = \{1, \dots, C\}\}$, $l = 1, \dots, M$ where M is the total number of labeled instances and C is the total number of classes (step 1 in *Figure 4.3*). At each subsequent time step t, new unlabeled data $\mathcal{U}^t = \{x_u^t \in X\}$ are received, $u = 1, \dots, N$ where N is the total number of unlabeled instances (step 2). Both labeled and unlabeled data are passed to **BaseClassifier** to generate a hypothesis $h^t: X \to Y$. A combined dataset $\mathfrak{D}^t$ is constructed by merging $\mathcal{L}^t$ and $\mathcal{U}^t$, where class labels for $\mathcal{U}^t$ are provided by h^t (step 3). With labels for all instances of $\mathfrak{D}^t$ now available, COMPOSE then extracts core supports for each class, selected from the core support region of the current distribution (steps 4 – 7). The underlying premise here is that the core support region of the data at the current time step – compared to any other time step – is most likely to have maximum overlap with the drifted distribution in the next time step, regardless of the nature of drift. Therefore, these core supports can be used to serve as labeled data for the next time step's SSL classifier. Specifically, the labeled dataset for the next time step ($\mathcal{L}^{t+1}, \mathcal{Y}^{t+1}$) is first initialized as an empty set (step 4). For each class, $c = 1, \dots, C$ identified by h^t; an α-shape class boundary object $\mathcal{B}_c$ is constructed using the method described in *Section 4.4* (denoted as function $f(\blacksquare)$ in step 5). The class boundary

object $\mathcal{B}_c$ is then compacted (i.e., shrunk) using the method described in *Section 4.5* to produce the core support region $\mathcal{B}_c'$ (denoted as function $g(\blacksquare)$ in step 6) such that desired core supports specified by compaction percentage CP are obtained. Then, all instances that reside in the compacted region $\mathcal{B}_c'$ are extracted as core supports and are retained to serve as labeled data for the next time step. Core supports obtained from each class are appended to finalize the labeled data ($\mathcal{L}^{t+1}, \mathcal{Y}^{t+1}$) in step 7.

Inputs: SSL algorithm – **BaseClassifier**; α-shape detail level – α; compaction percentage - CP

1. Receive labeled data
   $\mathcal{L}^0 = \{\boldsymbol{x}_l^t \in X\}, \mathcal{Y}^0 = \{y_l^t \in Y = \{1, \dots, C\}, l = 1, \dots, M\}$

Do for $t = 0,1, \dots$

2. Receive unlabeled data, $\mathcal{U}^t = \{x_u^t \in X, u = 1, \dots, N\}$
3. Call **BaseClassifer** with $\mathcal{L}^t$, $\mathcal{Y}^t$, and $\mathcal{U}^t$
   Obtain $h^t: X \to Y$,
   Let $\mathfrak{D}^t = \{(\boldsymbol{x}_l^t, y_l^t): x \in \mathcal{L}^t \forall l\} \cup \{(\boldsymbol{x}_u^t, h_u^t): x \in \mathcal{U}^t \forall u\}$
4. Set $\mathcal{L}^{t+1} = \emptyset$, $\mathcal{Y}^{t+1} = \emptyset$

Do for each class $c = 1, \dots, C$

5. Construct α-shape boundary, $\mathcal{B}_c = f(\alpha, \mathfrak{D}_c^t)$

Do Until number of core supports $\|CS_c\| = CP * \|\mathfrak{D}_c^t\|$

6. Compact α-shape boundary, $\mathcal{B}_c^{'} = g(\mathcal{B}_c)$

End

7. Extract core supports, $CS_c = \{x: x \in \mathcal{B}_c^{'}\} \cup \mathfrak{D}_c^t$, and add to labeled data for next time step
   $\mathcal{L}^{t+1} = \mathcal{L}^{t+1} \cup CS_c$
   $\mathcal{Y}^{t+1} = \mathcal{Y}^{t+1} \cup \{y_u: u \in [|CS_c|], y = c\}$

End

End

Figure 4.3. COMPOSE pseudocode

4.4 α-Shape Construction

In this section, we present the terminology used when discussing α-shapes and their construction, explore how α-shapes are affected by changing the α parameter, and explain how to construct an α-shape from data.

4.4.1 Terminology.

We first introduce the basic terminology used within the context of constructing α shapes. A d–simplex, or simply a simplex throughout this thesis, is the convex hull of $d+1$ vertices, connected via edges, where d is the dimensionality of the data. Examples of low dimensionality simplexes are provided in *Figure 4.4*: a 2-simplex is a triangle defined by three vertices; and a 3-simplex is a tetrahedron defined by four vertices. Each $d-$ simplex is constructed from multiple $(d-1)$-simplexes, called faces (e.g., each face of a triangle is a line; each face of a tetrahedron is a triangle). The circumsphere of a simplex is the hyper-sphere uniquely defined by the vertices of a simplex (e.g., a circle is defined by the three vertices of the triangle it circumscribes; a sphere is defined by the four vertices of the tetrahedron it circumscribes).

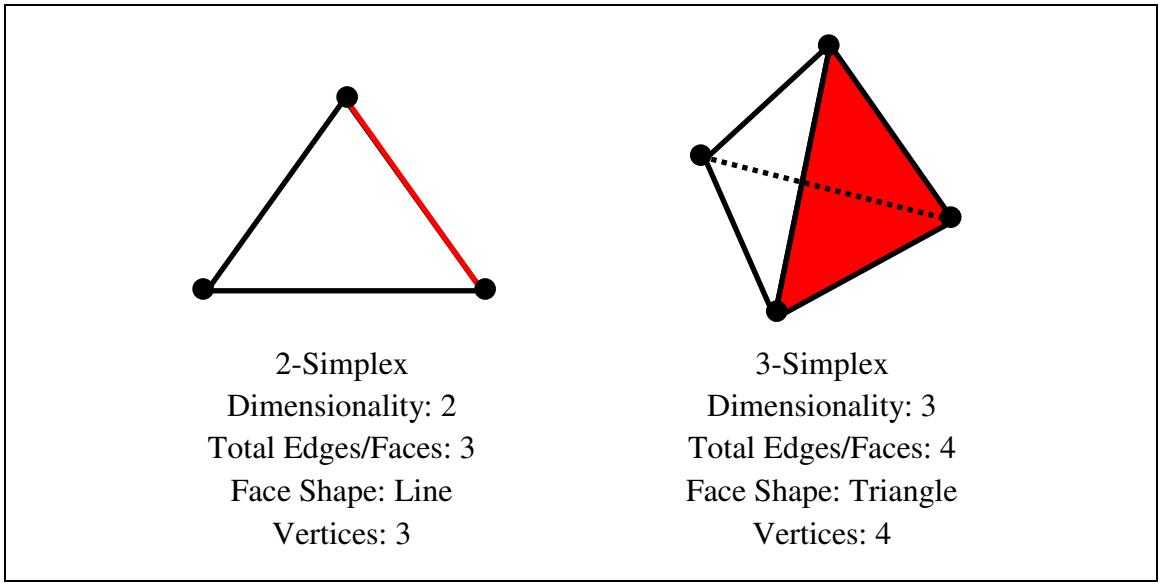

Figure 4.4. Examples of simplexes
A d-simplex resides in dimensionality, d, has d+1 vertices and d+1 faces. A single face of each simplex is show in red.

4.4.2 Effect of α parameter on α-shape. An α-shape is a set of connected faces creating a hull that describes a finite set of points at a specified level of detail, defined by the free parameter $\alpha > 0$. For a sufficiently large α, the resultant α-shape is the convex hull of the points. As α decreases, the α-shape may become concave, form holes, or include completely disconnected regions. These three aspects of α-shapes make them attractive for machine learning as they can properly represent voids and nested classes that many algorithms utilizing convex hulls or other simpler methods (such as calculating the centroid of a distribution) cannot. *Figure 4.5* demonstrates how α changes the representation of a data set in an α-shape. *Figure 4.5a* shows a large α resulting in the convex hull of the (blue) diamonds including a large region void of data, as well as an

opposing class of (red) circles. As α decreases in *Figure 4.5b-d*, the true feature space from which the set of diamonds was sampled becomes more apparent – the letter P. However, if α is chosen too small, as in *Figure 4.5e*, the α-shape becomes a group of disconnected regions, which is undesirable. The α parameter can be chosen heuristically, based on prior knowledge or experience, or based on sample density as proposed by Teichmann and Capps in [66].

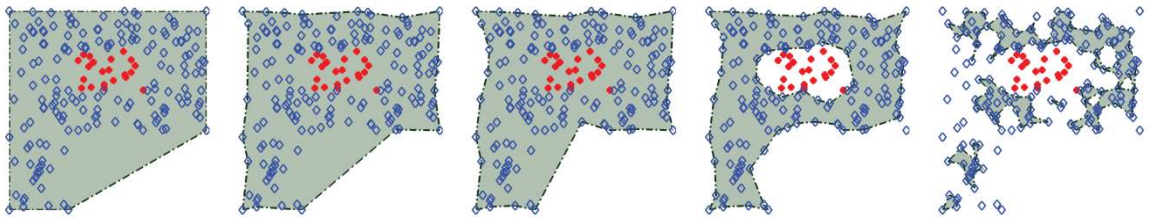

Figure 4.5. Effects of varying α parameter
The shaded region demonstrates an α-shape constructed on the set of blue diamonds at different levels of detail specified by α, decreasing from (a) to (e).

4.4.3 α-Shape construction. The pseudocode of the α-shape construction function is given in *Figure 4.6*, whose inputs are i) the α parameter specifying the desired level of detail, and ii) single-class data $\mathfrak{D}$ (as labeled by the semi-supervised learner in the previous step of the algorithm). α-shape construction begins with a Delaunay tessellation of $\mathfrak{D}$ (step 1 in *Figure 4.6*). Delaunay tessellations are an extension of Delaunay triangulations into higher dimensions. Delaunay tessellations nest simplexes such that no point in the set may lie inside the circumsphere of any simplex in the tessellation. The union of all the simplexes in the tessellation produces the convex hull of

Input: α-shape probing radius – α; Data features – $\mathfrak{D}$
1. Construct Delaunay tessellation of data, $T = Q(\mathfrak{D})$
2. Initialize α-shape as Delaunay tessellation $\mathcal{B} = T$
Do for each face, $\mathcal{F} \in T$
3. Find simplexes, s_1 and $s_2 \in T$, that share $\mathcal{F}$
4. Find radii of circumspheres, $\mu = r(\blacksquare)$
If $\mathcal{F}$ is an edge of T
Radius of simplex, $\mu_1 = r(s_1)$
Denote as boundary, $\mu_2 = Inf$
Else
$\mu_1 = \min[r(s_1), r(s_2)]$
$\mu_2 = \max[r(s_1), r(s_2)]$
End If
5. Categorize $\mathcal{F}$ and update $\mathcal{B}$ accordingly
Case 1: $\alpha > \mu_2$ $\mathcal{F}$ is interior
Case 2: $\mu_1 < \alpha < \mu_2$ $\mathcal{F}$ is regular, $\mathcal{B} = \mathcal{B} \backslash \{s_2\}$
Case 3: $\alpha < \mu_1$ $\mathcal{F}$ is singular, $\mathcal{B} = \mathcal{B} \backslash \{s_1, s_2\}$
End

Figure 4.6. α-Shape construction psuedocode

the set. To demonstrate this process pictorially, an example of a two-dimensional Delaunay triangulation is provided in *Figure 4.7*. The data provided, $\mathfrak{D}$, is shown in *Figure 4.7a*. *Figure 4.7b* demonstrates a possible simplex (triangle) constructed from the data; however, this is a non-Delaunay simplex since there are two data points (that are not vertices) residing inside the circle circumscribing the possible simplex. *Figure 4.7c* demonstrates another possible simplex on the same data; this time the selection is a Delaunay simplex because the circle circumscribing the simplex contains no additional data points. Continuing to select simplexes in this fashion results in the Delaunay triangulation shown in *Figure 4.7c*, note there are no data points inside any of the circumscribing circles.

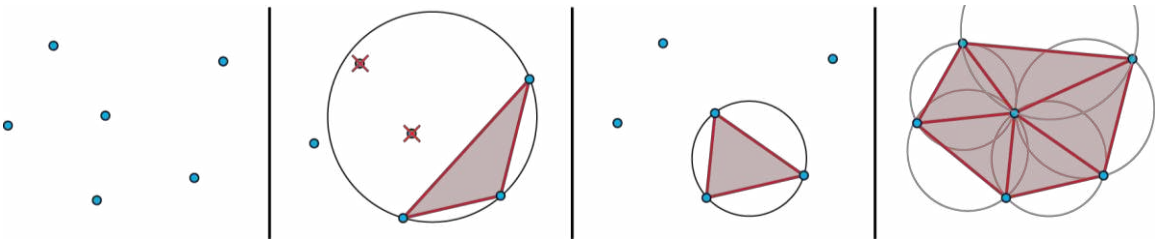

Figure 4.7. Delaunay triangulation
Delaunay triangulation requires a dataset (a) and then constructs triangles from the data such that no other data point resides in the circumsphere of any triangle. An improper triangle selection is shown in (b) since the two data points with the red X reside inside the black circle circumscribing the proposed triangle. A proper Delaunay triangle is show in (c) since no data resides in the circumcircle. The complete Delaunay triangulation is shown in (d).

There are several algorithms that accomplish Delaunay tessellations; we have used the Quickhull algorithm [67], denoted as $Q(\blacksquare)$ in step 1, for its speed and relative lower complexity whose upper bound is $\mathcal{O}\left(n^{\lfloor(d+1)/2\rfloor}\right)$, where n is the number of points in the set and d is dimensionality, and $\lfloor\blacksquare\rfloor$ is the floor function. It is important to note any Delaunay tessellation algorithm will work within the COMPOSE algorithm.

Once the convex hull of the data has been defined by the Delaunay tessellation, we initialize the α-shape, $\mathcal{B}$, to be the convex hull of the data (step 2). Each face, $\mathcal{F}$, is subsequently analyzed, categorized and, if necessary, certain simplexes containing that face are removed to produce the final α-shape (steps 3-5). To do so, we first iterate through every face, and identify the two simplexes, s_1 and s_2, that share $\mathcal{F}$ (step 3, and *Figure 4.8*). The radii of the circumspheres of each simplex are then calculated by passing the simplex's vertices to the circumsphere radius function (denoted $r(\blacksquare)$ in step 4, and described below) - the smaller radius is labeled μ_1 and the larger as μ_2 (*Figure*

4.8). If $\mathcal{F}$ is located at the edge of the tessellation (i.e., it is not shared by a second simplex), the radius of the (non-existent) second simplex is set to infinity, $\mu_2 = \infty$.

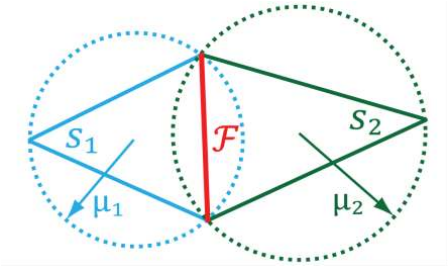

Figure 4.8. α-Shape construction simplex comparison
Face $\mathcal{F}$ (centered in red) to be classified is shared by simplex with smaller radius on left (blue) and simplex with larger radius on right (green)

The simplex passed to the circumsphere radius function is defined by its $d + 1$ non-coplanar vertices (instances) $\boldsymbol{x}_p$, $p = 1, \dots, d + 1$, each vertex defined by d coordinates (features):

$$\boldsymbol{x}_p = \left\{x_{p_1}, x_{p_2}, \dots, x_{p_d}\right\} \tag{4.1}$$

From the equation for circumcircle of a triangle [68], extended to higher dimensions, the equation of the circumsphere is:

$$\begin{vmatrix} \sum_d x_{\blacksquare_d}^2 & x_{\blacksquare_1} & x_{\blacksquare_2} & \cdots & x_{\blacksquare_d} & 1 \\ \sum_d x_{1_d}^2 & x_{1_1} & x_{1_2} & \cdots & x_{1_d} & 1 \\ \sum_d x_{2_d}^2 & x_{2_1} & x_{2_2} & \cdots & x_{2_d} & 1 \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ \sum_d x_{(d+1)_d}^2 & x_{(d+1)_1} & x_{(d+1)_2} & \cdots & x_{(d+1)_d} & 1 \end{vmatrix} = 0, \tag{4.2}$$

where $\boldsymbol{x}_\blacksquare$ is used to represent any point (instance) on the hypersphere, and $x_{\blacksquare_d}$ is its d^{th} feature. Cofactor expansion of the first row, valid for any point residing on the hypersphere, produces the equation of a hypersphere in general form:

$$\sum_d x_{\blacksquare_d}^2 \boldsymbol{M}_{11} + \left[\sum_d (-1)^d (x_{\blacksquare_d}) \boldsymbol{M}_{1(d+1)}\right] + \boldsymbol{M}_{1(d+2)} = 0 \qquad (4.3)$$

where $\boldsymbol{M}_{ij}$ represents a matrix minor – the determinant of the matrix after removing row i and column j. The result after completing the square and rearranging the terms is the standard form of a hypersphere:

$$\sum_d \left(x_{\blacksquare_d} - x_{0_d}\right)^2 = r^2 \qquad (4.4)$$

where

$$x_{0_q} = (-1)^{q+1} 0.5 \frac{M_{1(q+1)}}{M_{11}}, q = 1, \dots, d \qquad (4.5)$$

$$r^2 = \sum_d x_{0_d} - \frac{M_{1(d+2)}}{M_{11}} \qquad (4.6)$$

with $\boldsymbol{x}_0$ and r being the center and radius of the hypersphere, respectively.

Once computed, radii of the simplexes are compared to α to determine if the face is interior, regular, or singular (step 5 in *Figure 4.6*). An interior face, where $\alpha > \mu_2$, is completely encapsulated by the final α-shape resulting in both simplexes that share this face to remain within the α-shape. A regular face, where $\mu_1 < \alpha < \mu_2$, defines the boundary of the α-shape, these faces are shown as dark black faces in *Figure 4.9*. When analyzing a regular face, the simplex with the larger radius circumsphere, shown as red simplexes to the outside of dark black faces in *Figure 4.9*, is removed from the α-shape. The simplex with the smaller radius circumsphere remains, and are shown as green simplexes in *Figure 4.9*. A singular face, where $\alpha < \mu_1$, as described by Edelsbrunner [69], traditionally has two sub-categories: attached and unattached. In either case both simplexes are removed, however the shared edge remains protruding from the α-shape as a "spoke" in the attached subcategory. The use of α-shapes in COMPOSE does not require differentiation between these two subcategories, as the singular-attached case

always disappears during the α-shape compaction function described in *Section 4.5*. Hence, all singular faces and both simplexes that share the singular face are removed from the final α-shape. Examples of each type of edge and the resultant α-shape after simplexes have been removed are shown in *Figure 4.9*. While an α-shape is traditionally defined as the union of all regular and singular faces, it suffices for COMPOSE to define an α-shape to be the union of all simplexes not removed from the Delaunay tessellation.

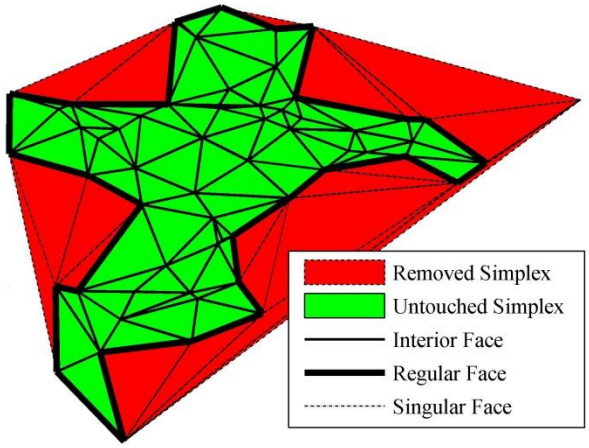

Figure 4.9. Sample α-shape classifications
Sample α-shape showing simplexes in Delaunay tessellation and how faces are classified in relation to placement in an α-shape.

The construction of the α-shape is the most expensive module of the COMPOSE algorithm, especially with high dimensional data, with the Delaunay tessellation running in $\mathcal{O}\left(n^{\lfloor (d+1)/2 \rfloor}\right)$ and producing $\mathcal{O}\left(n^{\lfloor d/2 \rfloor}\right)$ simplexes each containing $d+1$ faces that must be compared to α. We discuss methods to reduce complexity of this portion of the algorithm in *Chapter 6*.

4.5 α-Shape Compaction

This section highlights the changes in each version of COMPOSE explaining the reason for the changes and decrease in computational complexity achieved.

4.5.1 Version 1.0 – skeletal offsets. In version 1.0, as described in *Table 4.1*, the constructed α-shape was compacted using skeletal offsets – a method used extensively in image processing and computer aided drawing software to scale enclosed regions. Offsets are accomplished in two dimensions by translating the vertices of a polygon along its straight skeleton as described in [70]. The straight skeleton of a polygon is the combination of all arcs that bisect any two edges. An example of a shape and its straight skeleton are shown in *Figure 4.10*. This method of constructing a straight skeleton in two dimensions has a computational complexity of $\mathcal{O}(v^2 \log v)$, where v is the number of vertices. This method scales to three dimensions – albeit at great cost – and has problems when attempting to scale to higher dimensions. For these reasons a new method for compacting alpha-shapes was explored.

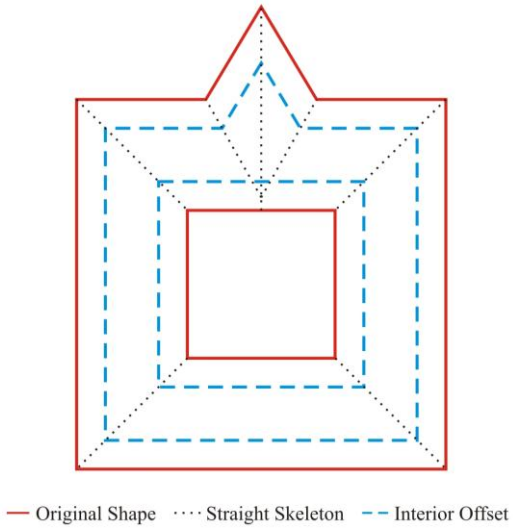

Figure 4.10. Skeletal offset
Original shape and its skeleton offset with sample of interior and exterior offsets. Note for a sufficiently large offsets, shape information may be lost (e.g., with a large enough offset the point at the top center of the shape may no longer be reflected as being part of the original shape).

4.5.2 Version 1.1 – fast Fourier transform based erosion. Version 1.0 of COMPOSE relied on computing the straight skeleton to compact the α-shape. Straight skeletons work well with two dimensional data; however, the straight skeleton approach does not easily scale to higher dimensions. Version 1.1 of the COMPOSE framework utilizes a Fast Fourier Transform (FFT) based image processing technique – erosion – to compact α-shapes in higher dimensions. As with all image processing, the object being analyzed must be represented discretely. In our case the continuous feature domain encapsulated by an α-shape must be discretized. This could be compared to a camera which captures its surroundings (continuous) and represents them by an image with discrete pixels. Forming a discrete representation of the α-shape constructed in

continuous space is accomplished using the α-shape discretizing function described in *Figure 4.11*.

Input: α-shape – $\mathcal{B}$; Resolution – $\mathcal{r}$
Do for each $d \in \mathcal{B}$
1. $\mathcal{V}_d = \min \mathcal{B}_d, \dots, \frac{k(\min \mathcal{B}_d + \max \mathcal{B}_d)}{\mathcal{r}+1}, \dots, \max \mathcal{B}_d$
$\forall k = 2, \dots, \mathcal{r} - 1$
End
2. Construct lattice, $\boldsymbol{L} \in \mathbb{R}^{d+1}$, from all permutations of points in $\mathcal{V}$. $\boldsymbol{L}_0$ is binary indicator initialized to $\mathbf{0}$.
Do for each simplex, $s \in \mathcal{B}$
  Do for each point, $P \in \boldsymbol{L}$
    3. Determine if P resides inside s using Barycentric coordinate function, $\lambda = b(s, P)$
    If $\boldsymbol{\lambda \geq 0}$
      $\boldsymbol{L}_{0,P} = 1$ and record simplex that contained it
    End
  End
End

Figure 4.11. α-Shape discretizing function pseudocode

The inputs to the discretizing function are the continuous valued α-shape $\mathcal{B}$ (specifically, the coordinates of simplex vertices); and the starting resolution, $\mathcal{r}$, dictating how many points are used in each dimension to represent the α-shape discretely. For each dimension of the α-shape (example shown in *Figure 4.12a*), a vector, $\mathcal{V}$, with $\mathcal{r}$ equally spaced points between the minimum and maximum coordinates is constructed (step 1 of *Figure 4.11* and depicted in *Figure 4.12b*). A lattice, denoted by tensor $\boldsymbol{L}$, is constructed in $\mathbb{R}^{d+1}$space using all permutations of coordinates in the aforementioned vectors, $\mathcal{V}_d$, and reserving $\boldsymbol{L}_0$ as a binary indicator representing whether the point specified by coordinates

$\boldsymbol{L}_{1,\ldots,d}$ resides inside or outside the α-shape. Initially the indicator value in $\boldsymbol{L}_0$ for each point is set equal to zero indicating that the point resides outside the α-shape. Transforming the lattice into an accurate description of the α-shape is accomplished by using Barycentric coordinates to determine if each lattice point resides inside any simplex in the α-shape. Data points that reside inside the simplex are represented by yellow dots in *Figure 4.12c* and the corresponding $\boldsymbol{L}$ tensor is shown in *Figure 4.12d* overlaying the simplex and grid.

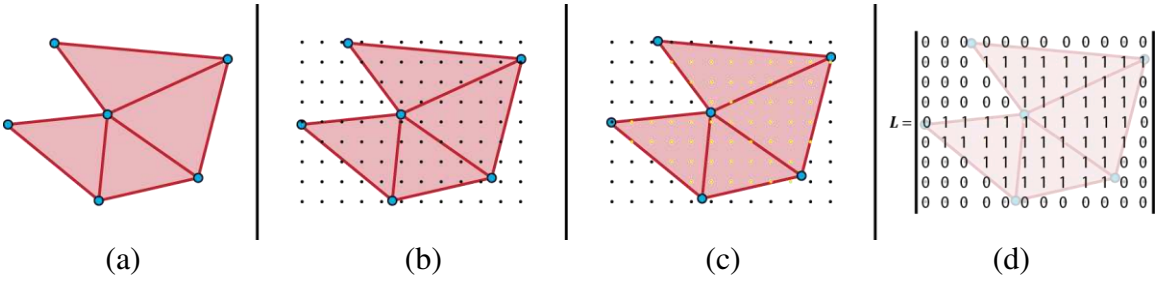

Figure 4.12. Discretizing an α-shape
The α-shape in (a) is discretized by overlaying a uniformly spaced grid as shown in (b) then determining the Barycentric coordinates of each point with regard to each simplex and marking any data point that resides within a simplex. These points are indicated by yellow dots in (c) and the corresponding $\boldsymbol{L}$ tensor is shown in (d) overlaying the α-shape and grid of discrete points.

Barycentric coordinates are often used to determine the center of mass of an object, but can also be used to determine if a point in the lattice resides in at least one simplex of the α-shape. Barycentric coordinates represent a point as the weighted sum of the vertices defining a simplex: if all weights are positive (or one weight is equal to zero) the point resides inside (or on) the simplex. The inside simplex test function using Barycentric coordinates (denoted as function b(■) in step 3 of *Figure 4.11*) requires the coordinates of i) the point being tested, $\boldsymbol{L}_{1,\ldots,d}$ and ii) the vertices of the simplex, s, being

evaluated. Using similar notation as Equation 4.1, let $\boldsymbol{P_0}$ be a column vector representing the test point and $\boldsymbol{P_1}$ through $\boldsymbol{P_N}$ be column vectors representing the vertices of the simplex. The test point can be described as weighted components of the vertices:

$$\begin{aligned} x_{0_1} &= \lambda_1 x_{1_1} + \lambda_2 x_{2_1} + \cdots + \lambda_N x_{N_1} \\ x_{0_2} &= \lambda_1 x_{1_2} + \lambda_2 x_{2_2} + \cdots + \lambda_N x_{N_2} \\ &\vdots \\ x_{0_d} &= \lambda_1 x_{1_d} + \lambda_2 x_{2_d} + \cdots + \lambda_N x_{N_d} \end{aligned} \tag{4.7}$$

where $\lambda_{1,\ldots,N}$ are the weights of each simplex vertex and $\sum \lambda = 1$. In order to solve this system of equations, we make the substitution $\lambda_N = 1 - \sum_d \lambda_d$ and place in matrix form, $\boldsymbol{T\lambda} = \boldsymbol{P_0} - \mathbf{P_N}$ where,

$$\mathbf{T} = \begin{bmatrix} \mathrm{x}_{1_1} - \mathrm{x}_{\mathrm{N}_1} & \mathrm{x}_{2_1} - \mathrm{x}_{\mathrm{N}_1} & \cdots & \mathrm{x}_{\mathrm{d}_1} - \mathrm{x}_{\mathrm{N}_1} \\ \mathrm{x}_{1_2} - \mathrm{x}_{\mathrm{N}_2} & \mathrm{x}_{2_2} - \mathrm{x}_{\mathrm{N}_2} & \cdots & \mathrm{x}_{\mathrm{d}_2} - \mathrm{x}_{\mathrm{N}_2} \\ \vdots & \vdots & \ddots & \vdots \\ \mathrm{x}_{1_\mathrm{d}} - \mathrm{x}_{\mathrm{N}_\mathrm{d}} & \mathrm{x}_{2_\mathrm{d}} - \mathrm{x}_{\mathrm{N}_\mathrm{d}} & \cdots & \mathrm{x}_{\mathrm{d}_\mathrm{d}} - \mathrm{x}_{\mathrm{N}_\mathrm{d}} \end{bmatrix} \tag{4.8}$$

Since the vertices define a simplex, the equations are linearly independent and $\mathbf{T}$ is invertible; therefore, the weights can be determined by $\boldsymbol{\lambda} = \mathbf{T}^{-1}(\mathbf{P_0} - \mathbf{P_N})$. Determining if the point resides inside the simplex requires a simple inequality test: if all weights are positive or any one is equal to zero, the point resides inside or on the simplex (λ_N must be included in the test and can be calculated using $\lambda_\mathrm{N} = 1 - \sum_\mathrm{d} \lambda_\mathrm{d}$). If indeed the point resides inside the simplex, the corresponding indicator value, $\mathbf{L}_{0,1,\ldots,\mathrm{d}}$, must be changed to a "1" and the simplex number that contained the points is recorded. All points of the lattice can be tested through one matrix multiplication if the definition of $\mathbf{P_0}$ is altered to be a matrix having a column for every point in the lattice, while the rows still represent each point's dimensional coordinates. If this method is utilized, a matrix the

same size as $\mathbf{P_0}$ is constructed by repeating vertex $\mathbf{P_N}$ to maintain correct matrix dimensionality for subtraction.

The complexity of the discretizing process is $\mathcal{O}(\mathrm{d}^2 r^{\mathrm{d}})$, where d is the dimensionality of the data and r is the resolution of the lattice. Timing tests varying the resolution and number of simplexes in different dimensional feature spaces showed that calculation time increases linearly with the number of simplexes, but exponentially with the dimensionality. Altering the resolution had a much greater impact in higher dimension, which is expected due to the r^{d} term.

After the alpha-shape has been discretized, the compaction process using FFT based erosion is conducted. The inputs to the discrete α-shape compaction function (pseudocode presented in *Figure 4.13*) are the discretized α-shape, which contains all coordinates and in/out indicators of the lattice constructed earlier; and the offset distance σ, which determines how far inward the α-shape is to be eroded/compacted. Erosion is completed by convolving the binary "image", constructed above, with a d-dimensional hypercubic binary structuring element $\boldsymbol{S}$(i.e., "filter"). The structuring element is constructed such that the length of one side of the hypercube is equal to the offset distance and the binary value of each "pixel" is one (step 1 of *Figure 4.13*).

Input: Discrete α-shape – $\boldsymbol{\mathcal{A}}'$; Offset distance – σ

1. Construct structuring object for erosion
$$\boldsymbol{\mathcal{S}} \in \mathbb{R}^d, card(\boldsymbol{\mathcal{S}}) = \sigma^d$$
2. Zero pad $\boldsymbol{\mathcal{L}}$ and $\boldsymbol{\mathcal{S}}$ to a hypercube with a side length
$$\sqrt[d]{card(\boldsymbol{\mathcal{S}})} + \sqrt[d]{card(\boldsymbol{\mathcal{A}}')} - 1$$
3. Convolve $\boldsymbol{\mathcal{L}}$ and $\boldsymbol{\mathcal{S}}$ in frequency domain
$$\boldsymbol{\mathcal{E}} = dFFT(\boldsymbol{\mathcal{A}}').* dFFT(\boldsymbol{\mathcal{S}}),$$
where $.*$ is point by point multiplication
4. Take inverse d-dimensional FFT
$$\boldsymbol{E} = IdFFT(\boldsymbol{\mathcal{E}})$$
5. Threshold $\boldsymbol{E}$ to convert to binary compacted α-shape
$$(\boldsymbol{E} = 0) = 0 \text{ and } (\boldsymbol{E} > 0) = 1$$
6. Extract centermost region of $\boldsymbol{E}$, having same cardinality and structure as $\boldsymbol{\mathcal{A}}'$
$$\boldsymbol{\mathcal{A}}'' = center(\boldsymbol{E}), \text{where } card(\boldsymbol{\mathcal{A}}'') = card(\boldsymbol{\mathcal{A}}')$$
7. Create an α-shape of core support region
$$\boldsymbol{\mathcal{A}}' = f(\alpha, \mathcal{L} \forall \mathcal{L}^0 = 0)$$
8. Determine which instances of $\mathfrak{D}$ are inside $\boldsymbol{\mathcal{A}}'$ using Barycentric coordinates

Figure 4.13. α-Shape compaction pseudocode

As with any filtering process, filter delay is inevitable. However, to negate the effect of the filter delay, which would be the equivalent of translating the α-shape in the feature space, both the "image" and the structuring element are zero padded such that the length of each side of the padded hypercube is $\sqrt[\mathrm{d}]{\mathrm{card}(\boldsymbol{\mathcal{S}})} + \sqrt[\mathrm{d}]{\mathrm{card}(\boldsymbol{\mathcal{A}}')} - 1$ (step 2 of *Figure 4.13*). This is equivalent to zero padding sequences of length N, M to length $\mathrm{N} + \mathrm{M} - 1$ to make linear and discrete convolution the same. Once both the "image" and structuring element are zero padded to the same size, convolution is efficiently conducted by taking the d-dimensional FFT of each and multiplying them point by point in the spatial frequency domain, $\boldsymbol{\mathcal{E}} = \mathrm{dFFT}(\boldsymbol{\mathcal{A}}').* \mathrm{dFFT}(\boldsymbol{\mathcal{S}})$, (step 3 of *Figure 4.13*). Converting the convolved image back to the spatial domain, where it must compared to a zero

threshold, is accomplished using the inverse d-dimensional FFT (step 4 of *Figure 4.13*). This process effectively analyzes the rate of change between each pixel and its neighboring pixels in every dimension. In regions completely outside or inside the α-shape, there is no change in neighboring pixels, they are all zeros or ones, respectively. However, for pixels on the boundary of the α-shape, multiplication of pixels will result in some values originally having a value one being changed to zero. The end result is the boundary moving inwards, towards the core supporting region of the α-shape. To convert the eroded image back to a binary representation, the image is compared to a zero threshold such that any pixel with a value greater than zero is set to one and zero values remain unchanged (step 5 of *Figure 4.13*). The eroded image is still larger in each dimension than the original input due to padding. To extract the true eroded α-shape and discount the pixels contributed by $\boldsymbol{S}$, the centermost pixels having the same cardinality as $\boldsymbol{\mathcal{A}'}$ are extracted (step 6 of *Figure 4.13*). The complexity of this portion of the algorithm is $\mathcal{O}\left(r^{2\mathrm{d}}\log\left(r^{\mathrm{d}}\right)\right)$.

Recall during the discretizing α-shape function that we constructed a lookup table indicating which discrete points reside in each simplex in the α-shape. The points in this table are passed through the compaction function, resulting in only the discrete compacted points still having a value of one. Conducting a reverse lookup in this table, allows us to determine which simplexes contain a discrete compact point. The vertices of these simplexes then constitute the core supports of the current distribution and are retained by COMPOSE as labeled data to be combined with the next batch of (possibly drifted) unlabeled data. Note that these points selected from the core support of the

current distribution are most likely instances to be in the region of support of the drifted distribution.

Figure 4.14 illustrates the aforementioned set of steps as an example. The figure contains an enlarged view of an α-shape for one class and depicts COMPOSE's process for selecting the core supports. Recall that the α-shapes are constructed for each class label in the data. The α-shape, shaded in light yellow in *Figure 4.14*, is constructed for the data (red) classified by the SSL algorithm (BaseClassifier) as belonging to some particular class. The discrete lattice, shown by black dots, spans the hyper cubic (rectangular region in this 2D figure) space containing the α-shape. Using Barycentric coordinates, discrete points that fall inside the α-shape are identified, which are indicated with blue stars. The binary representation of the discrete space (black points = 0, blue stars= 1) is compacted, where compacted points are shown as bold blue circles. Using the point-simplex look-up table, the vertices of each simplex containing a compacted point are highlighted with a black diamond and extracted as labeled data at the next time step.

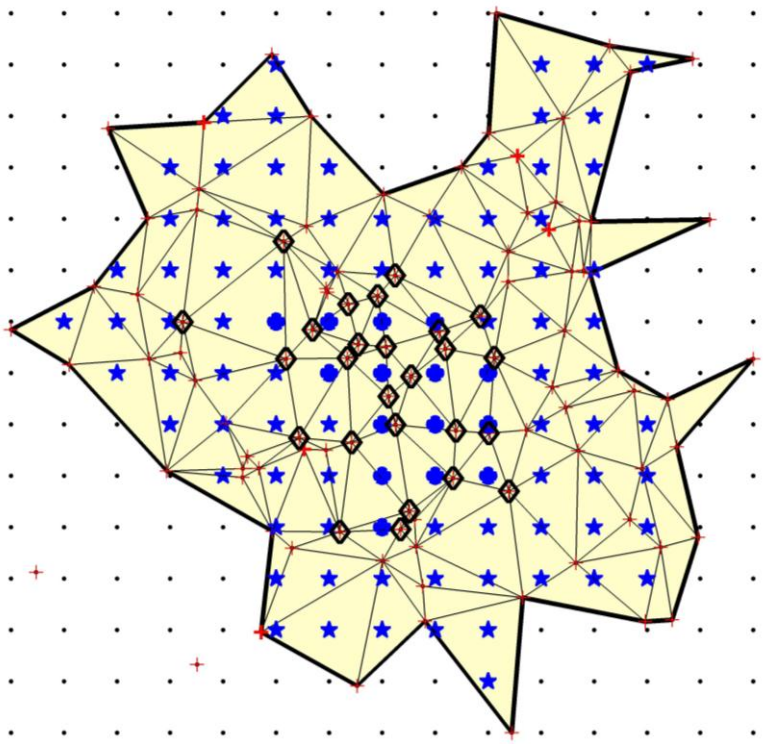

Figure 4.14. α-Shape compaction using FFT based erosion The process of extracting core supports (black diamonds) from an α-shape (shaded yellow region bounded by solid black line). The process includes constructing the discrete lattice (black points), identifying those (red plus) points that fall inside discretized α-shape (blue stars), compacting the inside points (blue circled stars), and identifying the vertices of simplexes that contain the compacted points to use as labeled data.

4.5.3 Version 1.2 – α -shape unwrapping. The great number of tunable parameters of Version 1.1 and computational resources required for high resolution "images" in high dimensions required further improvement. In Version 1.2, the most eloquent of the three versions, compaction is achieved by iteratively removing a layer of simplexes from the edges of the α-shape, as if unwrapping an onion, until the desired compaction percentage is achieved – percentage of compaction is the only parameter specified. The compaction threshold is found by multiplying the number of instances in

the initial α-shape by $(1 - \mathrm{CP})$, yielding the target number of instances to remove. Each time a layer of simplexes is peeled off, the number of instances in the compacted α-shape is reduced. Compaction is complete when the number of remaining core supports is less than or equal to the compaction threshold.

This method is illustrated in *Figure 4.15*, where each simplex removed numbered by the layer in which it is removed. The first (outermost) layer removed is indicated by "1" and shaded in red; the last layer is in light blue and contains "6". The data remaining after the compaction become the core supports, indicated by white stars clustered at the center of the α-shape.

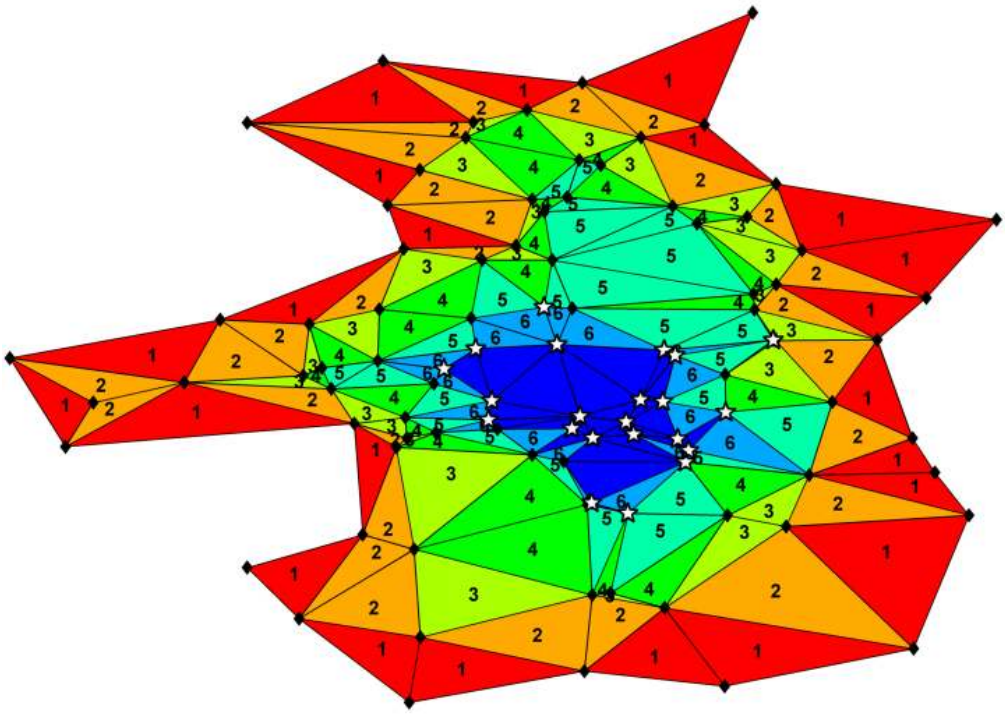

Figure 4.15. Graphical representation of unwrapped α-shape
Layers are removed in numerical order starting with (red) "1" and ending with (blue) "6" until core supports remain, represented by (white) stars. Compaction percentage used for this figure was 85%.

Identifying which simplexes reside at the edge of an α-shape is a simple task, as boundary simplexes have one or more faces that are not shared with another simplex. By creating a list of all faces and identifying to which simplex each belongs, a simple sort can identify unmatched faces. The simplex IDs associated with the unmatched faces are the simplexes located at the edge of the α-shape. The complexity of this method is $\mathcal{O}(s^2)$, where s is the total number of simplexes in the α-shape, which is linearly related to the total number of instances. This compaction function, unlike the original skeleton based compaction algorithm and the FFT based erosion is independent of dimensionality, and hence significantly reduces the complexity of the overall approach.

Chapter 5

Experiments and Discussions

5.1 Experimental Setup and Results on Synthetic Datasets

We have tested each version of COMPOSE on carefully designed synthetic data sets, using nonstationary Gaussian data, and demonstrated that later versions of the COMPOSE framework: 1) perform just as well, if not better, than earlier versions; 2) extend to higher dimensions than earlier versions; and 3) can adapt to the introduction of a new class.

COMPOSE version 1.0, which used skeletal offsets for object compaction, was limited to two-dimensional data, so only Experiments 1 and 2, presented in *Figure 5.1* and *Figure 5.3* respectively, were run. As COMPOSE progressed to version 1.2 (denoted as COMPOSE* in figures), Experiments 1 and 2 were rerun to demonstrate the later version performed just as well, if not better, than the earlier version. Two new synthetic Gaussian Experiments, Experiments 3 and 4 presented in *Figure 5.5* and *Figure 5.7* respectively were developed to test the ability of version 1.2 to process higher dimensional data and adapt to newly introduced classes in data. To better evaluate the capabilities of COMPOSE, each of the four experiments referenced above were repeated using the APT algorithm (presented in *Section 3.6*), the only other algorithm currently available for the extreme verification latency problem, and the optimal Bayes classifier, which provides an upper bound to performance. The Bayes classifier was trained in a fully supervised manner, having full access to correct labels for all instances at all time steps. This is a scenario that is deliberately designed to be unfair against COMPOSE and APT, as these algorithms maintained the initially labeled streaming environment

assumption where labels were provided only for a subset of the data, and only during the initial time step. All comparisons to Bayes classifier should be interpreted within this context.

In each of the four experiments listed above, we assumed Gaussian distributions starting at some initial state at an arbitrary time $t = 0$. COMPOSE was initialized using only 5% of randomly selected data labeled, though we ensured each class is represented by at least one labeled instance; ATP, however, requires a full set of labeled data at initialization. At each subsequent step t, the distributions drift according to the parametric equations shown in *Tables 5.1*, *5.2*, *5.3*, and *5.4*, and illustrated in *Figures 5.1*, *5.3*, *5.5*, and *5.7*, respectively for Experiments 1 - 4, with 100 new unlabeled instances presented per Gaussian mode. The experiments end after 100 steps, at some arbitrary time, $t = 1$. All experiments were repeated 50 times for COMPOSE and five times for ATP, providing the 95% confidence intervals indicated as the shaded regions around the performance curves. ATP was run only five times due to its significantly longer computation time as discussed in *Section 5.3* below.

COMPOSE's independence of SSL algorithm used as the BaseClassifier is demonstrated by Experiments 1 and 2 whose results are shown in *Figure 5.2* and *Figure 5.4*, respectively. Regardless of BaseClassifier selected, the performance closely follows the performance trend of Bayes rule. Our statement of independence does not claim that each classifier will perform equally well when paired with COMPOSE, it simply states that each classifier will follow a similar performance trend. It is important to note that each classifier has its strengths and weaknesses depending on the environment it is classifying. For example, of the three BaseClassifiers used with COMPOSE, label

spreading performed the poorest, which may be attributed to the placement of labeled instances. When labeled instances from a particular class span a larger area in feature space (albeit, possibly with less density), it is easier for that class to spread its label, since spreading can proceed in more directions and overtake a larger area of unlabeled instances faster. In a nonstationary environment that provides labeled instances at every time step directly from the underlying distribution, the labeled data are more likely to be scattered throughout the unlabeled data. Using COMPOSE, however, labeled data are located in a tighter cluster due to sampling from a compacted α-shape. This tight cluster of labeled data decreases the effectiveness of classification through label spreading. SSL algorithms that do not use label spreading, however, do not suffer from such a restriction.

After demonstrating classifier independence, the remaining experiments in this thesis are presented with cluster-and-label chosen as the semi-supervised algorithm. This algorithm was selected due to minimal free parameters it needs, and its ability to easily adapt to a multiclass problem – unlike, e.g., S3VM, which does not readily work in multiclass problems.

There are several variations of cluster-and-label; we used k-means to perform the clustering, and majority vote of labeled instances in the clusters for labeling the clusters. The algorithm begins with $k = 5$, the number of clusters to find, which iteratively reduces itself by one if it is unable to find a solution where every cluster contains at least one labeled point. COMPOSE free parameters (α and CP) were selected heuristically (shown within figures), were not optimized, and remained fixed throughout the experiments.

5.1.1 Unimodal and multimodal Gaussians. The two experiments in this section were featured in the initial publication of COMPOSE Version 1.0 [71], and serves as a benchmark for comparison of Version 1.2 [72]. The experiments are governed by parametric equations provided in *Tables 5.1*, *5.2*. As shown in *Figures 5.2* and *5.4*, version 1.2 of the COMPOSE framework (denoted by solid red line and marked COMPOSE*) performs better in both experiments when compared to its earlier counterpart (using cluster-and-label as the SSL). Performance of COMPOSE Version 1.0 with other SSL algorithms are also shown for comparison.

During periods of increased class overlap, time steps 60 – 70 in *Figure 5.2*, COMPOSE outperforms APT with statistical significance. During the remainder of the experiment both ILSE algorithms have similar performances, tracking Bayes classifier (black curve) extremely close.

The primary weakness of APT – the assumption that all subpopulations must be present at initialization – is most vividly seen in the second experiment that featured a scenario that split a unimodal distribution into a multimodal distribution, which have then merged to return to a unimodal distribution later. APT failed to track these diverging distributions, as illustrated in *Figure 5.4*, because the diverging distribution creates a new subpopulation that APT did not know at initialization. COMPOSE however, is able to track the distributions before the split, throughout the split, as well as after their merge. Furthermore, COMPOSE follows the performance of Bayes closely. This is a quite noteworthy accomplishment, considering the unfair circumstances under which COMPOSE operates against the Bayes classifier.

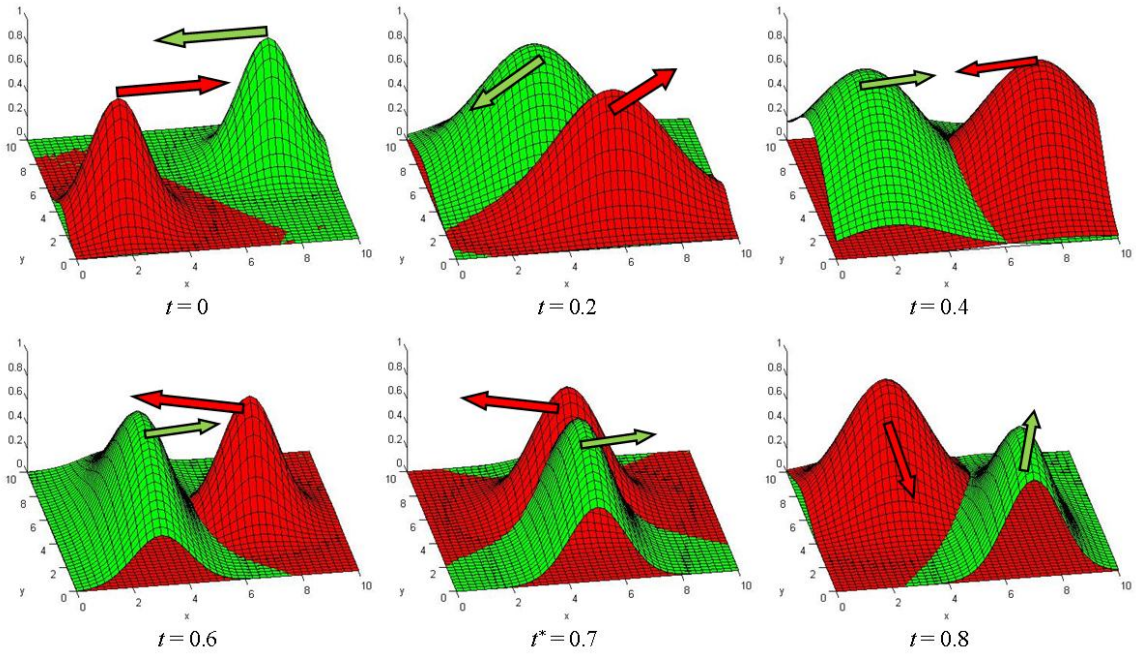

Figure 5.1. Experiment 1 – unimodal Gaussians

Table 5.1.

Parametric equations governing unimodal Gaussian experiment drift

Class	**0 ≤ t < 0.2**				**0.2 ≤ t < 0.4**				**0.4 ≤ t < 0.6**			
	μ_x	μ_y	σ_x	σ_y	μ_x	μ_y	σ_x	σ_y	μ_x	μ_y	σ_x	σ_y
C_1	2 +	3	1 + 5t	1	6 +	3 +	2	1 + 5t	8 - 5t	5	2 - 5t	2 - 5t
C_2	8 - 20t	7	1 + 5t	1 + 5t	4 - 10t	7 - 10t	2	2 + 5t	2 + 5t	5 - 5t	2 - 5t	3

Class	**0.6 ≤ t < 0.8**				**0.8 ≤ t ≤ 1**			
	μ_x	μ_y	σ_x	σ_y	μ_x	μ_y	σ_x	σ_y
C_1	7 - 20t	5 +	1 +	1 +	3	7 - 20t	1.5	1.5
C_2	3 +	4 - 10t	1	3	7	2 +	1 +	3 -

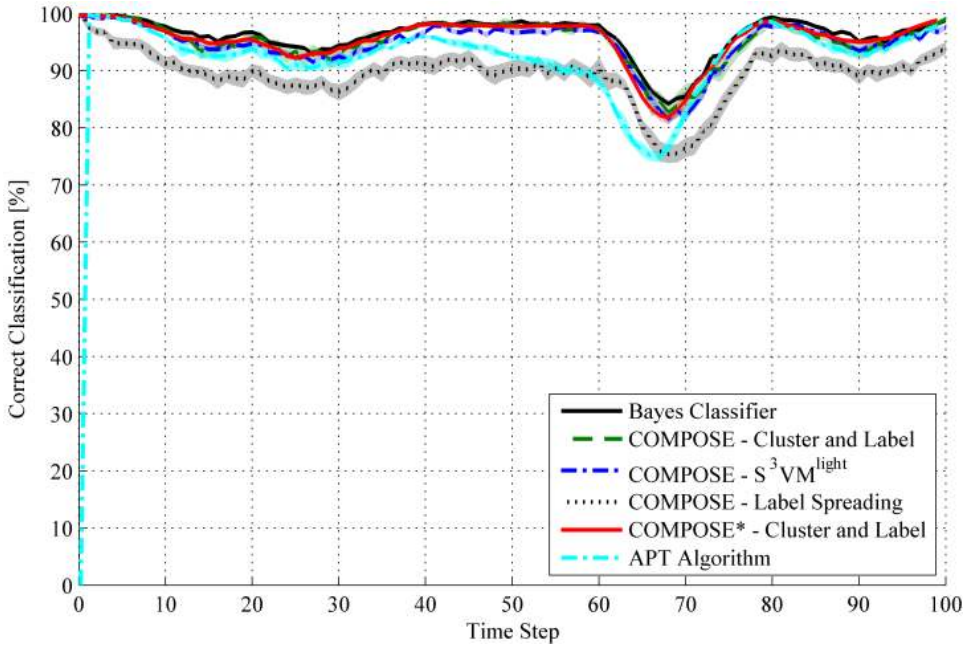

Figure 5.2. Results of unimodal Gaussian experiment

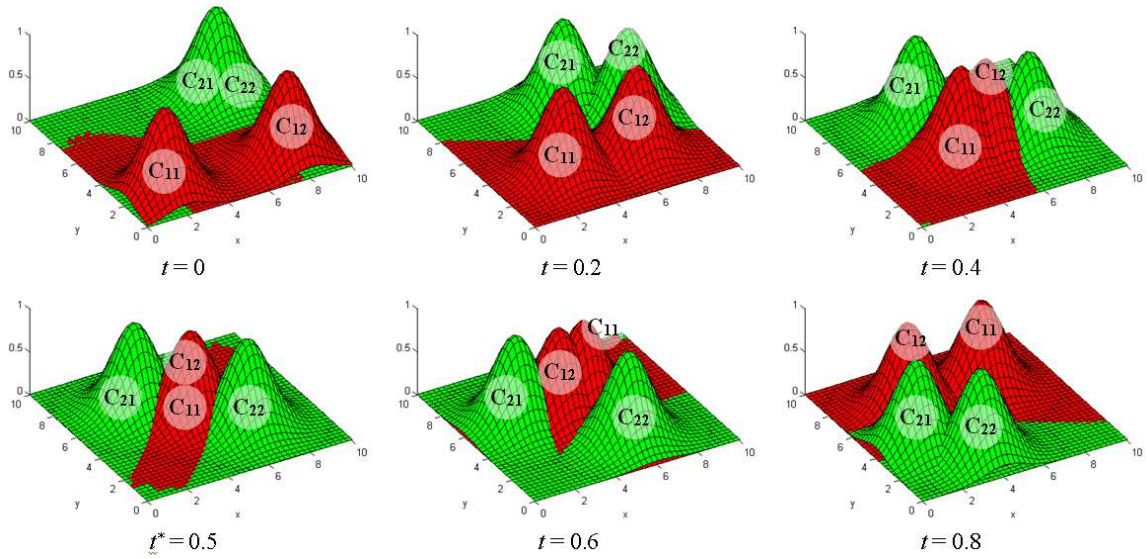

Figure 5.3. Experiment 2 – multimodal Gaussians

Table 5.2.

Parametric equations governing multimodal Gaussian experiment drift

Class	**0 ≤ t < 0.2**				**0.2 ≤ t < 0.4**				**0.4 ≤ t < 0.6**			
	μ_x	μ_y	σ_x	σ_y	μ_x	μ_y	σ_x	σ_y	μ_x	μ_y	σ_x	σ_y
C_{11}	2 +6t	2 + 6t	1	1	3.2 +	3.2 +	1	1	4.4 +	4.4 +	1	1
C_{12}	8 - 6t	2 + 6t	1	1	6.8 -	3.2 +	1	1	5.6 -	4.4 +	1	1
C_{21}	8 - 10t	8	1	1	6 - 10t	8 -	1	1	4 -	7.5-	1	1
C_{22}	8	8 - 10t	1	1	8 -	6 - 10t	1	1	7.5-	4 -	1	1

Class	**0.6 ≤ t < 0.8**				**0.8 ≤ t ≤ 1**			
	μ_x	μ_y	σ_x	σ_y	μ_x	μ_y	σ_x	σ_y
C_{11}	5.6 +	5.6 +	1	1	6.8 +	6.8 +	1	1
C_{12}	4.4 -	5.6 +	1	1	3.2 -	6.8 +	1	1
C_{21}	2.5-	6 - 10t	1	1	2	4 - 10t	1	1
C_{22}	6 - 10t	2.5-	1	1	4 - 10t	2	1	1

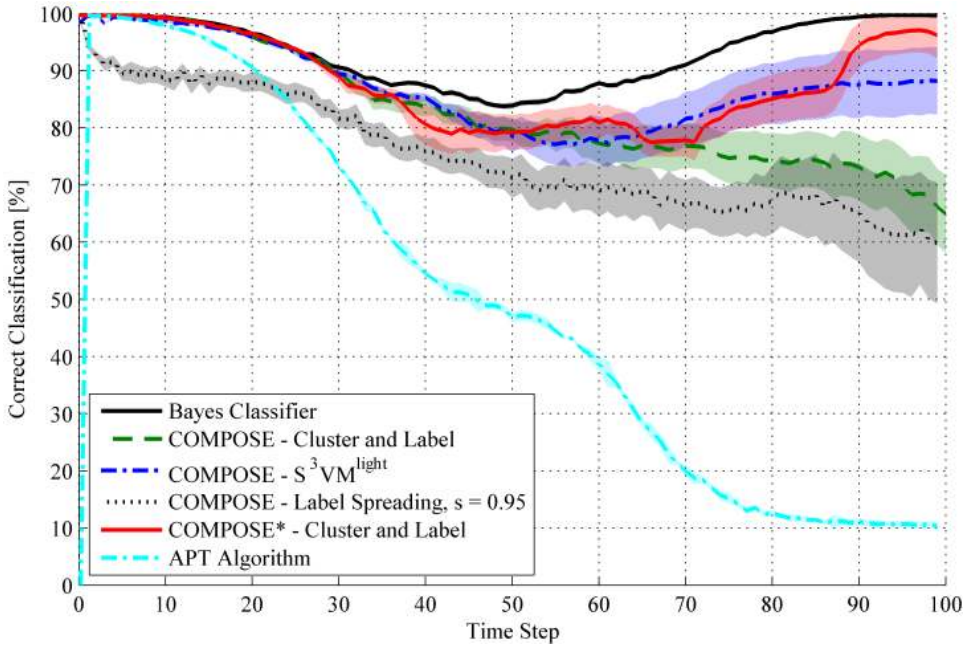

Figure 5.4. Results of multimodal Gaussian experiment

5.1.2 Unimodal Gaussian with added class. One of the new experiments added during testing of version 1.2 initializes two Gaussian distributions at $t = 0$, and then adds a third class at time step 40, as governed by the parametric equations of *Table 5.3*, and as illustrated in *Figure 5.5*. The third class is added with only 5% of its data labeled – with labels provided only during this time step – which constitutes the initialization of the new class for COMPOSE. In contrast, the full training set (i.e., all instances labeled) for the new class is provided to ATP. We also note that the labeled data provided only at this time step comes only from the new class to comply with ILSE assumptions. *Figure 5.6* compares COMPOSE performance against that of APT and Bayes classifier. COMPOSE outperforms APT with statistical significance during time intervals with substantial class overlap (time steps t = 0.2 to 0.6). During other times, the

differences in performances are not statistically significant. All classifiers experience a performance drop when the new class is added, which of course is expected.

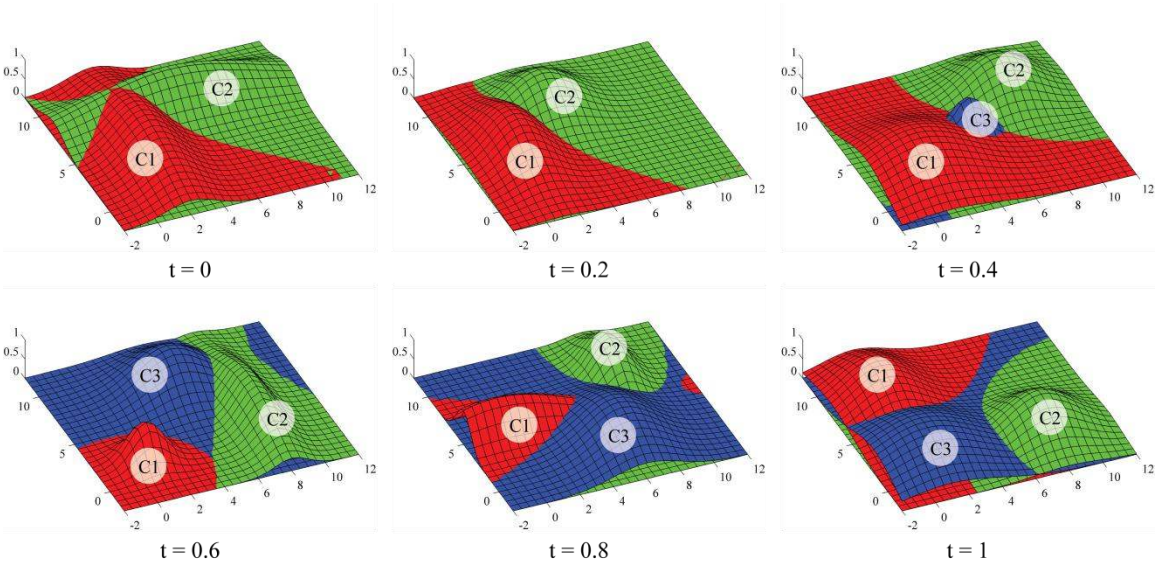

Figure 5.5. Experiment 3 – class added Gaussian

Table 5.3.

Parametric equations governing class added Gaussian experiment drift

Class	$0 \leq t < 0.2$				$0.2 \leq t < 0.4$				$0.4 \leq t < 0.6$			
	μ_x	μ_y	σ_x	σ_y	μ_x	μ_y	σ_x	σ_y	μ_x	μ_y	σ_x	σ_y
C1	2 - 5t	5	1.5	5 - 5t	1	5 - 10t	1.5 + 7.5t	3	1	3 - 5*t	3 - 10t	3 - 10t
C2	5 - 5t	8	5 - 15t	1.5 + 2.5t	4 + 20t	8	2	2	8	8 - 20t	2 - 5t	2 + 10t
C3	n/a	n/a	n/a	n/a	n/a	n/a	n/a	n/a	5	5 + 15t	1 + 5t	1 + 5t

Class	$0.6 \leq t < 0.8$				$0.8 \leq t \leq 1$			
	μ_x	μ_y	σ_x	σ_y	μ_x	μ_y	σ_x	σ_y
C1	1 - 5t	2 + 15t	1 + 15t	1	0 + 5t	5 + 15t	4 - 10t	1 + 10t
C2	8	4 + 20 t	1	4 - 10t	8	8 - 30t	1 + 5t	2
C3	5 + 5t	8 - 30t	2	2 +5t	6 - 25t	2	2 + 5t	3

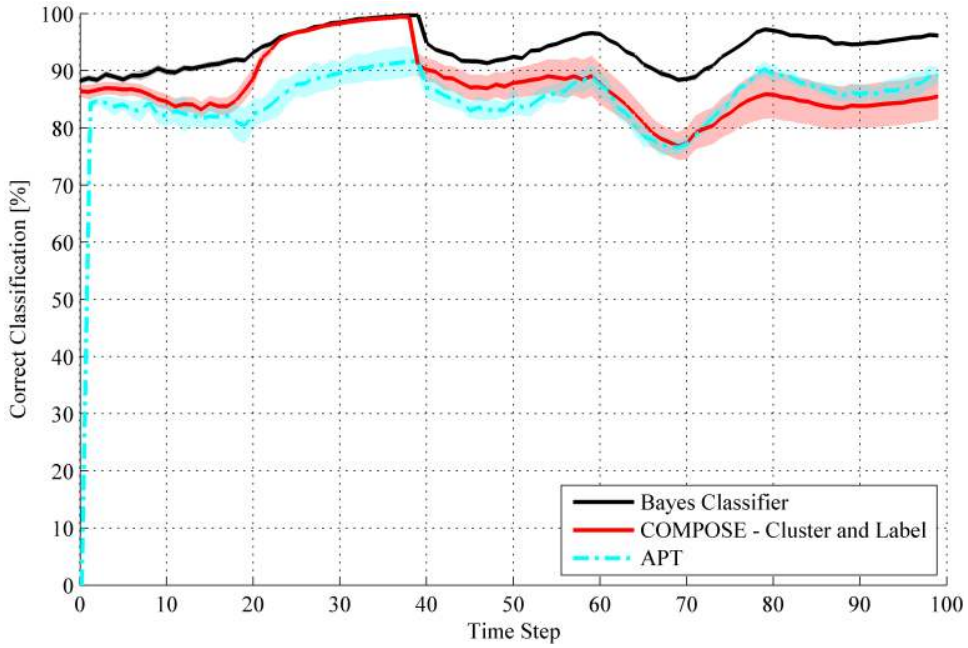

Figure 5.6. Results of class added Gaussian experiment

5.1.3 Unimodal Gaussians in 3D. The other new experiment added during testing of version 1.2, governed by equations of *Table 5.4* and illustrated in *Figure 5.7*, extends the feature space to three dimensions to demonstrate (and graphically illustrate) that revised COMPOSE can actually scale to higher dimensions (also see 8-dimensional real world dataset below). *Figure 5.8* compares COMPOSE's generalization performance to that of Bayes classifier and ATP. The important observation here is that COMPOSE can still follow Bayes extremely well, despite the unfair nature of the experimental setup, and outperforms APT with statistical significance during the more difficult periods of high overlap, and performing comparably during other time steps.

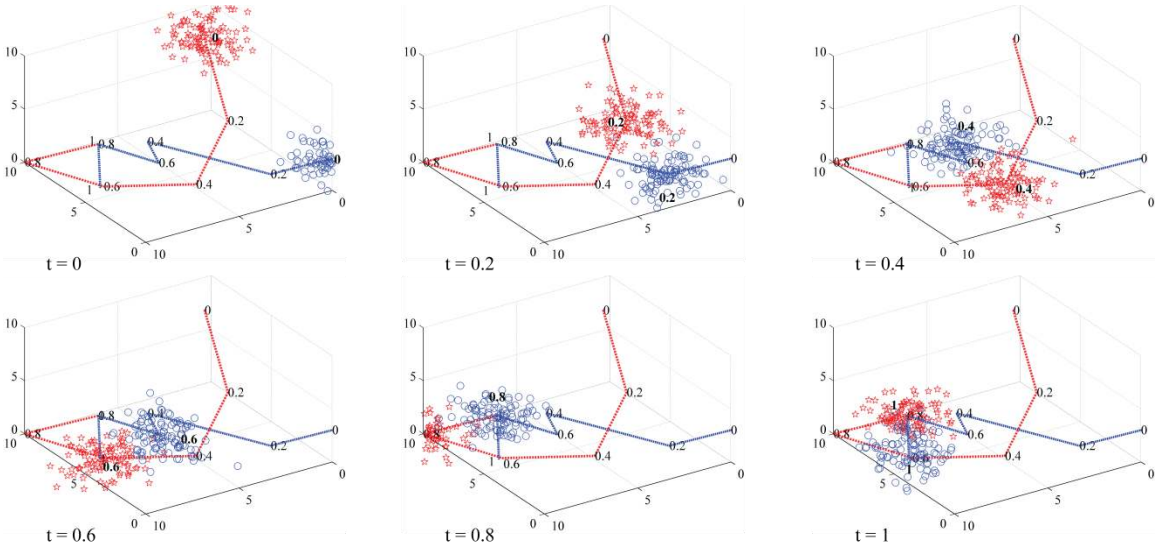

Figure 5.7. Experiment 4 – 3D Gaussians

Table 5.4.

Parametric equations governing 3D Gaussian experiment drift

Class	**$0 \leq t < 0.2$**			**$0.2 \leq t < 0.4$**			**$0.4 \leq t < 0.6$**		
	μ_x	**μ_y**	**μ_z**	**μ_x**	**μ_y**	**μ_z**	**μ_x**	**μ_y**	**μ_z**
C_1	9 -25t	1 + 10t	8 - 15t	4 - 10t	3 + 15t	5 - 15t	2 + 15t	6 + 15t	2 - 5t
C_2	0 + 10t	0 + 10t	3 - 10t	2 + 20t	2 + 20t	1 + 10t	6 - 20t	6 + 10t	3 + 10t

Class	**$0.6 \leq t < 0.8$**			**$0.8 \leq t \leq 1$**		
	μ_x	**μ_y**	**μ_z**	**μ_x**	**μ_y**	**μ_z**
C_1	5 + 25t	9 + 5t	1 - 5t	10 - 15t	10 - 10t	0 + 15t
C_2	2 + 25t	8	5 - 10t	7 - 10t	8 + 6t	3 - 5t

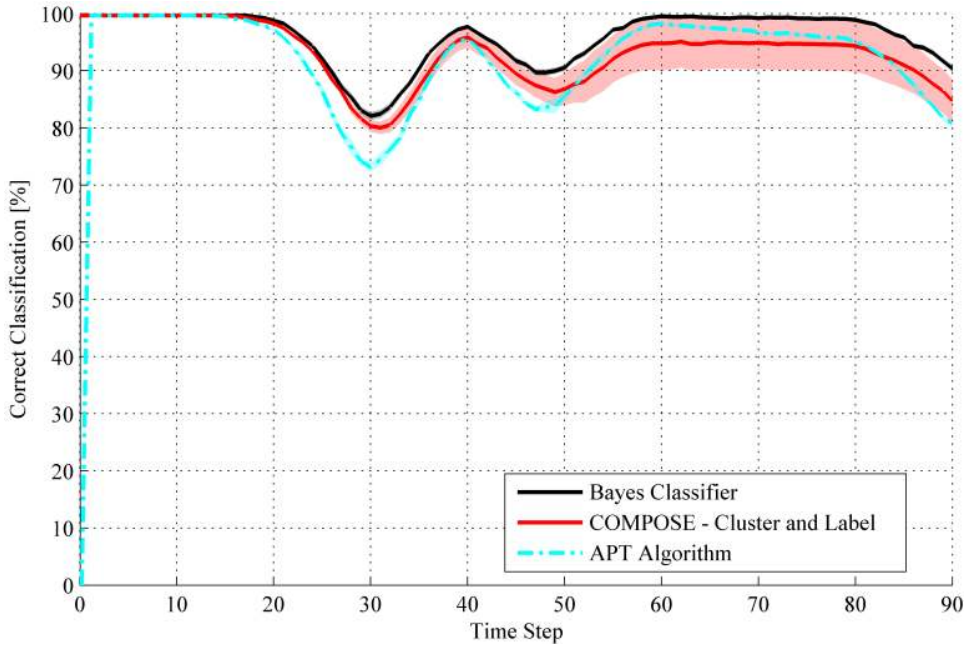

Figure 5.8. Results of 3D Gaussian experiment

5.2 Experimental Setup and Results of Real-World Data

We have also tested the latest version of COMPOSE using the National Oceanic and Atmospheric Administration (NOAA) weather dataset 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, minimum temperature and maximum temperature) were used to determine whether each day experienced rain or no-rain. The dataset contains 18,154 daily readings of which 5,693 are rain and the remaining 12,461 are no-rain. Data was grouped into 49 batches of one year intervals, containing 365 instances (days) each; the remaining data was placed into the 50th batch as a partial year.

This experiment was initialized with 5% of the 365 instances labeled. Every subsequent time step received the full set of additional 365 – all unlabeled – instances.

Since this is real-world data (and not drawn from a distribution), and since all available data are presented at each time step, only one trial is possible. Repeating trials would result in the same performance each time, so a confidence interval cannot be obtained. In Elwell et al.'s recent work [9], this dataset was used to test an ensemble of supervised learners (Learn^{++}.NSE – for **N**on-**S**tationary **E**nvironments) receiving labeled data with every time step in a seasonal fashion – batches of 90 instances. We compare yearly batch performance of COMPOSE and APT with that of Learn^{++}.NSE (with SVM as well as naïve Bayes used as BaseClassifier) in *Figure 5.9*. COMPOSE greatly outperforms APT, but the most compelling demonstration of COMPOSE's performance comes from comparing COMPOSE to Learn^{++}.NSE. COMPOSE trained in an ILSE setting (and with only 18 labeled instances), is competitive with an ensemble of classifiers that are trained in an entirely supervised manner, receiving fully labeled data at every time step.

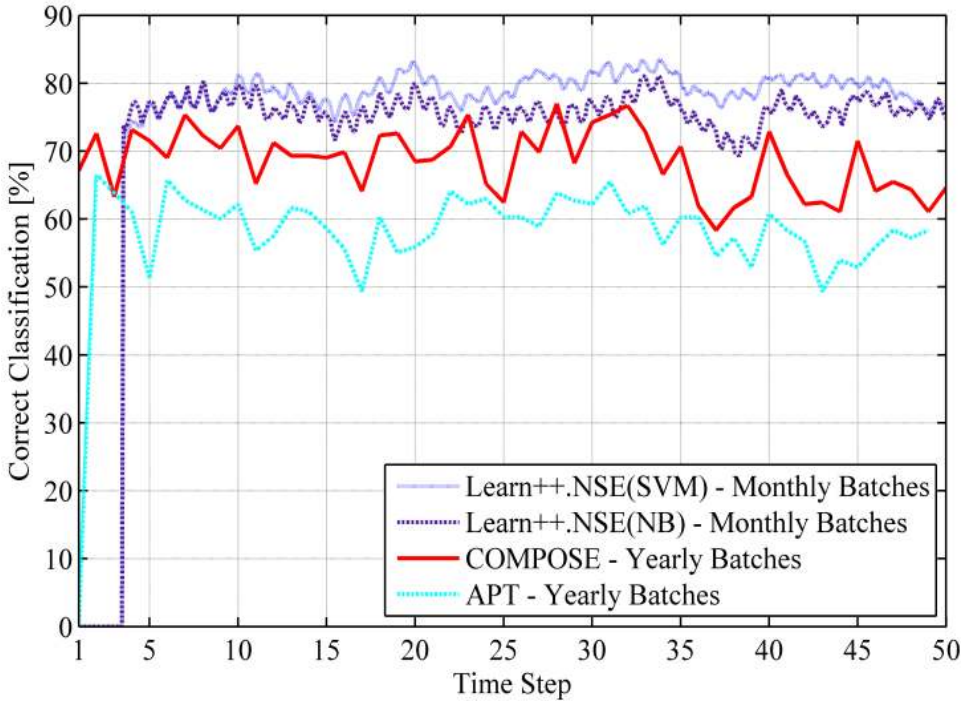

Figure 5.9. Results of NOAA weather dataset

5.3 Computation Time Tests

As the experiments have shown, COMPOSE can learn in an initially labeled streaming nonstationary environment, and successfully track the changing environment using unlabeled data only. The ability of COMPOSE to learn in such a setting comes at a cost: COMPOSE is a relatively computationally expensive algorithm, though not as expensive as APT, at least for the datasets used in our experiments.

The complexity of COMPOSE version 1.2 has in fact been reduced from its original version, where the skeleton algorithm used for compaction was its computationally most expensive module. With the unwrapping compaction utilized in version 1.2, the compaction function is no longer a computational bottle neck – in fact, it is no longer dependent on dimensionality. The most expensive module in COMPOSE is now the α shape generation, which runs in exponential time with respect to the number of

dimensions. We have run some timing experiments, described below, to better understand the behavior of the algorithm with respect to its computational complexity.

Figure 5.10 shows the computation time, averaged over 50 trials for COMPOSE and five trials for ATP, conducted on a modest 2.4 GHz processor (with 6GB RAM) for each of the synthetic experiments described in the previous section. In each case, the timing diagrams follow a similar trend: the initial few time steps are computed relatively quickly while a basis of core supports are built up; then, within a few additional time steps, the algorithm reaches a steady state and maintains approximately the same processing time (per time step) for the remainder of the experiment, unless new classes are added, which then adds a modest additional cost (see change in Unimodal Gaussian Added Class experiment steady state computation time at time step 40).

Comparing the Unimodal Gaussian Experiment (with 100 unlabeled instances added per class, resulting in 200 new instances per time step) and its 2.5s per time step steady state processing time with the Multimodal Gaussian Experiment (with 100 unlabeled instances added for each of the four modes, resulting in 400 new instances per time step) and its 5s per time step steady state processing time further shows that COMPOSE runs in nearly linear time with respect to the cardinality of the data.

Comparing the Unimodal Gaussian Experiment, $\mathtt{CP} = 0.70$, with the Unimodal Added Class Experiment, $\mathtt{CP} = 0.60$, suggests the greater the compaction percentage the faster the algorithm runs, as there are fewer core supports to maintain.

Comparing any of the 2D experiments to the 3D experiment shows that the computation time increases greatly with higher dimensional data. This increase in computational complexity with respect to the dimensionality is the primary cost of the

current algorithm. However, we believe the cost is justified given the difficulty of the task the algorithm seeks to solve. We should note that even with the 8-dimensional data, where processing for each time step takes 20-30 minutes (on a modestly configured computer), COMPOSE is well within useable limits for many applications that generate data less frequently than every 30 minutes. Any application, for example, that generates hourly or daily data can be easily used with current version of COMPOSE even with higher dimensions. Furthermore, we should reemphasize that the primary bottle neck in COMPOSE is not the data cardinality but rather its dimensionality. Therefore, the algorithm can easily handle large databases with modest dimensionality.

It is also worth noting that all computation times mentioned above were obtained using a modestly configured computer running an interpreted language (Matlab). Optimizing the algorithm (many of its steps can be run in parallel), implementing it in a compiled language and running it in a parallel computing setting can further improve its computational efficiency, which is tasked in future work as described in *Chapter 6*.

Comparing computation times of COMPOSE and ATP, *Table 5.5* shows a significant difference. As expensive as COMPOSE is, it completed the synthetic dataset experiments an order of magnitude faster than ATP on the same computer in the same interpreted Matlab environment.

Finally, since the most expensive module in the current version of COMPOSE is the alpha-shape generation – essentially a density estimation algorithm – alternative density estimation approaches such as Gaussian mixture models may further improve the computational efficiency. Evaluating such alternative density estimation approaches is also within the scope of future work proposed in *Chapter 6*.

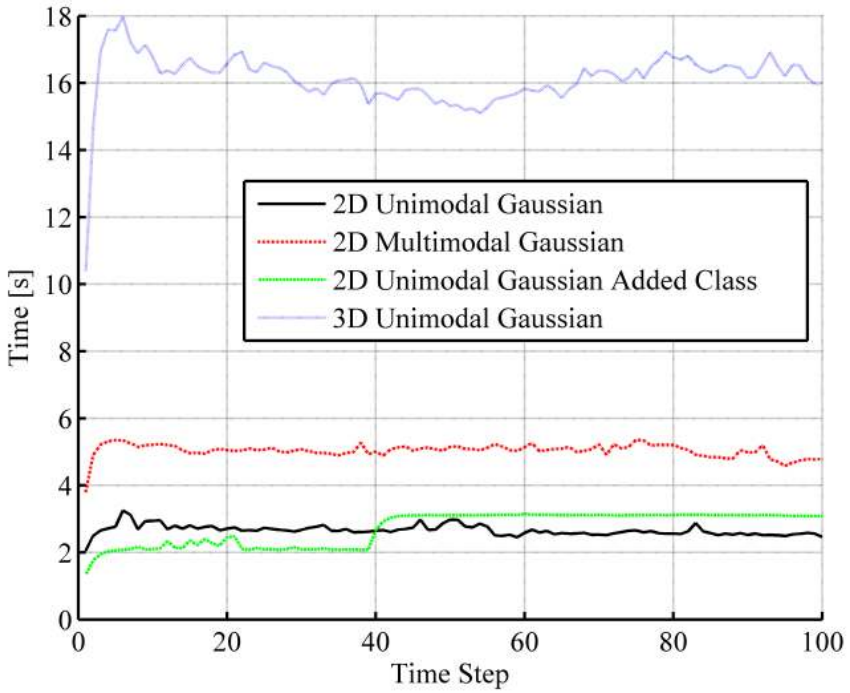

Figure 5.10. Computation time of experiments

Table 5.5.

COMPOSE and APT computation comparison

Dataset	**COMPOSE [minutes]**	**APT [minutes]**
2D Unimodal	4.16	3,600
2D Multimodal	8.33	20,303
2D Unimodal – Class Added	4.33	21,390
3D Unimodal	26.66	22,776

5.4 Choice of Free Parameters and Their Effects

To better understand the impact of each of COMPOSE's free parameters, the α-value and compaction percentage CP, we have repeated the synthetic data experiments varying each parameter independently. We first looked at the effect of CP, keeping α

constant using a family of curves. A sample of these (using the multimodal Gaussian data) is shown in *Figure 5.11*, which indicates that a proper choice of CP is necessary. We also plotted performance keeping CP constant and allowing α-value to vary – whose sample plots are presented in *Figure 5.12* for three different values of CP. These results show that when the compaction percentage is chosen incorrectly, too high as in *Figure 5.12a* or too low as in *Figure 5.12c* – the performance varies greatly with respect to α. However, if CP is chosen properly, as in *Figure 5.12b*, the algorithm performance becomes less sensitive to the α parameter.

From this analysis, we conclude that selecting the compaction percentage correctly has the biggest impact on COMPOSE's performance. There appears to be a logical explanation for this: if α shapes are compacted too much, core supports relevant to the future distribution are lost. If compacted too little, the core supports may overlap with a rival class in the future time step and become misleading.

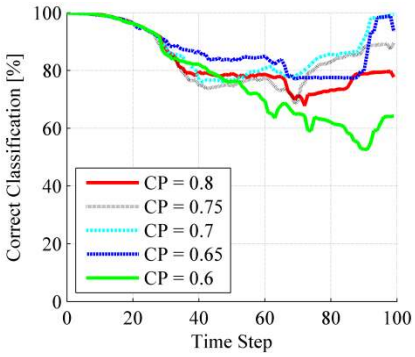

Figure 5.11. Constant α and varied CP Typical family of curves with α-value (α = 0.40 shown) held constant and compaction percentage allowed to vary.

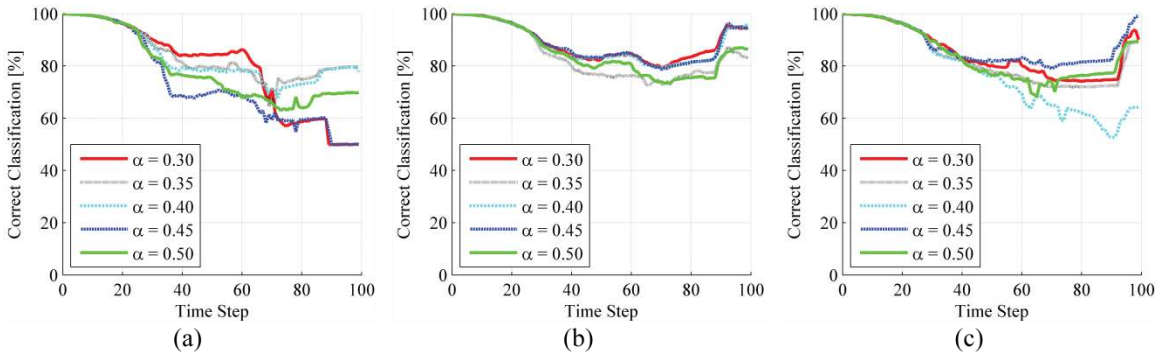

Figure 5.12. Constant CP and varied α

Family of curves with CP held constant and α-value allowed to vary. When CP is too high, e.g., 0.8 as in (a), or too low, e.g., 0.6 as in (c), the algorithm is sensitive to variations in α. When CP is selected close to optimal value, e.g., 0.68 as in (b), the performance variation and the sensitivity to α decreases dramatically.

Chapter 6

Conclusions and Future Work

This thesis introduces and describes COMPOSE, for semi-supervised learning from a nonstationary (drifting) environment experiencing extreme verification latency. In this environment, the nonstationary data, drawn from a drifting distribution, arrive in a streaming manner. Beyond an initial batch, the entire data stream is assumed unlabeled. Our preliminary results have been quite promising, demonstrating that COMPOSE can indeed learn and track the drifting distributions in such a challenging environment.

COMPOSE can track any streaming nonstationary environment as long as the class conditional distributions overlap at subsequent instances. We refer to this condition as limited drift. This is a practically reasonable assumption, as in most natural phenomena – perhaps with the exception of catastrophic or abrupt failures – the changes to the data distribution is usually gradual. One particularly pathological scenario is worth mentioning as an extreme case that violates the limited drift assumption: a sudden change of class labels while data distribution itself remains constant. In such a case there is precisely zero overlap between $p^t(\mathbf{x}|y)$ and $p^{t+1}(\mathbf{x}|y)$. COMPOSE cannot track such a change, since the algorithm receives no future labeled data in the ILSE setting. Toy examples of this scenario include the shifting hyperplane as used in [14], and rotating checkerboard example as used in [9], [57]. We know of no practical example of this scenario. While COMPOSE is guaranteed to track subsequent overlapping distributions, we have noticed the algorithm also performs well when the distributions do not overlap, given the following condition is met – for any given class, its drifted distribution must be closer than any other opposing class's drifted distribution. This observation has not been

validated yet, and is mentioned in future work below, however, intuitively this observation makes sense since most SSL classification is achieved through grouping instances that reside in a similar or close feature domain.

On the other hand, we note that COMPOSE can naturally work in the more relaxed environment, where labeled data are provided regularly or intermittently. In such a case, COMPOSE simply employs the provided labeled data as new core supports to be used in future time steps. COMPOSE can then accommodate the aforementioned change to class membership scenarios, as well as abrupt change scenarios.

Under the ILSE setting, the focus of this paper, preliminary results show that COMPOSE outperforms APT in regions of class overlap, as well as scenarios where data distributions diverge into multiple modes. APT requires all modes to be presented at initialization and further assumes that any drift to the data distribution be structured. Furthermore, while COMPOSE is computationally intensive algorithm, it appears to be more efficient than APT.

Nevertheless, the α-shape construction used by COMPOSE is indeed a computationally expensive process, one that is exponential in dimensionality. Future work includes exploring more efficient ways of constructing α-shapes, or using alternate density estimation techniques, such as Gaussian Mixture Models (GMM) or kernel density estimation. While such changes may require modifications to the compaction method, the foundational concepts of COMPOSE remain the same – select instances from the geometric center (core region) of high density regions of each class to be used as labeled data and combine with the unlabeled data of subsequent time step. This is why we refer to COMPOSE more as a framework, rather than just an algorithm. COMPOSE can

be a family of algorithms, depending on how the core supports are determined, what SSL algorithm is used as a BaseClassifier, or how the compaction is applied.

There is, of course, much room for improvement: articulating a more rigorous definition of limited drift (e.g., defining limited drift with respect to Kullback- Leibler divergence or Hellinger distance between two subsequent distributions), optimizing or automating selection of algorithm parameters, and expanding the experimental work to other real–world and even higher dimensional data, all constitute our current and future work.

Despite its limitations and the aforementioned room for improvement, we believe that COMPOSE shows significant promise in addressing extreme verification latency, performing quite well against other approaches. It is worth mentioning that COMPOSE's limited drift assumption is much less restrictive than those of other algorithms. Perhaps most remarkable is the performance comparison of COMPOSE against the Bayes classifier, and Learn^{++}.NSE (an ensemble of supervised learners). In these experiments, the experimental conditions for comparison were deliberately set to be grossly unfair against COMPOSE, where the competing algorithms were run in a fully supervised mode.

Finally, we should mention that COMPOSE introduces tools from computational geometry that are not often used in machine learning research but may have applications to other machine learning problem domains. We hope that this work will stimulate new discussions and new efforts, and perhaps open computational geometry based approaches to other machine learning problems, where such approaches have been mostly underexplored.

6.1 Summary of Future Work

The work presented in this thesis was the basis for a NSF grant proposal that was later funded. For those that continue work on the COMPOSE framework I have compiled a list of future tasks mentioned throughout this thesis for easy reference. Future works to be considered are:

- Creating a rigorous definition of limited drift with respect to established metrics such as Kullback- Leibler divergence or Hellinger distance.
- More efficient ways of constructing compactable boundary objects (such as α-shapes) by exploring alternative density estimation techniques, such as Gaussian Mixture Models (GMM) or kernel density estimation.
- Implementation and testing of various methods to incorporate receipt of future labeled data if the extreme latency assumption can be relaxed, allowing periodic receipt of labeled batches. When new data are received, does COMPOSE perform better if reinitialized using the only the new labeled data or is there some benefit to retaining core supports established before the arrival of new labeled data?
- Implementing the current version of COMPOSE to maximize its use of parallel processing and explore the decrease in computation time achieved.
- Explore dynamic selection of free parameter of the COMPOSE framework such as the α value or compaction percentage.

Chapter 7

Summary of Contributions

This thesis makes several contributions to the machine learning community, primarily in the fields of nonstationary environments and verification latency. Verification latency still remains a largely underexplored are due to its complexity. However, in our data driven, technologically advancing society this scenario will appear more regularly and will need to be addressed. The COMPOSE framework takes some of the early steps exploring this area of machine learning, showing that learning these environments is possible, albeit presently at a high computational cost. The COMPOSE framework has set the bar demonstrating:

- Semi-supervised learning algorithms are a good classifier selection to tackle nonstationary environments with limited labeled data.
- Given properly selected labeled data the SSL algorithms follow similar classification trends.
- Selecting data at the geometric core of a slowly drifting distribution to propagate information to later drifted distributions works well.

References

[1] S. D. Duda R., Hart P., Pattern Classification, 2nd ed. New York, NY: John Wiley & Sons, 2001, p. 17.

[2] S. B. Stromsten, "Classification Learning from Both Classified and Unclassified Examples," Stanford University, 2002.

[3] X. Zhu, T. Rogers, R. Qian, and C. Kalish, "Humans Perform Semi-supervised Classification Too," in Proceedings of the 22Nd National Conference on Artificial Intelligence (AAAI'07), 2007, pp. 864–869.

[4] J. Gielis, "A generic geometric transformation that unifies a wide range of natural and abstract shapes.," Am. J. Bot., vol. 90, no. 3, pp. 333–338, Mar. 2003.

[5] S. Grossberg, "Nonlinear Neural Networks: Principles, Mechanisms, and Architectures," Neural Networks, vol. 1, no. 1, pp. 17–61, 1988.

[6] C. Alippi and M. Roveri, "Just-in-time adaptive classifiers; part I: detecting nonstationary changes," IEEE Trans. Neural Networks, vol. 19, no. 7, pp. 1145–1153, 2008.

[7] A. Bifet, "Adaptive Learning and Mining for Data Streams and Frequent Patterns," Universitat Politcnica de Catalunya, 2009.

[8] S. Chen and H. He, "Towards incremental learning of nonstationary imbalanced data stream: a multiple selectively recursive approach," Evol. Syst., vol. 2, no. 1, pp. 35–50, 2011.

[9] R. Elwell and R. Polikar, "Incremental Learning of Concept Drift in Nonstationary Environments," IEEE Trans. Neural Networks, vol. 22, no. 10, pp. 1517–1531, Oct. 2011.

[10] H. Haibo, C. Sheng, L. Kang, and X. Xin, "Incremental Learning From Stream Data," Neural Networks, IEEE Trans., vol. 22, no. 12, pp. 1901–1914, Dec. 2011.

[11] J. Z. Kolter and M. A. Maloof, "Dynamic weighted majority: an ensemble method for drifting concepts," J. Mach. Learn. Res., vol. 8, pp. 2755–2790, 2007.

[12] J. Quinonero-Candela, M. Sugiyama, A. Schwaighofer, and N. D. Lawrence, Dataset Shift in Machine Learning. Cambridge, Massachusetts: The MIT Press, 2009.

[13] P. P. Rodrigues, J. Gama, and J. P. Pedroso, “Hierarchical Clustering of Time-Series Data Streams,” Knowl. Data Eng. IEEE Trans., vol. 20, no. 5, pp. 615–627, 2008.

[14] W. N. Street and Y. Kim, “A streaming ensemble algorithm (SEA) for large-scale classification,” in Seventh ACM SIGKDD International Conference on Knowledge Discovery & Data Mining (KDD-01), 2001, pp. 377–382.

[15] A. Tsymbal, M. Pechenizkiy, P. Cunningham, and S. Puuronen, “Dynamic integration of classifiers for handling concept drift,” Inf. Fusion, vol. 9, no. 1, pp. 56–68, Jan. 2008.

[16] G. Widmer and M. Kubat, “Learning in the presence of concept drift and hidden contexts,” Mach. Learn., vol. 23, no. 1, pp. 69–101, 1996.

[17] I. Zliobaite and Z, “Combining similarity in time and space for training set formation under concept drift,” Intell. Data Anal., vol. 15, no. 4/2011, pp. 589–611, 2011.

[18] G. Marrs, R. Hickey, and M. Black, “The impact of latency on online classification learning with concept drift,” Knowl. Sci. Eng. …, 2010.

[19] O. Chapelle, B. Scholkopf, and A. Zien, Semi-Supervised Learning. Cambridge, MA: MIT Press, 2006.

[20] K. Chen and S. Wang, “Semi-Supervised Learning via Regularized Boosting Working on Multiple Semi-Supervised Assumptions,” IEEE Trans. Pattern Anal. Mach. Intell., vol. 33, no. 1, pp. 129–143, Jan. 2011.

[21] A. Dempster, N. Laird, and D. Rubin, “Maximum Likelihood from Incomplete Data via the EM Algorithm,” J. R. Stat. Soc. Ser. B, vol. 39, no. 1, pp. 1–38, 1977.

[22] X.Zhu and A.Goldberg, Introduction to Semi-Supervised Learning. Morgan & Claypool, 2009, pp. 31–32.

[23] T.Joachims, “Transductive Inference for Text Classification using Support Vector Machines,” in Proc. 16th Int. Conf. Machine Learning, 1999, pp. 200–209.

[24] V.Vapnik and A.Sterin, “On structural risk minimization for overall risk in a problem of pattern recognition,” Autom. Remote Control, vol. 10, pp. 1495–1503, 1977.

[25] D.Zhou, O.Bousquet, T.Lal, J.Weston, and B.Scholkopf, “Learning with Local and Global Consistency,” in Advances in Neural Information Processing Systems, no. 16, Cambridge, MA: MIT Press, 2004, pp. 321–328.

[26] X.Zhu and Z.Ghahramani, “Learning from Labeled and Unlabeled Data with Label Propagation,” Carnegie Mellon University, Pittsburgh, PA, 2002.

[27] J. C. Schlimmer and R. H. Granger, “Incremental learning from noisy data,” Mach. Learn., vol. 1, no. 3, pp. 317–354, Sep. 1986.

[28] D. P. Helmbold and P. M. Long, “Tracking drifting concepts by minimizing disagreements,” Mach. Learn., vol. 14, no. 1, pp. 27–45, 1994.

[29] J. Case, S. Jain, S. Kaufmann, A. Sharma, and F. Stephan, “Predictive learning models for concept drift,” Theor. Comput. Sci., vol. 268, no. 2, pp. 323–349, Oct. 2001.

[30] M. B. Harries, C. Sammut, and K. Horn, “Extracting hidden context,” Mach. Learn., vol. 32, no. 2, pp. 101–126, 1998.

[31] I. Zliobaite, “Identifying hidden contexts in classification,” in 15th Pacific-Asia Conference on Knowledge Discovery and Data Mining, PAKDD 2011, May 24, 2011 - May 27, 2011, 2011, vol. 6634 LNAI, no. PART 1, pp. 277–288.

[32] T. Joachims, “Detecting concept drift with support vector machinesA1 - Klinkenberg,Ralf,” in 17th International Conference on Machine Learning, 2000, pp. 487–494.

[33] R. Klinkenberg, “Learning drifting concepts: example selection vs. example weighting,” Intell. Data Anal. Spec. Issue Increm. Learn. Syst. Capab. Deal. with Concept Drift, vol. 8, no. 3, pp. 281–300, 2004.

[34] G. Hulten, L. Spencer, and P. Domingos, “Mining time-changing data streams,” in Proceedings of the seventh ACM SIGKDD international conference on Knowledge discovery and data mining, 2001, pp. 97–106.

[35] M. Nunez, R. Fidalgo, and R. Morales, “Learning in Environments with Unknown Dynamics: Towards more Robust Concept Learners,” J. Mach. Learn. Res., vol. 8, pp. 2595–2628, 2007.

[36] P. Wang, H. Wang, X. Wu, W. Wang, and B. Shi, “A Low-Granularity Classifier for Data Streams with Concept Drifts and Biased Class Distribution,” IEEE Trans. Knowl. Data Eng., vol. 19, no. 9, pp. 1202–1213, 2007.

[37] J. Gama, P. Medas, G. Castillo, and P. Rodrigues, “Learning with drift detection,” in Advances in Artificial Intelligence - SBIA 2004, 2004, vol. 3171, pp. 286–295.

[38] C. Alippi and M. Roveri, “Just-in-Time Adaptive Classifiers;Part II: Designing the Classifier,” IEEE Trans. Neural Networks, vol. 19, no. 12, pp. 2053–2064, Dec. 2008.

[39] C. Alippi, G. Boracchi, and M. Roveri, “Just in time classifiers: managing the slow drift case,” in International Joint Conference on Neural Networks (IJCNN 2009), 2009, pp. 114–120.

[40] C. Alippi, G. Boracchi, and M. Roveri, “Change Detection Tests Using the ICI Rule,” in IEEE International Joint Conference on Neural Networks (IJCNN 2010), 2010, pp. 1190–1196.

[41] S. Hoeglinger and R. Pears, “Use of Hoeffding trees in concept based data stream mining,” in International Conference on Information and Automation for Sustainability (CIAFS 2007), 2007, pp. 57–62.

[42] P. Vorburger and A. Bernstein, “Entropy-based Concept Shift Detection,” in Sixth International Conference on Data Mining (ICDM ’06.), 2006, pp. 1113–1118.

[43] H. Abdulsalam, D. Skillicorn, and P. Martin, “Classification Using Streaming Random Forests,” IEEE Trans. Knowl. Data Eng., vol. 23, no. 1, pp. 22–36, 2011.

[44] G. Ditzler and R. Polikar, “Hellinger distance based drift detection for nonstationary environments,” in Computational Intelligence in Dynamic and Uncertain Environments (CIDUE), 2011 IEEE Symposium on, 2011, pp. 41–48.

[45] T. R. Hoens, N. V Chawla, and R. Polikar, “Heuristic Updatable Weighted Random Subspaces for Non-stationary Environments,” in 11th IEEE International Conference on Data Mining (ICDM 2011), 2011, pp. 241–250.

[46] C. J. Tsai, C. I. Lee, and W. P. Yang, “Mining decision rules on data streams in the presence of concept drifts,” Expert Syst. Appl., vol. 36, no. 2, Part 1, pp. 1164–1178, Mar. 2009.

[47] L. Cohen, G. Avrahami-Bakish, M. Last, A. Kandel, and O. Kipersztok, “Real-time data mining of non-stationary data streams from sensor networks,” Inf. Fusion, vol. 9, no. 3, pp. 344–353, 2008.

[48] L. Cohen, G. Avrahami, M. Last, and A. Kandel, “Info-fuzzy algorithms for mining dynamic data streams,” Appl. Soft Comput., vol. 8, no. 4, pp. 1283–1294, Sep. 2008.

[49] R. French, “Catastrophic forgetting in connectionist networks,” Trends Cogn. Sci., vol. 3, no. 4, pp. 128–138, 1999.

[50] A. Bifet, G. Holmes, B. Pfahringer, R. Kirkby, and R. Gavalda, “New Ensemble Methods For Evolving Data Streams,” in Knowledge and Data Discovery (KDD 2009), 2009, pp. 139–148.

[51] J.Kolter and M.Maloof, “Dynamic Weighted Majority: An Ensemble Method for Drifting Concepts,” J.Mach.Learning Res., vol. 8, pp. 2755–2790, 2007.

[52] J. Gao, W. Fan, and J. Han, “On appropriate assumptions to mine data streams: analysis and practice,” in International Conference on Data Mining, 2007, pp. 143–152.

[53] M. Masud, J. Gao, L. Khan, J. Han, and B. Thuraisingham, “Classification and Novel Class Detection in Concept-Drifting Data Streams under Time Constraints,” IEEE Trans. Knowl. Data Eng., vol. 23, no. 6, pp. 859–874, 2011.

[54] H. He and S. Chen, “IMORL: Incremental Multiple-Object Recognition and Localization,” IEEE Trans. Neural Networks, vol. 19, no. 10, pp. 1727–1738, 2008.

[55] A. Bifet, E. Frank, G. Holmes, and B. Pfahringer, “Accurate ensembles for data streams: Combining restricted Hoeffding trees using stacking.,” in 2nd Asian Conference on Machine Learning, 2010, vol. 13.

[56] A. Bifet, “MOA: Massive Online Analysis,” http://moa.cs.waikato.ac.nz. 30-Dec-2010.

[57] R. Elwell and R. Polikar, “Incremental learning in nonstationary environments with controlled forgetting,” in International Joint Conference on Neural Networks (IJCNN 2009)., 2009, pp. 771–778.

[58] R. Elwell and R. Polikar, “Incremental learning of variable rate concept drift,” in International Workshop on Multiple Classifier Systems, 2009, vol. 5519, pp. 142–151.

[59] X. Li, Peipei; Wu, Xindong; Hu, “Mining Recurring Concept Drifts with Limited Labeled Streaming Data,” in Proceedings of the 2nd Asian Conference on Machine Learning (ACML2010), 2010, pp. 241–252.

[60] G. Ditzler and R. Polikar, “Semi-supervised learning in nonstationary environments,” in Int. Joint Conf. on Neural Networks (IJCNN 2011), 2011, pp. 2741–2748.

[61] M. Masud, J. Gao, L. Khan, J. Han, and B. Thuraisingham, “A Practical Approach to Classify Evolving Data Streams: Training with Limited Amount of Labeled Data,” in IEEE 8th Int. Conf. Data Mining (ICDM ’08), 2008, pp. 929–934.

[62] P. Zhang, X. Zhu, and L. Guo, “Mining Data Streams with Labeled and Unlabeled Training Examples,” in IEEE 9th Int. Conf. Data Mining (ICDM ’09), 2009, pp. 8627–636.

[63] K. Bennett and A. Demiriz, “Semi-Supervised Support Vector Machines,” in Proceedings of Neural Information Processing Systems, 1999, pp. 368–374.

[64] P. Zhang, X. Zhu, J. Tan, and L. Guo, “Classifier and Cluster Ensembles for Mining Concept Drifting Data Streams,” in IEEE 10th Int. Conf. Data Mining (ICDM ’10), 2010, pp. 1175–1180.

[65] G. Krempl, “The Algorithm APT to Classify in Concurrence of Latency and Drift,” in Advances in Intelligent Data Analysis X SE - 22, vol. 7014, J. Gama, E. Bradley, and J. Hollmén, Eds. Springer Berlin Heidelberg, 2011, pp. 222–233.

[66] M.Teichmann and M.Capps, “Surface Reconstruction with Anisotropic Density-Scaled Alpha Shapes,” in Proc IEEE Visualization, 1998, pp. 67–72.

[67] C. B. Barber, D. P. Dobkin, and H. Huhdanpaa, “The quickhull algorithm for convex hulls,” ACM Trans. Math. Softw., vol. 22, no. 4, pp. 469–483, Dec. 1996.

[68] D. Pedoe, Circles: A Mathematical View, vol. 2. Dover Publications, 1979.

[69] H.Edelsbrunner and E.Mucke, “Three-Dimensional Alpha Shapes,” in Assoc.Computing Machinery Trans.Graph., 1994, vol. 13, no. 1, pp. 43–72.

[70] O.Aichholzer, D.Alberts, F.Aurenhammer, and B.Gaertner, “Straight skeletons of simple polygons,” in Proc. 4th Int. Symp. LIESMARS, 1995, pp. 114–124.

[71] K. B. Dyer and R. Polikar, “Semi-supervised learning in initially labeled non-stationary environments with gradual drift,” in The 2012 International Joint Conference on Neural Networks (IJCNN 2012), 2012, pp. 1–9.

[72] K. Dyer, R. Capo, and R. Polikar, “COMPOSE: A Semi-Supervised Learning Framework for Initially Labeled Non-Stationary Streaming Data,” IEEE Trans. Neural Networks Learn. Syst. Spec. issue Learn. Nonstationary Dyn. Environ., 2014.